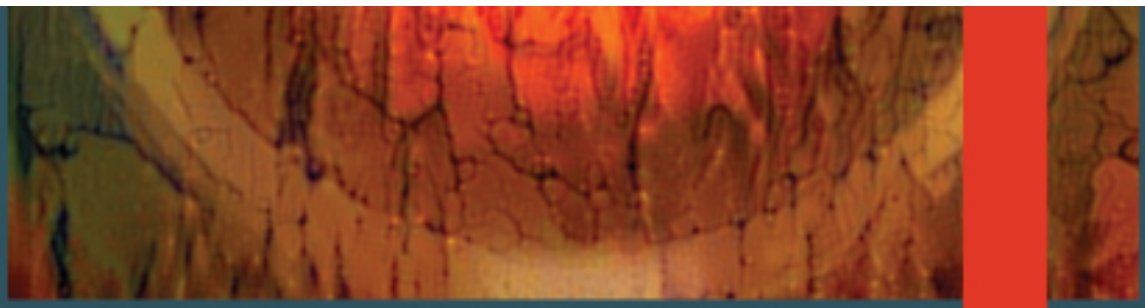
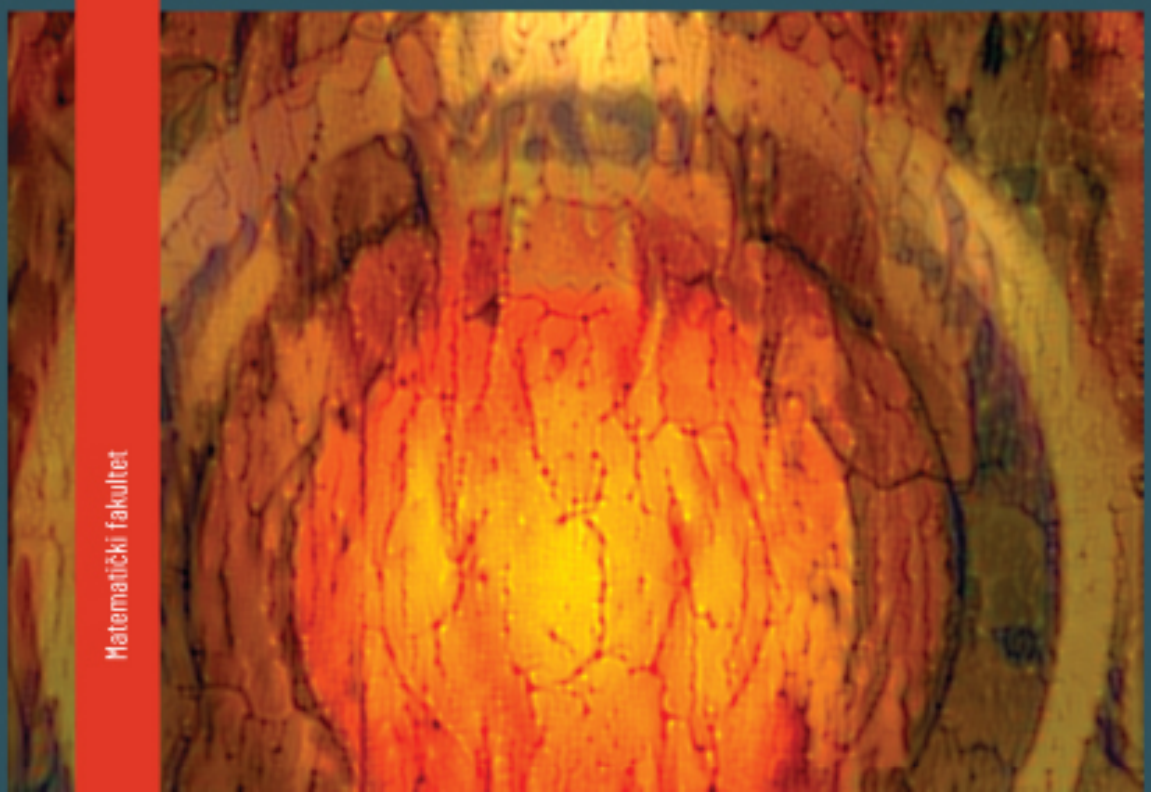


Predrag Janičić
Mladen Nikolić

Veštačka inteligencija



Matematički fakultet

Predrag Janičić Mladen Nikolić

VEŠTAČKA INTELIGENCIJA

Četvrto izdanje

Beograd
2024.

Autori:
dr Predrag Janičić, redovni profesor na Matematičkom fakultetu u Beogradu
dr Mladen Nikolić, vanredni profesor na Matematičkom fakultetu u Beogradu

VEŠTAČKA INTELIGENCIJA

Izdavač: Matematički fakultet Univerziteta u Beogradu
Studentski trg 16, 11000 Beograd
Za izdavača: *prof. dr Zoran Rakić*, dekan

Recenzenti:
dr Vesna Marinković, docent na Matematičkom fakultetu u Beogradu
dr Predrag Tadić, docent na Elektrotehničkom fakultetu u Beogradu

Obrada teksta i ilustracije: *autori* (osim za slike nabrojane na kraju knjige)
Dizajn korica: *Aleksandra Jovanić*
Slika na naslovnoj strani: *alati veštačke inteligencije*

CIP - Каталогизација у публикацији
- Народна библиотека Србије, Београд

004.8(075.8)
ЈАНИЧИЋ, Предраг, 1968-
Veštačka inteligencija / Predrag Janičić, Mladen Nikolić ; [ilustracije autori]. - Beograd : Univerzitet u Beogradu, Matematički fakultet, 2021 (Beograd : Skripta internacional). - 352 стр. : ilustr. ; 24 cm
Tiraž 100. - Bibliografija: str. 351-352.
ISBN 978-86-7589-148-2
1. Николић, Младен, 1981- [аутор] [илустратор]
a) Вештачка интелигенција
COBISS.SR-ID 34371337

Sadržaj

Predgovor

Ova knjiga nastala je na osnovu beleški koje prate predavanja za predmet *Veštačka inteligencija* (na trećoj godini smera *Informatika* na Matematičkom fakultetu Univerziteta u Beogradu) koje smo držali akademskih godina od 2007/08 do 2018/19 (prvi autor) i 2019/20, kao i vežbe od 2007/08 do 2011/12 (drugi autor). Dugi niz godina tražili smo, zajedno sa svojim studentima, najbolje načine da predstavimo ključna polja veštačke inteligencije. Nadamo da izbor sadržaja i način prezentovanja mogu da budu zanimljivi ne samo studentima, već i svima drugima koje interesuje ova oblast računarstva.

Zahvaljujemo recenzentima, Vesni Marinković i Predragu Tadiću na detaljnim komentarima i mnoštvu sugestija koji su značajno doprineli kvalitetu knjige. Zahvaljujemo i drugim kolegama koji su nam dali dragocene komentare i savete, pre svega Filipu Mariću, Miroslavu Mariću i Dušku Vitasu. Zahvaljujemo na veoma pažljivom čitanju, ispravkama i komentarima i studentima koji su slušali predmet *Veštačka inteligencija*, pre svega Nemanji Mićoviću, Mladenu Canoviću, Jani Protić, Vojislavu Stankoviću, Nikoli Premčevskom, Nikoli Ajzenhameru, Neveni Marković, Uni Stanković, Jeleni Simović, Nikoli Dimitrijeviću, Aleksandru Mladenoviću, Vladimiru Simonovskom, Petru Vukmiroviću, Danielu Doži, Ivanu Baleviću, Milanu Kovačeviću, Nebojši Ložnjakoviću, Milošu Samardžiji, Vojislavu Grujiću, Nemanji Antiću, Denisu Aličiću, Miodragu Radojeviću, Dalmi Beara, Mateji Marjanoviću, Đorđu Nemetu, Ognjenu Milinkoviću, Milošu Petričkoviću, Maji Gavrilović i Lazaru Perišiću. Zahvaljujemo i kolegi Petru Veličkoviću na čijim slikama su zasnovane dve slike u delu o mašinskom učenju. Preostale ilustracije napravili smo mi (koristeći program GCLC), izuzev onih slika koje ilustruju konkrente primene i za koje su navedeni izvori.

Zahvaljujemo Aleksandri Jovanić koja je napravila dizajn korica i u njega veštom ljudskom rukom uklopila sliku kreiranu primenom tehnika veštačke inteligencije. Osnovni motivi na toj slici su „oko“ računara HAL iz čuvenog filma „Odiseja u svemiru 2001“ (koji je česta personifikacija sistema veštačke inteligencije) i bik sa zidova pećine Altamire (jedan od najstarijih sačuvanih ljudskih artefakata), čiji je vizuelni stil preuzet algoritmom neuronskog prenosa stila.

U pripremi predavanja i ove knjige koristili smo nekoliko znamenitih knjiga o veštačkoj inteligenciji, ali, još mnogo više, mnoštvo drugih izvora koje bi bilo teško pobrojati – često su to bili materijali ili samo pojedinačni komentari sa interneta o specifičnim osobinama algoritama opisanih u knjizi, a često i naša lična iskustva proistekla iz praktične primene algoritama. Zbog toga, spisak literature naveden na kraju knjige treba shvatiti uslovno – tu su pobrajani samo najčešće korišćeni izvori.

Ova knjiga dostupna je (besplatno) u elektronskom obliku preko naših internet strana, što predstavlja naš doprinos korpusu besplatnih i otvorenih sadržaja.

Bićemo zahvalni čitaocima na svim ispravkama, sugestijama i komentarima koje nam pošalju.

Predrag Janičić i Mladen Nikolić

Beograd, mart 2021.

Predgovor drugom izdanju

U drugom izdanju otklonjene su otkrivene greške i dodato je nekoliko objašnjenja, primera i ilustracija. Na novom pažljivom čitanju i mnoštvu novih komentara i sugestija zahvaljujemo koleginici Vesni Marinković. Na komentarima zahvaljujemo i asistentu Denisu Aličiću, kao i studentima Lazaru Perišiću, Milošu Petričkoviću i Martinu Župčiću.

Bićemo i dalje zahvalni čitaocima na svim ispravkama, sugestijama i komentarima koje nam pošalju.

Predrag Janičić i Mladen Nikolić

Beograd, februar 2022.

Predgovor trećem izdanju

U trećem izdanju otklonjene su otkrivene, uglavnom sitne, greške i dodato je nekoliko objašnjenja. Na komentarima zahvaljujemo studentima Dušanu Mirčiću i Vukašinu Markoviću. Bićemo i dalje zahvalni čitaocima na svim ispravkama, sugestijama i komentarima koje nam pošalju.

I ovo izdanje knjiga dostupno je (besplatno) u elektronskom obliku preko naših internet strana.

Predrag Janičić i Mladen Nikolić

Beograd, februar 2023.

Predgovor četvrtom izdanju

U trećem izdanju otklonjene su otkrivene, uglavnom sitne, greške i dodato je nekoliko objašnjenja i primera. Bićemo i dalje zahvalni čitaocima na svim ispravkama, sugestijama i komentarima koje nam pošalju.

I ovo izdanje knjiga dostupno je (besplatno) u elektronskom obliku preko naših internet strana.

Predrag Janičić i Mladen Nikolić

Beograd, februar 2024.

Glava 1

Uvod

Veštačka inteligencija jedna je od retkih oblasti nauke o kojoj skoro svi – i eksperti i oni koji to nisu – imaju neki stav. Neko smatra da ona donosi velike koristi, neko smatra da od nje prete opasnosti, a neko veruje i u jedno i u drugo. Neobično je onda da, s druge strane, ne postoji opšta saglasnost o tome šta je uopšte veštačka inteligencija i čime se ona bavi. Pod inteligencijom se obično podrazumeva sposobnost usvajanja, pamćenja i obrade određenih znanja. Iako postoji i shvatanje po kojem je centralni cilj veštačke inteligencije oponašanje ljudske inteligencije, većina podoblasti veštačke inteligencije ima drugačiji cilj. Obično je to rešavanje problema u kojima se javlja *kombinatorna eksplozija*, tj. u kojima je broj mogućnosti toliko veliki da se ne može sistematično ispitati u razumnom vremenu. Najveći deo zadataka veštačke inteligencije može se opisati u terminima algoritmike, pretrage, deduktivnog i induktivnog zaključivanja, i drugih preciznih matematičkih pojmova. Tek mali deo istraživača bavi se metodama koje pretenduju da dostignu opšte rasuđivanje u stilu čoveka.

Veštačka inteligencija sastoji se od više podoblasti, naizgled slabo povezanih svojim sadržajem. Pitanje je onda šta je to zajedničko za njih, sem to što se uglavnom sve bave problemima u kojima se javlja mnoštvo mogućnosti koje se ne mogu ispitati sistematičnom pretragom. Postoji više pristupa koji objedinjuju sve podoblasti veštačke inteligencije u jedan, jedinstveni okvir. Jedan takav je pristup zasnovan na agentima kao, na primer, u uticajnoj knjizi „Veštačka inteligencija: moderni pristup“ Rasela (Stuart Russell) i Norviga (Peter Norvig). U tom pristupu, sve komponente problema koji se rešava objašnjavaju se u terminima agenata koji imaju neke akcije na raspolaganju. Drugi pristup, korišćen i u ovoj knjizi, centralno mesto daje *procesu rešavanja problema*. Pri tome, kada se govori o „problemu“, obično se misli na čitavu klasu srodnih zadataka — na primer, na određivanje najkraćeg puta između dva čvora grafa, ili na ispitivanje zadovoljivosti iskazne formule, ili na prepoznavanje lica na fotografiji. Pojedinačne zadatke koji pripadaju ovakvim klasama zovemo *instance problema* (ili *primerci problema*) (na primer, određivanje najkraćeg puta između konkretna dva čvora konkretnog grafa ili ispitivanje zadovoljivosti konkretne formule). Za skoro sve metode veštačke inteligencije zajednička je opšta struktura procesa rešavanja problema. Faze rešavanja obično su: modelovanje, tj. opisivanje zadatog problema u strogim, matematičkim terminima; rešavanje problema opisanog u matematičkim terminima; interpretiranje i analiza rešenja. Dublja priroda ovih faza razlikuje se između različitih podoblasti veštačke inteligencije.

1.1 Kratka istorija veštačke inteligencije

Veštačka inteligencija kao samostalna informatička disciplina prvi put je promovisana na znamenitoj konferenciji *The Dartmouth Summer Research Conference on Artificial Intelligence* organizovanoj u Dartmutu (Sjedinjene Američke Države), 1956. godine. Tom prilikom predloženo je, od strane Džona Makartija, i sâmo ime discipline, ne sasvim srećno sročeno, jer baš je ono često izazivalo nedoumice i podozrenje. Konferencija je trajala mesec dana i bila je pre svega usmerena ka profilisanju nove oblasti. Konferenciju su organizovali Džon Makarti (John McCarthy), Marvin Minski (Marvin Minsky), Natanijel Ročester (Nathaniel Rochester) i Klod Šenon (Claude Shannon). U početnim godinama, pa i decenijama, nizali su se uspesi nove oblasti, a očekivanja su rasla još i brže – ranih šezdesetih, na primer, „mislilo se da je nešto poput računarskog vida (prepoznavanja oblika na slikama) pogodno za letnji projekat studenta master studija“. Minski je 1967. godine izjavio da će problem kreiranja veštačke inteligencije (u stilu ljudske inteligencije) biti rešen u okviru jedne generacije. Ta velika očekivanja ovekovečena su već šezdesetih i sedamdesetih godina u naučno-fantastičnim filmovima i knjigama u kojima računar nadmašuje čoveka u svim intelektualnim aktivnostima. Veštačka inteligencija postala je i polje preko kojeg se stremilo nadmoći u tadašnjem hladnom ratu između Zapada i Istoka. Osamdesetih godina počelo je da dominira shvatanje da su očekivanja od veštačke inteligencije bila previsoka. Fondovi za istraživanja počeli su da se smanjuju i nastupila je takozvana „zima veštačke inteligencije“. Oblast je, pomalo

usporeno, ipak preživela krizu da bi krajem devedesetih godina dvadesetog veka ponovo privukla pažnju čitavog sveta: računari su pobedili svetskog prvaka u šahu i dokazali teoreme koje čovek nije ranije uspevao da dokaže. Bilo je i drugih uspeha, da bi se u dvadeset prvom veku javio talas izuzetno uspešnih sistema zasnovanih na „dubokom učenju“ – sistema koji su uspešno vršili prepoznavanje lica na fotografijama, prevođenje prirodnih jezika, navigaciju vozila i drugo. Veštačka inteligencija danas je skoro sveprisutna, a opšte je uverenje da će se ta sveprisutnost u budućnosti dalje proširivati i produbljivati.

1.2 Znanje i zaključivanje

Za pojam inteligencije suštinske su dve komponente: znanje i zaključivanje. Komponenta zaključivanja predstavlja takođe neku vrstu znanja – to je znanje (koje se naziva i meta-znanjem) o procesu izvođenja novog znanja iz raspoloživog znanja. Pod znanjem podrazumevamo i istinite, potvrđene činjenice, ali i hipoteze, nepotpune informacije i informacije date sa određenim verovatnoćama. Zaključivanje može biti deduktivno – zasnovano na rigoroznim opštim pravilima čijom primenom se dobijaju nove konkretne činjenice. Zaključivanje može biti i induktivno – u njemu se na osnovu mnoštva činjenica pokušava izvođenje opštih pravila. Postoje i druge forme i drugi okviri zaključivanja.

1.3 Simbolički i statistički zasnovana veštačka inteligencija

Mnoge metode veštačke inteligencije zasnovane su na simboličkoj reprezentaciji: i problem i algoritam rešavanja opisani su eksplicitno, a osobine algoritma mogu se analizirati rigorozno, matematički. I opis problema i algoritam su vrlo specifični, prilagođeni jednom konkretnom zadatku i obično se teško uopštavaju. Ti eksplicitni opisi obično se daju u terminima teorije grafova ili matematičke logike. Za ove metode često se kaže da su GOFAI (od engleskog „Good Old-Fashioned Artificial Intelligence“ – „dobra stara veštačka inteligencija“) i čine takozvani „prvi talas“ veštačke inteligencije. One se najčešće oslanjaju na deduktivno zaključivanje.

Od 2010-ih godina, novi moćni računari omogućili su povratak neuronskih mreža (koje su razvijane još 1950-ih), kroz duboko učenje i ostvarili fantastične rezultate u mnogim poljima, kao što je računarski vid, automatsko prevođenje, automatsko upravljanje vozilima, igranje strateških igara, itd. Ovi rezultati zasnovani su na statistici i mašinskom učenju, tj. na induktivnom zaključivanju. U njima nema eksplicitnog opisivanja procesa rešavanja konkretnih primeraka problema, već se koriste meta-algoritmi kojima se, koristeći raspoložive podatke, kreiraju algoritmi za rešavanje konkretnih problema. Ovakvi sistemi obično nisu u stanju da ponude i neko objašnjenje za rešenja koje nude. U primenama je glavna uloga čoveka u fazi modelovanja problema i pripremi podataka za obučavanje. O ovim algoritmima i njihovim osobinama znatno je teže formalno rasuđivati nego o algoritmima koji se zasnivaju na eksplicitnom opisu problema, ali je moguće izvesti statističke ocene njihovog kvaliteta. Sistemi zasnovani na statistici i mašinskom učenju čine takozvani „drugi talas“ veštačke inteligencije.

Rasprostranjeno je uverenje da će u budućnosti dosadašnja dva pristupa morati da se integrišu, vodeći do takozvanog „trećeg talasa“ veštačke inteligencije. Očekuje se da će u trećem talasu sistemi veštačke inteligencije samostalno kreirati modele koji će moći da objasne kako stvari funkcionišu.

U ovoj knjizi, simbolički zasnovana veštačka inteligencija obrađuje se u prva dva dela, a statistički zasnovana veštačka inteligencija u trećem delu.

1.4 Uska i opšta veštačka inteligencija

Do sada dominantan pravac razvoja veštačke inteligencije je razvoj sistema specijalizovanih za konkretne zadatke. To je pravac koji se naziva „uska veštačka inteligencija“. Alternativa je „veštačka opšta inteligencija“ (eng. *artificial general intelligence*, *AGI*). Njen cilj je razvoj jedinstvenog sistema koji može da uči, rasuđuje i rešava sve probleme koje može i čovek. Taj cilj neki istraživači pokušavaju da postignu matematičkim modelovanjem ljudskog mozga, iako mnogi smatraju da je to teško dostižno ili nemoguće. Istraživač i futurista Rej Kercvajl (Raymond Kurzweil, trenutno radi u kompaniji Google), u svojoj uticajnoj knjizi „Singularnost je blizu” iz 2005. godine, tvrdi da nije daleko trenutak kada će računari prevazići čoveka u svim intelektualnim aktivnostima i procenjuje da će računari već 2029. godine moći da prođu Tjuringov test (videti poglavlje 1.5), što će, dalje, oko 2045. godine dovesti do „sveopšteg preokreta u ljudskim mogućnostima“.

1.5 Filozofski i etički aspekti veštačke inteligencije

Jedno od ključnih pitanja vezanih za veštačku inteligenciju je da li ona ima za cilj oponašanje prirodne (ljudske ili životinjske) inteligencije. Pitanje koje prethodi ovom pitanju je šta je uopšte inteligencija. Možemo smatrati da inteligencija podrazumeva sledeće sposobnosti: sposobnost pamćenja, skladištenja znanja i mogućnost njegove obrade, sposobnost učenja – usvajanja novih znanja, sposobnost komunikacije sa drugim inteligentnim

bićima ili mašinama, itd. Može se smatrati, dakle, da biće ili mašina imaju atribute inteligentnog, ako imaju navedena svojstva. Alen Tjuring (Alan Turing) formulisao je sledeći znameniti test: ako su u odvojene dve prostorije smeštene jedna ljudska osoba i neki uređaj i ako na identična pitanja pružaju odgovore na osnovu kojih se ne može pogoditi u kojoj sobi je čovek, a u kojoj uređaj, onda možemo smatrati da taj uređaj ima atribute veštačke inteligencije. Tjuring je 1947. godine, govoreći o računarima koji simuliraju čovekovo rasuđivanje, predložio i kreiranje mašina „programiranih da uče i kojima je dopušteno da čine greške" jer, kako kaže, „ako se od mašine očekuje da bude nepogrešiva, onda ona ne može biti inteligentna". Ovakav pristup, koji dozvoljava greške, karakterističan je za mašinsko učenje.

Mnogi problemi veštačke inteligencije mogu se opisati u matematičkim terminima. Pitanje je za koje klase problema postoje opšti načini rešavanja (koje mogu da primene ljudi ili mašine). Rezultati Gedela i Tjuringa iz prve polovine dvadesetog veka pokazali su da postoje matematičke teorije (uključujući i jednostavne teorije kakva je aritmetika) koje su nepotpune i neodlučive. Na primer, postoje tvrđenja o prirodnim brojevima koja su tačna, ali se ne mogu dokazati iz aksioma aritmetike. Štaviše, skup aksioma aritmetike nije moguće proširiti tako da se to promeni. Ovi rezultati govore da za neke probleme ne mogu da postoje mašine koje mogu da reše svaku njihovu instancu, te da preostaje da se njima bave kao i ljudi: da koriste svojstva i zapažanja specifična za konkretne date instance. Drugim rečima, za instance ovakvih problema traganje za rešenjem neće uvek biti isto i neće garantovati uspeh.

Pored filozofskih pitanja o tome gde su granice ljudske i veštačke inteligencije, važna su i etička pitanja koja se odnose na mašine koje mogu da samostalno donose nekakve odluke. Još od ranih dana veštačke inteligencije postoji strah od „mašina koje misle", a sa njihovim razvojem ta pitanja sve su češća a i ti strahovi jačaju. Neke od najčešćih etičkih dilema odnose se na izbor i filtriranje informacija na internetu i društvenim mrežama koje stižu do korisnika, kao i na sisteme za autonomnu vožnju. Na primer, pitanje je kako treba definisati ponašanje vozila u slučaju da mora da ugrozi jednog od dva učesnika u saobraćaju. Još pre toga, pitanje je da li takve odluke prepustiti samom autonomnom sistemu, bez eksplicitne odluke ugrađene unapred. Ukoliko dođe do ugrožavanja bezbednosti učesnika u saobraćaju, povreda, ili štete, postavljaju se i mnoga dodatna, pravna pitanja, na primer – da li za štetu odgovara proizvođač autonomnog automobila ili čovek koji je u njemu sedeo. Mnoga ovakva pitanja će vrlo uskoro biti od praktičnog značaja, a odgovori na njih tek treba da se formulišu.

Deo I

Pretraga

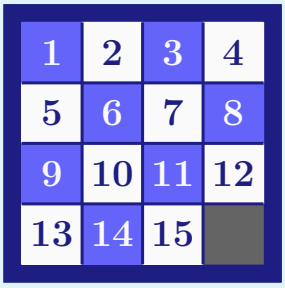
Glava 2

Rešavanje problema korišćenjem pretrage

Veštačka inteligencija bavi se, prevashodno, problemima u kojima se javlja *kombinatorna eksplozija*, tj. ogroman broj mogućnosti koje treba ispitati na neki način. Rešavanje takvih problema obično se svodi na neku vrstu *pretrage* skupa mogućnosti radi pronalaženja niza *koraka* ili *akcija* kojima se ostvaruje cilj. *Rešenje problema pretrage* tada je niz koraka (akcija) koji vodi od polaznog stanja do ciljnog stanja ili nekog od stanja koja zadovoljavaju zadate uslove. U nekim problemima rešenjem se smatra dostignuto stanje, a nije bitan put kojim se do njega stiglo. Neke od važnih realnih primena algoritama pretrage su pronalaženje najkraćih puteva i planiranje putovanja, navigacija robota, rutiranje u računarskim mrežama, igranje strateških igara, automatsko nalaženje redosleda sklapanja delova u industriji, dizajniranje čipova, dizajniranje proteina sa traženim svojstvima, rešavanje logističkih problema i slično.

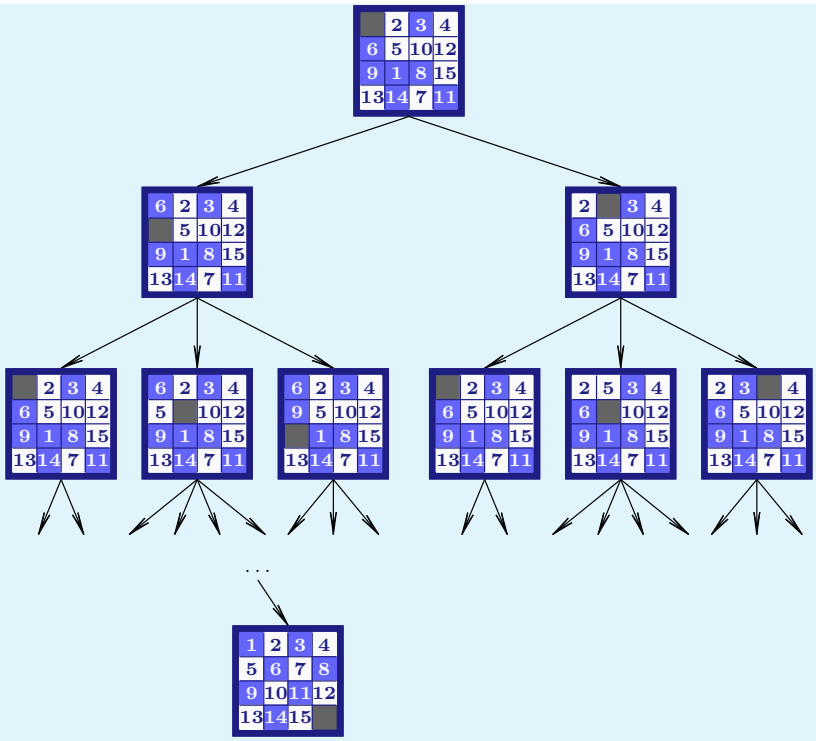
U mnogim problemima nemoguće je razmotriti sve mogućnosti u razumnom vremenu, te je pretragu potrebno *usmeravati* (tj. *navoditi*) kako bi se ranije razmotrile mogućnosti za koje je izglednije da vode rešenju problema. Za uspešno navođenje potrebne su informacije o kvalitetu pojedinačnih stanja, specifične za dati problem. Probleme pretrage za koje su raspoložive takve informacije zovemo *problemi informisane pretrage*, a one druge – *problemi neinformisane pretrage*.

Primer 2.1. Lojdova slagalica *(ili „*slagalica 15*“) sastoji se od 15 kvadrata raspoređenih na tabli veličine* 4×4 *polja. Kvadrati su numerisani brojevima od 1 do 15. Slagalicu je potrebno urediti tako da su polja poređana redom od prvog reda i da je poslednje polje u četvrtom redu prazno. Taj, ciljni raspored polja može se kompaktno zapisati kao* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, _] *i prikazan je na narednoj slici.*



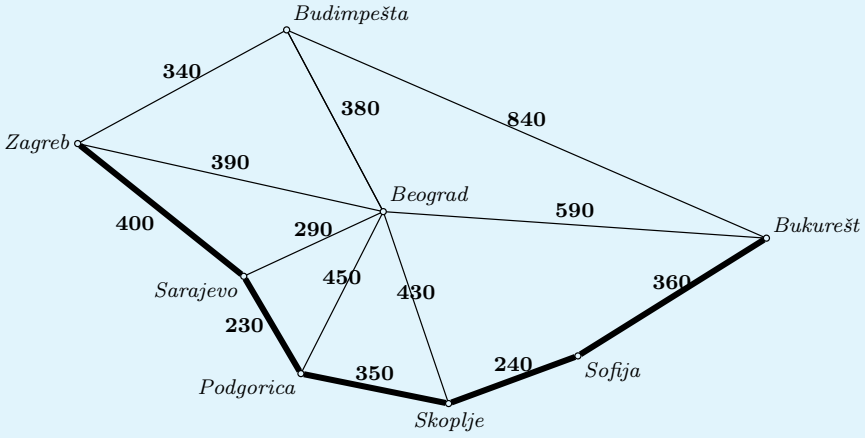
Kada je dat proizvoljan raspored polja na tabli, u svakom koraku može se pomeriti jedno od dva ili jedno od tri ili jedno od četiri polja. Dakle, za svaki raspored broj mogućih akcija je između dva i četiri.

Slagalicu je moguće složiti tako što se razmatraju svi mogući koraci u polaznom stanju, a zatim svi mogući koraci u dobijenim stanjima i tako dalje, sve dok se ne naiđe na traženi, ciljni raspored. Razmatranje svih mogućih koraka za početni raspored [_, 2, 3, 4, 6, 5, 10, 12, 9, 1, 8, 15, 13, 14, 7, 11] *ilustrovano je na narednoj slici.*



Očigledno, ovaj pristup sigurno dovodi do rešenja za bilo koju poziciju koja je dobijena od složene pozicije regularnim potezima (ako se slagalica rasklopi i poljima 14 i 15 se zamene mesta, onda više nije moguće složiti slagalicu). Očigledno je i da je ovaj pristup potpuno nepraktičan i zahteva ispitivanje ogromnog broja mogućnosti. Zaista, za proizvoljnu poziciju koja je dobijena od složene pozicije regularnim potezima, slagalicu je moguće složiti u najviše 80 koraka a to je najmanje gornje ograničenje — postoje pozicije za koje postoji rešenje u 80 koraka, ali ne i u manje od 80 koraka. To znači da je za garantovano pronalaženje rešenja potrebno ispitati više od 2^{80} mogućnosti, što je, naravno, praktično neizvodivo. Zbog toga, praktično sprovodivo rešenje zahteva neku dodatnu ideju i usmeravanje pretrage *kako ne bi bile ispitivane sve mogućnosti. Takvo usmeravanje često nije jednostavno. Na primer, ako se u slaganju slagalice razmatraju samo koraci koji vode do pozicija koje su bliže rešenju, pri čemu se za određivanje „rastojanja pozicije od ciljne pozicije“ može uzeti zbir rastojanja svakog polja od njegove ciljne pozicije, time se ne dolazi uvek do rešenja. Naime, u nekim pozicijama nema koraka koji vodi ka boljoj poziciji (videti primer 4.2).*

Primer 2.2. *U skupu gradova od kojih su neki međusobno povezani putevima, zadatak je od jednog grada stići do nekog drugog zadatog grada. Ovaj problem može se rešavati kao problem pretrage: pretraga može da kreće od početnog grada, da se zatim razmatraju svi gradovi do kojih se može doći neposredno, i tako dalje, sve dok se ne dođe do ciljnog grada. Primer ovakvog problema ilustrovan je na narednoj slici i konkretan zadatak može biti, na primer, nalaženje puta od Zagreba do Bukurešta.*



Od navedenog problema bitno je različit problem u kojem se ne traži bilo koji, nego najkraći put između dva grada. I taj problem ima više varijanti – sve dužine između povezanih gradova mogu biti unapred poznate ili ne, vazdušna rastojanja između svih gradova mogu biti poznata ili ne, itd.

Primer 2.3. *Knjiga „Propositiones ad Acuendos Juvenes“ („Problemi za britkost mladih“) iz devetog veka n.e., verovatno autora Alkoina iz Jorka (Alcuin of York), jedna je od prvih knjiga popularne matematike. Knjiga sadrži pedesetak zanimljivih problema od kojih su mnogi u opticaju i dan-danas. Jedan od najpoznatijih je problem o vuku, kozi i kupusu: „Jedan čovek kupio je vuka, kozu i kupus. Na putu kući, čovek je došao na obalu reke i iznajmio čamac. U čamcu je pored njega mogao da bude još samo vuk, samo koza ili samo kupus. Ako bi koza i kupus bili na istoj obali i bez prisustva čoveka, koza bi pojela kupus. Ako bi vuk i koza bili na istoj obali i bez prisustva čoveka, vuk bi pojeo kozu. Kako da čovek pređe reku i sačuva sve što je kupio?“*

I ovaj problem može se rešavati kao problem pretrage: pretraga može da kreće od početnog rasporeda (svi su na prvoj obali), zatim se sistematično razmatraju svi mogući koraci (svi mogući prelasci reke), sve dok se ne dobije traženi, ciljni raspored (svi su na drugoj obali).

Primeri primena algoritama pretrage. Algoritmi za usmerenu pretragu koriste se u mnogim oblastima. Algoritmi za usmereno traženje puta između dva čvora grafa koriste se u planiranju putovanja, rutiranju, video igrama, pa čak i u pomalo neočekivanim oblastima kao što je parsiranje za verovatnosne konteksno-slobodne gramatike, koje imaju primene u obradi prirodnog jezika.

Igranje strateških igara jedno je od klasičnih polja veštačke inteligencije u kojem su postignuti i neki od najznačajnijih rezultata. Godine 1997. specijalizovani računar Deep Blue kompanije IBM pobedio je tadašnjeg svetskog šampiona u šahu Garija Kasparova koristeći efikasne algoritme pretrage zasnovane na alfa-beta odsecanju. Jedan od trenutno najboljih šahovskih programa je Stockfish, takođe zasnovan na alfa-beta pristupu. Tokom prethodnih nekoliko godina pojavili su se i neki veoma uspešni alternativni pristupi, zasnovani na Monte Karlo pretrazi i na neuronskim mrežama, kao što su Komodo MCTS i AlphaZero. Program za igru go, AlphaGo, zasnovan na neuronskim mrežama 2016. godine pobedio je osamnaestostrukog prvaka sveta Li Sedola.

Genetski algoritmi, koji se mogu smatrati metaheurističkim algoritmom pretrage, imaju primene u širokom spektru optimizacionih problema. Genetski algoritmi koriste se, na primer, za izbor najboljeg plana upita u objektno-relacionim sistemima za upravljanje bazama podataka kao što je PostgreSQL. NASA je početkom ovog veka dizajnirala antene za satelite i svemirske letelice korišćenjem genetskih algoritama koji su se izvršavali na tridesetak umreženih računara. Dizajnirane antene bile su jeftinije, a davale su bolje performanse od drugih. Jedna od konkretnih primena genetskih algoritama je i optimizacija profita berzanskih portfolija.

Faze rešavanja problema korišćenjem pretrage. Tôk rešavanja problema korišćenjem pretrage obično obuhvata sledeće osnovne faze:

- modelovanje problema;
- rešavanje problema opisanog u matematičkim terminima;
- interpretiranje i analiza rešenja.

2.1 Modelovanje problema

Modelovanje problema predstavlja formulisanje problema precizno, u matematičkim terminima, korišćenjem pogodnih matematičkih struktura. Takva formulacija obično ima sledeće elemente:

Skup mogućih stanja: U toku procesa pretrage razmatraju se različita *stanja*. U nekim algoritmima, za odlučivanje u nekom trenutku potrebno je poznavanje samo stanja koja su neposredno dostupna, dok je u nekim algoritmima potrebno poznavanje svih stanja unapred.

Polazno stanje: Rešavanje problema kreće od jednog određenog stanja, koje nazivamo *polaznim stanjem*.

Test cilja: Problem je rešen ako se dođe do *ciljnog stanja*. Potrebno je da postoji raspoloživ efektivan test koji proverava da li se došlo do ciljnog stanja tj. do završetka procesa pretrage.

Skup mogućih akcija: U svakom koraku pretrage može se preduzeti neki korak, neka akcija. Niz akcija preduzetih u odgovarajućim trenucima treba da dovede do rešenja problema. Skup mogućih akcija može biti isti u svakom stanju ili može da se razlikuje od stanja do stanja, što zavisi od problema koji se rešava.

Funkcija prelaska: Ova funkcija preslikava par stanje-akcija u novo stanje, dobijeno izborom neke akcije u nekom stanju.[1] Stanja koja su neposredno dostupna iz nekog stanja zovemo i *susednim stanjima*.

Funkcija cene: Ova funkcija preslikava par stanje-akcija u numeričku vrednost — cenu preduzimanja date akcije u datom stanju.

Kod nekih problema nabrojani elementi se lako i prirodno uočavaju, dok je kod drugih najpre potrebno problem preformulisati na neki pogodan način.

Primer 2.4. *Elementi problema iz primera 2.1 mogu biti definisani na sledeći način:*

- Skup stanja: *skup svih permutacija* $[s_1 s_2 \dots s_{16}]$ *skupa* $\{\_, 1, 2, \dots, 15\}$.
- Polazno stanje: *bilo koje stanje slagalice (za neke od njih ciljni raspored nije moguće dobiti).*
- Test cilja: *provera da li je stanje jednako* [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, _].
- Skup mogućih akcija: *za neka stanja može biti* {*levo*, *desno*, *gore*, *dole*}*, gde se date akcije odnose na pomeranje praznog polja levo, desno, gore i dole. Iako je naizgled prirodnije kao akcije razmatrati pomeranje kvadrata susednih praznom polju na prazno polje, ovakva formulacija je jednostavnija zbog uniformnosti. Za neka stanja (kada je prazno polje na rubu slagalice) moguće su samo neke dve ili neke tri od navedenih akcija.*
- Funkcija prelaska: *preslikava par stanje-akcija u stanja koja nastaju pomeranjem praznog polja na neku od četiri moguće strane.*
- Funkcija cene: *može biti konstantna za svaku akciju (na primer, 1), pošto se sva pomeranja mogu smatrati jednako skupim. Cena rešenja je u tom slučaju jednaka ukupnom broju pomeranja potrebnih za slaganje slagalice.*

Primer 2.5. *Elementi problema stizanja iz jednog grada u drugi (primer 2.2) su:*

- Skup stanja: *skup gradova.*
- Polazno stanje: *grad iz kojeg se kreće.*
- Test cilja: *provera da li je tekući grad jednak ciljnom gradu.*
- Skup mogućih akcija: *kretanje ka neposredno dostupnim gradovima (skup mogućih akcija u ovom problemu se razlikuje od stanja do stanja, jer su za različite gradove različiti i skupovi neposredno dostupnih gradova).*
- Funkcija prelaska: *određena je vezama između gradova.*
- Funkcija cene: *na primer, cena za jednu akciju jednaka je dužini puta ili ceni goriva potrebnog za prevoz između susednih gradova.*

Problemi pretrage često se pogodno opisuju u terminima grafova (i mogu se vizualizovati na odgovarajući način). *Graf prostora stanja* onda opisuje skup mogućih stanja i mogućih akcija i svakom čvoru grafa pridruženo je jedno stanje, a svakoj grani jedna akcija. Takav graf može da bude usmeren ili neusmeren. Neusmeren je ako za svako stanje A iz kojeg se može nekom akcijom doći do stanja B, postoji odgovarajuća akcija iste cene kojom se iz stanja B može doći do stanja A. U primeru slagalice, graf prostora stanja je neusmeren, svakom čvoru grafa pridružen je jedan raspored, a granama odgovaraju pojedinačni koraci. I u primeru gradova, graf prostora stanja je neusmeren, a svakom čvoru pridružen je jedan grad. Za igru šah, međutim, graf bi bio usmeren (jer postoje pozicije A i B takve da se iz A može jednim potezom doći do B, ali ne i obratno). Graf prostora stanja može da bude veoma veliki, ali je obično konačan.

Primer 2.6. *Modelujmo problem opisan u primeru 2.3:*

- *Skup stanja je skup mogućih rasporeda na dve obale reke (taj raspored zapravo je određen rasporedom*

[1] Ukoliko ova funkcija nije poznata, nije poznato u koje će se stanje dospeti posle preduzimanja određene akcije i proces odlučivanja postaje kompleksniji. Funkcija prelaska nije poznata, na primer, u slučaju nepoznate ili promenljive okoline.

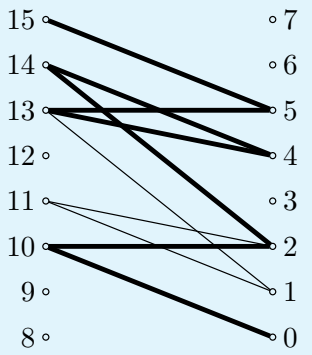
na jednoj obali). Ako je čovek na prvoj obali, označavaćemo to jedinicom, a ako nije – označavaćemo to nulom. Analogno za vuka, kozu i kupus, u tom poretku. Dakle, svako stanje možemo da reprezentujemo nizom od četiri binarne cifre.

- *Polazno stanje je stanje* 1111.
- *Test cilja je provera da li je stanje jednako* 0000.
- *Iz jednog stanja u drugo moguće je doći ako su ispunjeni uslovi:* (i) *vrednost prve cifre se menja (čovek uvek prelazi sa obale na obalu);* (ii) *ako je prva cifra prvog stanja jedinica, onda od preostalih jedinica najviše jedna može da postane nula;* (iii) *ako je prva cifra prvog stanja nula, onda od preostalih nula najviše jedna može da postane jedinica (jer čovek čamcem može da preveze samo vuka, samo kozu ili samo kupus). U stanja* 1100, 1001, 1000, 0111, 0110, 0011 *se ne može doći (da ne bi vuk pojeo kozu ili koza kupus) i iz njih nema mogućih akcija. Ako zapis stanja tretiramo kao binarni broj, možemo ga kraće označiti i dekadnim brojem.*
- *Na osnovu mogućih akcija, funkcija prelaska određuje sledeće veze između stanja:*

15	→	*5*	*7*		–
14	→	*4, 2*	*6*		–
13	→	*5, 4, 1*	*5*	→	*15, 13*
12		–	*4*	→	*14, 13*
11	→	*2, 1*	*3*		–
10	→	*2, 0*	*2*	→	*14, 11, 10*
9		–	*1*	→	*13, 11*
8		–	*0*	→	*10*

- *Funkcija cene nije relevantna ako se u zadatku traži bilo kakvo rešenje, ali to se može pooštriti zahtevom za najkraćim rešenjem, tj. rešenjem u kojem ima najmanje prelaženja reke i tada za svaki prelazak možemo smatrati da ima cenu* 1.

Opisani problem može se elegantno opisati u terminima grafova. Graf prostora stanja *čine čvorovi* $0, 1, \ldots, 15$. *Grane su određene navedenom tabelom (dobijeni graf je bipartitni graf). Ako se može doći iz jednog stanja u drugo, onda se i iz tog drugog može doći u prvo. Dakle, graf je neusmeren. Dobijeni graf prikazan je na narednoj slici:*



Zadatak u ovako formulisanom problemu postaje: u dobijenom grafu pronaći (najkraći) put od čvora 15 *do čvora* 0. *Na slici je podebljanim linijama prikazan traženi put.*

2.2 Rešavanje problema

U fazi rešavanja često je u grafu prostora stanja potrebno pronaći čvor sa nekim svojstvom ili najkraći put do nekog čvora. Pretraživanjem grafa nastaje *stablo pretraživanja* ili *stablo pretrage*. To stablo ne mora nužno da se kreira eksplicitno, kao struktura u memoriji računara, već može da se kreira samo implicitno, procesom obilaska čvorova. U stablu pretrage svakom čvoru pridruženo je jedno stanje, ali jedno stanje može da bude posećeno više puta tokom obilaska, te može da se nalazi u više čvorova stabla pretrage. Zato stablo pretrage može da bude beskonačno i onda kada je prostor stanja konačan. Kada se kaže „čvor“, obično je iz konteksta

jasno da li se misli na čvor grafa prostora stanja ili na čvor u stablu pretrage, a često se isto označavaju čvor i stanje koje mu je pridruženo.

Primer 2.7. *Problem reprezentovan kao u primeru 2.6, može se rešiti sistematičnim pretraživanjem grafa: početni čvor je* 15*, ciljni* 0*. Ako se želi rešenje sa najmanjim brojem prelazaka reke, treba primeniti sistematični obilazak grafa u širinu. Tako se može dobiti sledeće rešenje:* $15-5-13-4-14-2-10-0$.

U mnogim problemima, ciljni čvor u grafu nije moguće pronaći u realnom vremenu sistematičnim pretraživanjem, te je radi bržeg pronalaženja cilja pretragu potrebno usmeravati dodatnim informacijama o kvalitetu pojedinačnih stanja, specifičnim za dati problem (ukoliko su one raspoložive). Kao što je već rečeno, probleme pretrage za koje su raspoložive takve informacije zovemo *problemi informisane pretrage*, a one druge – *problemi neinformisane pretrage*. Granica između ove dve grupe nije oštra, jer može postojati više nivoa raspoloživih informacija za isti problem. Algoritmi pretrage dele se, prema tome kojoj grupi problema su prilagođeni, na *algoritme informisane (tj. usmerene) pretrage* i na *algoritme neinformisane pretrage*.

Primer 2.8. *U problemu iz primera 2.2, ako su raspoložive samo informacije o neposredno povezanim gradovima (pa i cene prelaska u njih), u pitanju je problem neinformisane pretrage i za rešavanje se koriste (neinformisani) algoritmi kao što su algoritmi za sistematičnu pretragu u širinu ili u dubinu ili Dajkstrin algoritam. Ukoliko su, na primer, poznata vazdušna rastojanja između gradova, te informacije mogu se iskoristiti za navođenje pretrage, pa problem pripada grupi problema informisane pretrage, a za njegovo rešavanje može se koristiti algoritam kao što je A*.*

2.3 Interpretiranje i analiza rešenja

Dobijeno rešenje matematički formulisanog problema potrebno je formulisati u terminima početnog problema i potrebno je razumeti njegova svojstva.

Primer 2.9. *U okviru primera 2.7, kao rešenje je dat sledeći put u grafu:* $15-5-13-4-14-2-10-0$*. Ako se oznake čvorova grafa prikažu kao binarni brojevi koji označavaju stanja, dobijeni put je:* $1111-0101-1101-0100-1110-0010-1010-0000$*. Lako se može rekonstruisati prvi korak algoritma –* $1111-0101$*: na prvoj obali su bili čovek, vuk, koza i kupus, a onda samo vuk i kupus. Dakle, u prvom koraku čovek na drugu obalu odvozi kozu. Slično se može rekonstruisati čitavo rešenje u terminima polaznog problema: čovek prevozi kozu–čovek se vraća sâm–čovek prevozi kupus–čovek prevozi kozu–čovek prevozi vuka–čovek se vraća sâm–čovek prevozi kozu.*

Algoritmi koji se koriste u rešavanju problema pretrage često (mada ne uvek) su egzaktni i rešenja koja vraćaju sigurno ispunjavaju neke zadate uslove. U takvim situacijama nema potrebe sprovoditi evaluiranje dobijenih rešenja. Neki algoritmi pretrage ne vrše sistematičnu pretragu i tada je potrebno znati da li garantovano vraćaju isti (obično brže pronađen) rezultat kao da je primenjena sistematična pretraga. Pored toga, potrebno je znati da li korišćeni algoritam pretrage uvek vraća rešenje koje je najbolje moguće (u nekom smislu), a ako to nije ispunjeno, poželjno je znati koliko dobijeno rešenje može da odstupa od optimalnog rešenja (na primer, ako je pronađen put do neke lokacije u gradu, pitanje je koliko on odstupa od najboljeg puta). Dodatno, iako su algoritmi koji se koriste za pretragu jasno definisani, neki njihovi parametri mogu biti predmet podešavanja i, na primer, mogu biti određivani metodama mašinskog učenja. Tada bi taj deo rešavanja problema prolazio kroz faze uobičajene za mašinsko učenje.

Najvažnija opšta svojstva koje algoritmi pretrage mogu da imaju su sledeća:

Potpunost je svojstvo koje garantuje da će algoritam naći neko rešenje problema ako rešenja uopšte postoje. Ovo svojstvo je očito poželjno, ali se u nekim slučajevima ne zahteva. Naime, u slučaju vrlo teških problema često je moguće formulisati heuristike koje ne garantuju pronalaženje rešenja, ali u visokom procentu slučajeva nalaze dobra rešenja mnogo brže nego potpuni algoritmi.

Optimalnost je svojstvo koje garantuje nalaženje rešenja sa najmanjom cenom (pri čemu se cenom rešenja može smatrati zbir cena akcija koje se preduzimaju).[2] *Optimalno rešenje* je rešenje sa najmanjom cenom i ono ne mora biti jedinstveno. Moguće je da algoritam koji nema svojstvo optimalnosti često pronalazi rešenja bliska optimalnim, ali u značajno kraćem vremenu.

Vremenska i prostorna složenost govore, kao i za druge vrste algoritama, o tome koliko je vremena i memorijskog prostora potrebno za sprovođenje procesa pretrage. Obično se razmatra složenost najgoreg i prosečnog slučaja.

[2] Svojstvo optimalnost algoritma može da ima i drugačije značenje: da je (vremenska ili prostorna) složenost algoritma najbolja među svim algoritmima koji rešavaju taj problem.

Glava 3

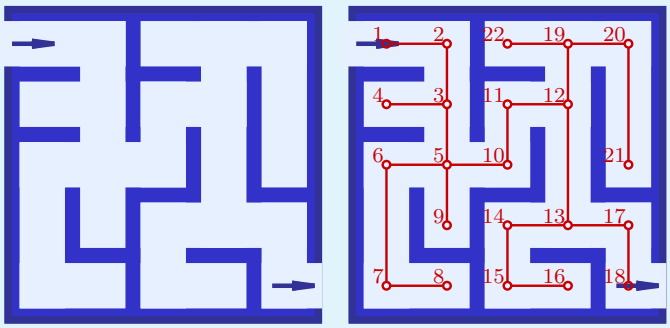
Neinformisana pretraga

U svim problemima pretrage, podrazumeva se da je moguće opaziti tekuće stanje, preduzimati akcije i prepoznati ciljno stanje. Specifično za neinformisanu pretragu (eng. *uninformed search*) je to što nema dodatnih informacija koje mogu pomoći u navođenju koje ubrzava pronalaženje ciljnog stanja. U primeru pronalaženja najkraćeg puta od nekog grada do ciljnog grada, scenario neinformisane pretrage odgovara, na primer, situaciji u kojoj se u svakom gradu zna koji je to grad, moguće je izabrati jedan od puteva ka drugim gradovima (do kojih su poznata rastojanja), moguće je pamtiti posećene gradove i prepoznati odredišni, ali nema informacija o rastojanju od pojedinačnih gradova do odredišta (koje bi mogle da usmeravaju pretragu). Tipičan primer problema neinformisane pretrage je i problem lavirinta koji je opisan u nastavku.

Primer 3.1. Lavirint *u ravni sastoji se od skupa povezanih hodnika kojima je moguće kretati se. Svaki hodnik ima jedno ili više polja i dva kraja. Jedno polje je* ulaz, *a jedno* izlaz *iz lavirinta. Ulaz, izlaz, krajeve hodnika, kao i polja koja su zajednička za dva hodnika zovemo* čvorovima lavirinta. *Cilj je pronaći put od ulaza do izlaza preko čvorova lavirinta. Elementi ovog problema su sledeći:*

- *Skup stanja: skup čvorova lavirinta.*
- *Polazno stanje: ulaz u lavirint.*
- *Test cilja: provera da li je tekući čvor izlaz iz lavirinta.*
- *Skup mogućih akcija: izbor puta (tj. sledećeg čvora lavirinta) u svakom koraku.*
- *Funkcija prelaska: određena je vezama između čvorova lavirinta.*
- *Funkcija cene: cena za prelazak iz jednog čvora u susedni je, na primer,* 1.

Naredna slika ilustruje lavirint (levo) i graf prostora stanja koji mu odgovara (desno).



Elementi problema pretrage (skup stanja i funkcija prelaska), pa i problema neinformisane pretrage, najčešće se prirodno izražavaju u terminima grafova. U slučaju lavirinta, radi se o grafu čiji su čvorovi čvorovi lavirinta, a grane su putevi između tih čvorova lavirinta (primer 3.1). I algoritmi pretrage najčešće su formulisani u vidu algoritama obilaska grafova.

3.1 Obilazak grafa u dubinu i širinu

Obilazak grafa u dubinu (eng. *depth-first search — DFS*) i u širinu (eng. *breadth-first search — BFS*) su metode neinformisane pretrage koje ispituju čvorove u grafu tražeći rešenje, što je obično neki specifičan čvor. Ove metode sistematski pretražuju ceo graf bez ikakvog navođenja i staju sa pretragom kada pronađu traženi čvor ili kada obiđu čitav graf ne pronašavši traženi čvor.

3.1.1 Pretraga u dubinu

Pretraga u dubinu je pretraga koja kreće od polaznog čvora i zatim napreduje ka sve daljim čvorovima. Ukoliko u nekom čvoru nema suseda koji još uvek nisu ispitani, pretraga se vraća unazad do čvora čiji svi susedi nisu ispitani i nastavlja dalje. U nerekurzivnoj verziji, čvorovi na tekućem putu od polaznog čvora obično se čuvaju na steku, tj. u LIFO listi. Da ne bi došlo do beskonačne petlje, potrebno je čuvati informaciju o čvorovima koji su već posećeni. Ovaj postupak opisan je algoritmom DFS na slici 3.1, a slika 3.2 ilustruje obilazak jednog grafa. Prikazani algoritam, ukoliko pronađe ciljni čvor, u tom trenutku na steku *put* sadrži redom čvorove koji čine traženi put.

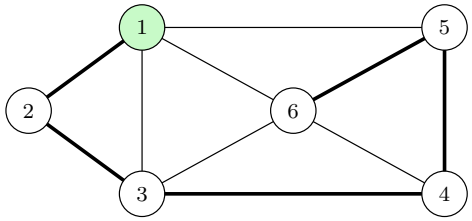
Algoritam: DFS (pretraga u dubinu)

Ulaz: Graf G, polazni čvor i ciljni čvor

Izlaz: Put od polaznog do ciljnog čvora u grafu G (ako postoji takav put)

```
1:  na stek put i u skup posećenih čvorova stavi samo polazni čvor;
2:  dok god stek put nije prazan radi
3:      uzmi čvor n sa vrha steka put;
4:      ako je n ciljni čvor onda
5:          izvesti o uspehu i vrati put konstruisan na osnovu sadržaja steka put;
6:      ako n nema susednih čvorova koji nisu posećeni onda
7:          izbaci n sa steka put;
8:      inače
9:          izaberi prvi neposećeni čvor m susedan čvoru n i dodaj ga na vrh steka put i u skup posećenih
            čvorova;
10: izvesti da traženi put ne postoji.
```

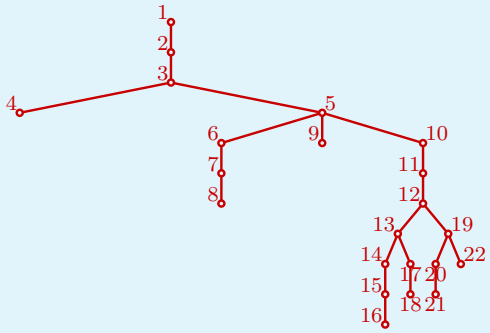
Slika 3.1: DFS — algoritam pretrage u dubinu.



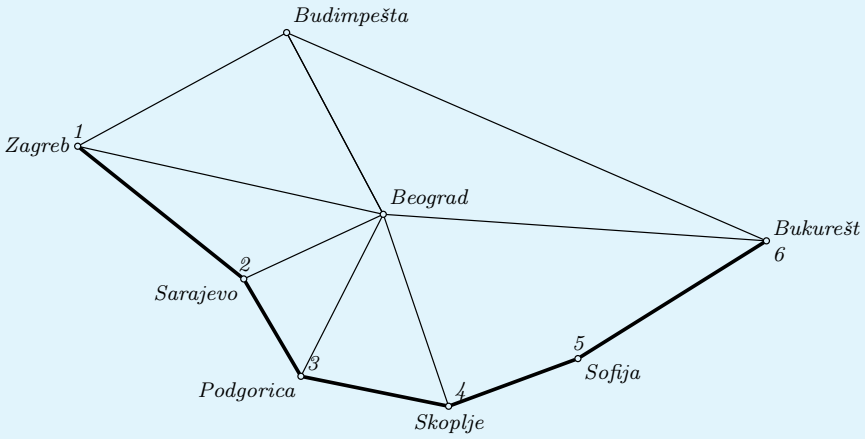
Slika 3.2: Primer obilaska grafa primenom algoritma DFS (oznake čvorova ukazuju na poredak obilaska čvorova).

Primer 3.2. *Jedan način pronalaženja izlaza iz lavirinta (videti primer 3.1) je držati se desne strane hodnika i pratiti zidove dok se ne naiđe na izlaz. Ovaj način traženja izlaza odgovara obilasku odgovarajućeg grafa u dubinu (sve dok se ne naiđe na traženi čvor grafa). Dakle, algoritam DFS može se upotrebiti za pronalaženje izlaza iz lavirinta koji je opisan grafom (pri čemu algoritam vraća ceo put od ulaza do izlaza).*

Naredna slika ilustruje stablo koje odgovara obilasku grafa lavirinta u dubinu (pri čemu se uvek prvo bira desni put). Ukoliko se traži izlaz iz lavirinta, oblilazak prikazanog stabla staje nakon dolaska u čvor 18.

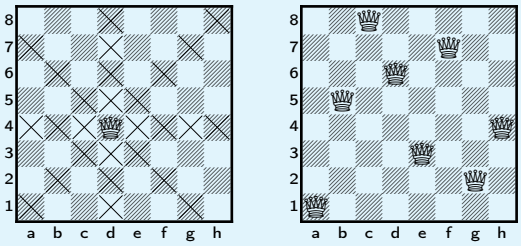


Primer 3.3. *Ukoliko se, pošavši od Zagreba traži put do Bukurešta primenom algoritma DFS i ukoliko se prilikom izbora sledećeg grada prednost daje južnijem, bio bi pronađen put Zagreb-Sarajevo-Podgorica-Skoplje-Sofija-Bukurešt. Ovaj put je po dužini vrlo loš izbor, što je i bilo očekivano pošto algoritam ne uzima u obzir dužine puteva između gradova. Kako se može naći najkraći put, biće prikazano kasnije.*



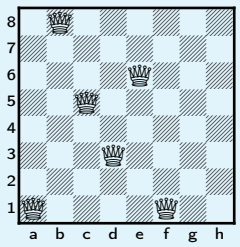
Bektreking (eng. *backtracking*) je modifikovana varijanta pretrage u dubinu. Modifikacija se sastoji u tome da se napredovanje u dubinu prekida čim se (na neki način) ustanovi da se ciljni čvor ne nalazi među potomcima tekućeg čvora i tada nastupa vraćanje na prethodni čvor. Ilustrativan primer za bektreking je rešavanje problema osam dama.

Primer 3.4. Problem osam dama *formulisan je 1848. godine i od tada je bio predmet mnogih matematičkih i informatičkih istraživanja. Cilj je rasporediti osam dama na šahovskoj tabli tako da se nikoje dve dame ne napadaju. Skup polja koja jedna dama napada definisan je u skladu sa opštim pravilima šaha i ilustrovan je na narednoj slici (levo). Na narednoj slici (desno) prikazano je i jedno rešenje ovog problema.*

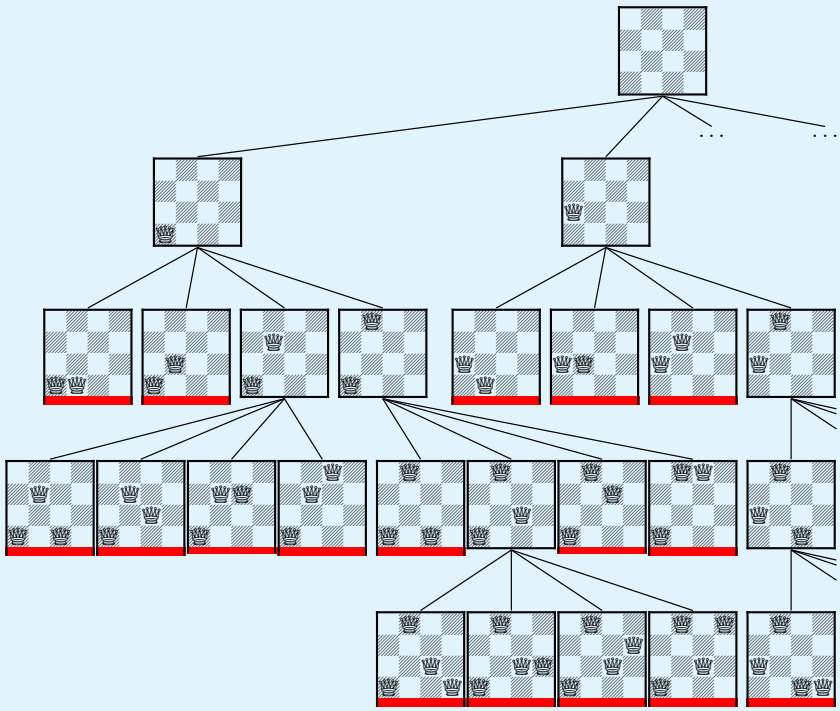


Prostor stanja koji se analizira čine svi rasporedi od 0 *do* 8 *dama (uključujući i one u kojima se neke dame međusobno napadaju). Postoji grana od jednog stanja (rasporeda) ka drugom ukoliko se drugi može dobiti od prvog dodavanjem jedne dame na slobodno polje na tabli. Neki raspored moguće je dobiti različitim redosledima dodavanja dama polazeći od prazne table, ali se dodavanjem dama ne može dobiti tabla sa manjim brojem dama. Dakle, radi se o usmerenom acikličnom grafu. Polazno stanje je prazna tabla, a ciljno stanje je bilo koje stanje koje zadovoljava uslove problema (za osam dama postoji* 92 *rešenja).*

U bilo kom rešenju, očigledno, u jednoj koloni mora biti tačno jedna dama, pa se pretraga može vršiti na sledeći način: dame se dodaju redom po kolonama, po jedna u svaku kolonu. U stablu pretrage, na svakom putu od prazne table do rasporeda sa osam dama od kojih se barem neke napadaju, postoji prvi raspored za koji važi to isto. Kako se duž puteva kroz stablo pretrage dame samo dodaju, nakon tog rasporeda i svi drugi rasporedi na putu imaju dame koje se napadaju. Dakle, postupak pretrage ne treba nastavljati kada se naiđe na prvi raspored sa dve dame koje se napadaju. Na narednoj slici prikazan je jedan raspored koji ne treba ispitivati dalje.



Na narednoj slici prikazan je deo stabla pretrage koja koristi opisani bektreking postupak za problem četiri dame (za problem osam dama stablo pretrage preveliko je za ilustraciju). Crvenom bojom podvučena su stanja za koja nema potrebe nastaviti ispitivanje (te se dalja pretraga u tim čvorovima prekida), kao i završna stanja koja nisu ciljna.



Problem n dama *je uopštenje problema osam dama: treba rasporediti n dama na tabli dimenzija $n \times n$ tako da se nikoje dve ne napadaju. Ovaj problem može se rešavati kao što je opisano za slučaj $n = 8$.*

Na prethodnom primeru mogu se uočiti neke tipične osobine bektrekinga. Bektreking se zasniva na proširivanju tekućeg parcijalnog rešenja. Polazno parcijalno rešenje je prazno rešenje. U prethodnom primeru, to je prazna šahovska tabla, a proširivanje parcijalnog rešenja se vrši dodavanjem dame na tablu. Proširivanje parcijalnog rešenja u nekim slučajevima nije isplativo ili nije moguće i tada se pretraga vraća unazad, odakle dolazi i ime tehnike. U problemu dama, nije isplativo nastaviti pretragu ukoliko je dostignut raspored u kojem se dve dame napadaju. Prilikom izbora naredne grane u pretrazi, prati se neki poredak izbora, u slučaju problema dama, na primer – sledeće prazno polje u skladu sa nekom numeracijom polja.

3.1.2 Pretraga u širinu

Pretraga u širinu razmatra čvorove koji su susedni nekom čvoru, onda redom sve njihove susede, pa redom sve njihove susede, i tako dalje. U traganju za čvorom koji zadovoljava neki uslov, biće pronađen onaj na najmanjem rastojanju (pri čemu se pod rastojanjem misli na broj grana) od polaznog čvora. Čvorovi koji se razmatraju obično se čuvaju u redu, tj. u FIFO listi. Da ne bi došlo do beskonačne petlje, potrebno je čuvati informaciju o čvorovima koji su već posećeni. Ta informacija ne mora da se čuva eksplicitno, već kroz informaciju o čvoru roditelju. Opisani algoritam BFS prikazan je na slici 3.3. Slika 3.4 ilustruje obilazak grafa primenom algoritma BFS.

Primer 3.5. *U slučaju traženja puta od Zagreba do Bukurešta primenom algoritma BFS, na početku je tekući grad Zagreb. Iz Zagreba, pronalaze se Sarajevo, Beograd i Budimpešta (pretpostavimo da su tim redom, u smeru suprotnom od smera kazaljke na satu, poređani kao susedi čvora Zagreb). Oni čine novi red S i za njih se pamti da je prethodni grad Zagreb, koji se uklanja iz reda. Iz Sarajeva se pronalazi put do Podgorice koja se dodaje na kraj reda S, a Sarajevo se iz njega uklanja. Iz Beograda se pronalazi put do Skoplja i Bukurešta, koji se dodaju na kraj reda S, a Beograd se iz njega uklanja. Iz Budimpešte se ne pronalazi put ni do jednog grada koji već nije obrađen. Budimpešta se uklanja iz reda. Iz Podgorice se ne pronalazi put ni do jednog grada koji već nije obrađen. Podgorica se uklanja iz reda. Iz Skoplja se pronalazi put do Sofije koja se dodaje na kraj reda S, a Skoplje se iz njega uklanja. Kada Bukurešt postane grad koji se analizira, konstatuje se da je to ciljni grad, konstruiše se traženi put (Zagreb-Beograd-Bukurešt) i algoritam se zaustavlja.*

Na narednoj slici ilustrovano je opisano traženje puta od Zagreba do Bukurešta. Gore je prikazan graf prostora stanja, a dole je prikazano stablo pretrage.

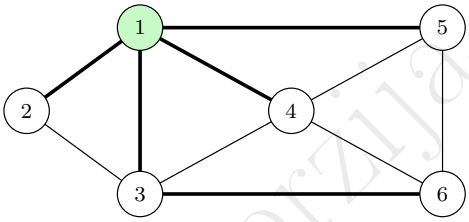
Algoritam: BFS (pretraga u širinu)

Ulaz: Graf G, polazni čvor i ciljni čvor

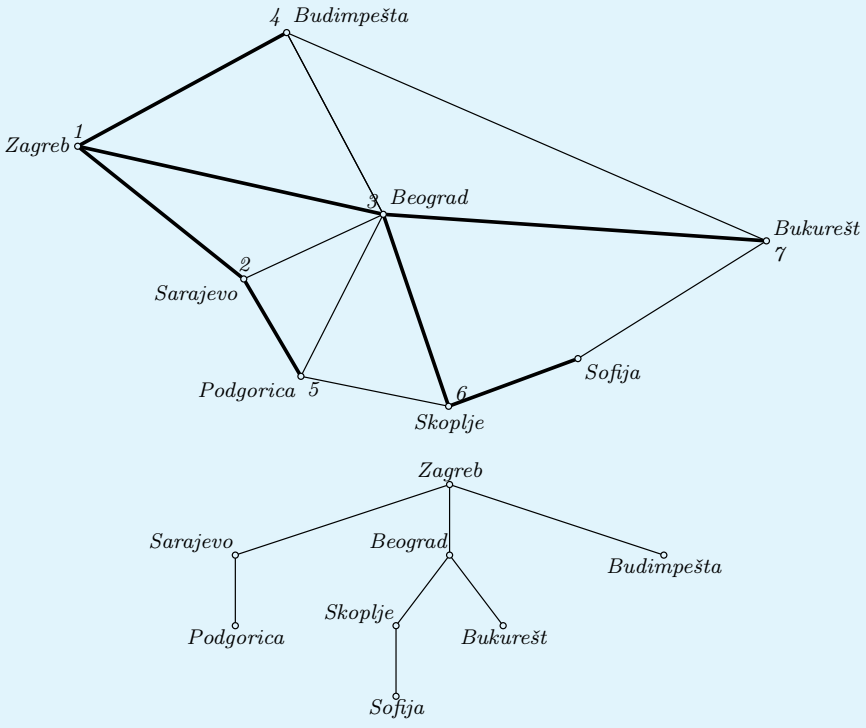
Izlaz: Put od polaznog do ciljnog čvora u grafu G (ako postoji takav put)

1: za svaki čvor zapamti da mu je roditelj nedefinisan;
2: stavi (samo) polazni čvor u red S;
3: **dok god** red S nije prazan **radi**
4: uzmi čvor n sa početka reda S i obriši ga iz reda;
5: **ako** je n ciljni čvor **onda**
6: izvesti o uspehu i vrati put od polaznog do ciljnog čvora (idući unazad od ciljnog čvora, prateći roditeljske čvorove);
7: **za** svaki od susednih čvorova m čvora n za koji nije definisan roditelj i koji nije polazni čvor **radi**
8: zapamti n kao roditelja i dodaj m na kraj reda S;
9: izvesti da traženi put ne postoji.

Slika 3.3: BFS — algoritam pretrage u širinu.



Slika 3.4: Primer obilaska grafa primenom algoritma BFS (oznake čvorova ukazuju na poredak obilaska čvorova).



DFS pretraga pogodnija je od BFS pretrage za usmeravanje koje bira čvorove koji više obećavaju. Vremenska složenost oba algoritma je reda zbira broja čvorova i broja grana grafa koji se pretražuje ($O(|V| + |E|)$), a prostorna je reda broja čvorova ($O(|V|)$).

3.2 Dajkstrin algoritam

Dajkstrin algoritam je algoritam za pretragu grafa koji nalazi najkraće puteve u grafu sa nenegativnim cenama koje su pridružene granama. Razvio ga je holandski informatičar Edzger Dajkstra (Edsger Dijkstra) 1959. godine. Algoritam se može koristiti za određivanje najkraćeg puta od datog čvora do datog ciljnog čvora, ali i za određivanje najkraćih puteva od datog čvora do svih drugih čvorova grafa. Ideja Dajkstrinog algoritma može se ilustrovati na sledeći način. Pretpostavimo da je svakom čvoru grafa pridružena jedna kugla a da su dve takve kugle povezane nitima ako i samo ako postoji grana između odgovarajućih čvorova u grafu. Neka su dužine tih niti proporcionalne cenama odgovarajućih grana. Neka je čitava konfiguracija skupljena na jednoj tački na tlu. Uzmimo kuglu koja odgovara polaznom čvoru i podižimo je od tla, zajedno sa čitavom konfiguracijom, tako da se nikoje dve niti ne upletu i tako da se sve kugle koje su podignute kreću duž iste vertikalne linije. Postepeno se kugle, jedna po jedna, odvajaju od tla, a najmanje rastojanje izmedu polazne kugle i bilo koje podignute kugle jednako je neposrednom rastojanju (vazdušnom linijom) izmedu njih. Opšta ideja algoritma je slična: postoje čvorovi koji su već „podignuti sa tla“ i oni koji su još uvek „na tlu“. Za one koji su podignuti sa tla već znamo najkraće puteve od polaznog čvora. U svakom koraku još jedan čvor „podižemo sa tla“ i izračunavamo njegovo najmanje rastojanje od polaznog čvora (razmatrajući samo one čvorove koji su mu susedni i koji su već iznad tla). Ukoliko na kraju ovog postupka „na tlu“ ostanu još neki čvorovi, to znači da do njih ne postoji put od polaznog čvora.

Algoritam: Dajkstrin algoritam

Ulaz: Graf G, polazni čvor i ciljni čvor

Izlaz: Najkraći put od polaznog do ciljnog čvora u grafu G (ako postoji takav put)

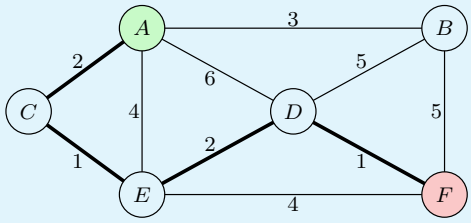
1: stavi sve čvorove grafa u skup Q;
2: za svaki čvor iz Q upamti trenutno najmanje rastojanje od polaznog čvora: za polazni čvor to rastojanje je 0, a za sve ostale čvorove je ∞;
3: **dok god** skup Q nije prazan **radi**
4: izaberi iz Q čvor n sa najmanjim ustanovljenim rastojanjem od polaznog čvora i obriši ga iz Q;
5: **ako** je n ciljni čvor **onda**
6: konstruiši put od polaznog do ciljnog čvora (idući unazad od ciljnog čvora, prateći roditeljske čvorove) i izvesti o uspehu;
7: **za** svaki čvor m iz Q koji je direktno dostupan iz n **radi**
8: **ako** je rastojanje od polaznog čvora do m preko čvora n manje od tekućeg rastojanja od polaznog čvora do m **onda**
9: promeni informaciju o roditelju čvora m na čvor n i upamti novo rastojanje;
10: izvesti da traženi put ne postoji.

Slika 3.5: Dajkstrin algoritam.

Dajkstrin algoritam prikazan je na slici 3.5. U svakoj iteraciji, bira se čvor n iz skupa čvorova Q (svi ti čvorovi još uvek su „na tlu“) takav da je vrednost tekućeg najmanjeg rastojanja od polaznog čvora do njega najmanja. Taj čvor se tada briše iz skupa Q. Ukoliko je to ciljni čvor, onda se konstruiše traženi put od polaznog čvora (koristeći informaciju o roditeljskim čvorovima). Inače, za svaki čvor m iz Q koji je susedan čvoru n proverava se da li se (preko n) može popraviti tekuće najmanje rastojanje od polaznog čvora i, ako može, čvor n se postavlja za roditelja čvora m. Invarijanta petlje je da se za čvorove koji nisu u Q zna najkraće rastojanje od polaznog čvora. Dodatno, ako takvih čvorova ima k, onda upravo ti čvorovi čine k najbližih čvorova polaznom čvoru.

U najjednostavnijoj implementaciji Dajkstrinog algoritma, skup Q se implementira kao obična povezana lista ili niz. Složenost algoritma sa takvom implementacijom skupa Q je $O(|V|^2 + |E|) = O(|V|^2)$, gde je V skup čvorova, a E skup grana grafa. Za retke grafove (koji imaju mnogo manje grana od $|V|^2$), Dajkstrin algoritam može se implementirati efikasnije. Na primer, varijanta koja koristi min-hip za određivanje tekućeg najbližeg čvora ima složenost $O((|E| + |V|) \log |V|)$ (min-hip je struktura u formi balansiranog binarnog stabla za koju važi da je element u svakom čvoru manji od ili jednak elementima u svim njegovim potomcima).

Primer 3.6. *Dajkstrin algoritam potrebno je primeniti na naredni graf, za pronalaženje najkraćeg puta od čvora A do čvora F (taj traženi put prikazan je podebljanom linijom):*

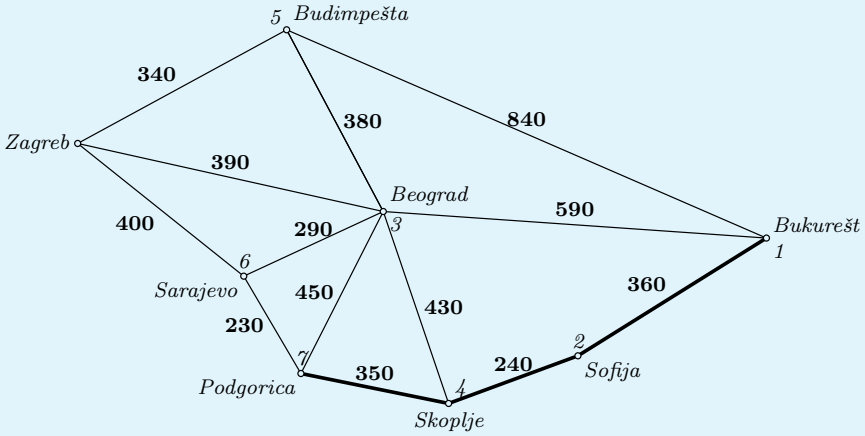


Naredna tabela prikazuje efekat primene algoritma:

korak	*B*	*C*	*D*	*E*	*F*	*čvor n (roditelj)*
1	∞	∞	∞	∞	∞	*A*
2	3	**2**	6	4	∞	*C (A)*
3	**3**		6	3	∞	*B (A)*
4			6	**3**	8	*E (C)*
5			**5**		7	*D (E)*
6					**6**	*F (D)*

U ovom primeru, čvor A je polazni, a čvor F ciljni čvor. Polje tabele za neki čvor prikazuje vrednost najkraćeg do tada nađenog rastojanja od polaznog do tog čvora.

Primer 3.7. *Dajkstrin algoritam potrebno je primeniti za nalaženje puta od Bukurešta do Podgorice (na narednoj slici podebljanom linijom prikazan je najkraći put – Bukurešt-Sofija-Skoplje-Podgorica):*



Naredna tabela ilustruje izvršavanje algoritma:

korak	*Bg*	*So*	*Bud*	*Sk*	*Pg*	*Sa*	*Zg*	*čvor n (roditelj)*
1	∞	∞	∞	∞	∞	∞	∞	*Bukurešt*
2	590	**360**	840	∞	∞	∞	∞	*Sofija (Bukurešt)*
3	**590**		840	600	∞	∞	∞	*Beograd (Bukurešt)*
4			840	**600**	1040	880	980	*Skoplje (Sofija)*
5			**840**		950	880	980	*Budimpešta (Bukurešt)*
6					950	**880**	980	*Sarajevo (Beograd)*
7					**950**		980	*Podgorica (Skoplje)*

Glava 4

Informisana pretraga

Informisana (ili heuristička) pretraga (eng. *informed search*) koristi ne samo informaciju o mogućim akcijama (koracima) u svakom stanju, već i dodatno znanje o konkretnom problemu koje može da usmerava pretragu ka stanjima koja više obećavaju, za koje postoji nekakvo očekivanje da brže vode ciljnom stanju, tj. rešenju problema. To dodatno znanje treba da bude nekakva ocena, mera kvaliteta stanja. Ta mera kvaliteta često nije egzaktna, nego predstavlja nekakvu procenu, heurističku meru.[1]

Funkciju koja ocenjuje kvalitet stanja zovemo *funkcija evaluacije*. Ukoliko je funkcija evaluacije označena sa f, onda $f(n)$ označava ocenu stanja n. Podrazumevaće se da su cene akcija (ili cene grana grafa) nenegativne. Već je rečeno da se problemi pretrage često mogu pogodno zadati u terminima grafova koji opisuju prostor stanja, pa će se umesto „stanja“ i „ocena stanja“ govoriti i „čvor“ i „ocena čvora“. U nastavku će često biti reči o *rastojanjima* i *dužini puta* između čvorova i slično, ali se većina razmatranja može odnositi i na cene puteva koje su zasnovane, na primer, na potrebnom vremenu, ukupnoj putarini, i slično.

Prilikom rešavanja problema pretragom, generiše se stablo pretrage (obično samo implicitno) čijim su čvorovima pridružena stanja. Pošto kroz jedno isto stanje može da se prođe više puta tokom pretrage, može da bude više čvorova stabla pretrage sa istim tim stanjem. Pošto ocena stanja može da zavisi od trenutnog konteksta procesa pretrage, obično je preciznije reći „ocena čvora (stabla pretrage)“ nego „ocena stanja“.

4.1 Pohlepna pretraga

Pohlepnim algoritmom naziva se algoritam koji teži neposrednom povećanju vrednosti neke ciljne funkcije. Ovakav algoritam ne procenjuje dugoročni kvalitet izabranih akcija, tj. koliko one doprinose ostvarenju konačnog cilja, već bira akciju koja se na osnovu znanja dostupnog u trenutku izbora procenjuje kao najbolja među raspoloživim akcijama. Postoji više varijanti pohlepnih algoritama, a jedna varijanta pohlepne pretrage u grafu prikazana je na slici 4.1, gde je sa f označena funkcija evaluacije, tj. funkcija koja procenjuje kvalitet čvorova.

Primer 4.1. *U primeru pronalaženja najkraćih puteva između gradova (primer 2.2), ukoliko su, na osnovu mape, poznata sva vazdušna rastojanja između gradova, ona mogu da se koriste za (prilično naivnu) funkciju evaluacije. Pohlepna pretraga tada može da za sledeći grad uvek bira onaj koji je vazdušnim putem najbliži ciljnom gradu. Razmotrimo gradove prikazane na narednoj slici (podebljanim ciframa ispisana su kopnena, običnim ciframa vazdušna rastojanja između dva grada).*

[1] Heuristike su tehnike za usmeravanje i sužavanje pretrage u problemima u kojima se javlja kombinatorna eksplozija. Reč „heuristika“ potiče od grčke reči „heurisko“ koja znači „tražiti“ ili „otkrivati“. Srodna grčka reč „heureka“ ili „eureka“ znači „našao sam“ ili „otkrio sam“ i obično se vezuje za Arhimeda i njegov uzvik kada je došao do jednog znamenitog otkrića. Aristotel je koristio termin „heuristika“ za otkrivanje novog znanja (ili demonstriranje postojećeg) kroz komunikaciju i interakciju između izlagača i slušalaca. Perl (1984) pod heuristikama smatra „kriterijume, metode ili principe za izbor između nekoliko mogućih akcija one akcije koja obećava da će biti najkorisnija za postizanje nekog cilja”.

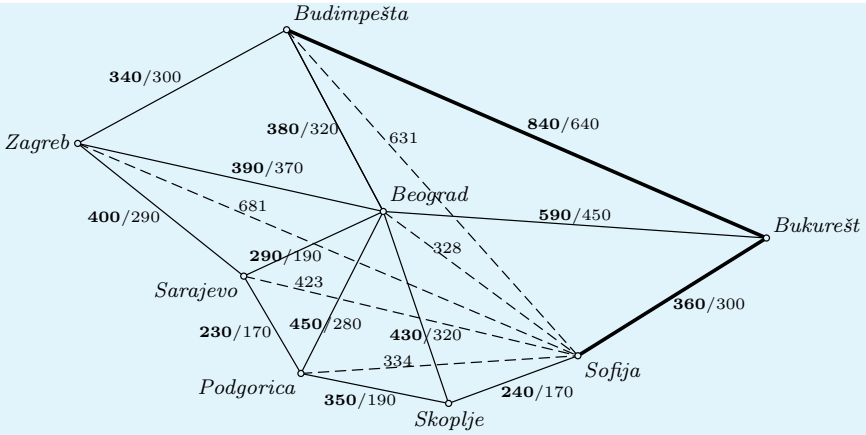
Algoritam: Pohlepna pretraga u grafu

Ulaz: Graf G, polazni čvor i ciljni čvor

Izlaz: Niz koraka od polaznog do ciljnog čvora ili neuspeh (i niz koraka do najboljeg pronađenog čvora)

1: tekući čvor postavi na polazni čvor;
2: **ponavljaj**
3: **ako** je tekući čvor ciljni čvor **onda**
4: izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora (idući unazad — od ciljnog čvora);
5: **ako** nema nijednog čvora m koji je direktno dostupan iz tekućeg čvora i ima bolju ocenu $f(m)$ od ocene tekućeg čvora **onda**
6: izvesti o neuspehu i vrati tekući čvor kao najbolji pronađeni (i konstruiši put od polaznog do tog čvora);
7: **inače**
8: od čvorova koji su direktno dostupni iz tekućeg čvora izaberi čvor m koji ima najbolju ocenu $f(m)$, zapamti tekući čvor kao njegovog roditelja i postavi m da bude novi tekući čvor.

Slika 4.1: Algoritam pohlepne pretrage.



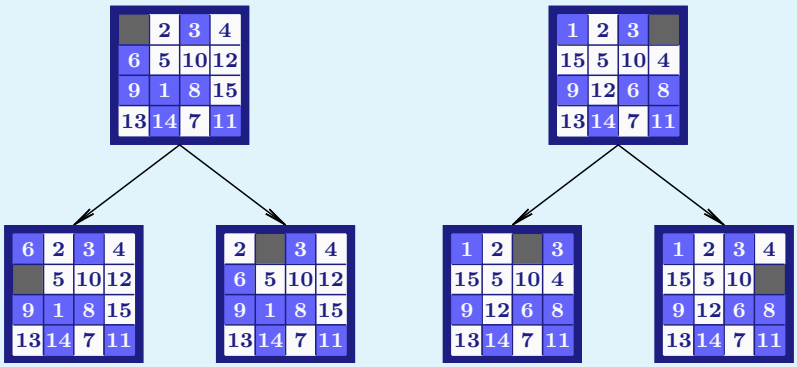
Neka se traži put od Budimpešte do Sofije. Na slici su isprekidanim linijama sa Sofijom povezani gradovi do kojih ne postoji direktna kopnena veza. Ocena za Budimpeštu i njoj direktno dostupne gradove – Zagreb, Beograd, Bukurešt su 631, 681, 328 i 300. Bukurešt ima najbolju ocenu i ona je bolja od ocene za Budimpeštu, te se bira Bukurešt (i Budimpešta se navodi kao roditelj). Iz Bukurešta su direktno dostupni Budimpešta, Beograd i Sofija i njihove ocene su 631, 328 i 0, te se bira Sofija (i Bukurešt se navodi kao roditelj). Onda se konstatuje da je Sofija ciljni čvor i vraća se put Budimpešta-Bukurešt-Sofija. Primetimo da ovo nije najkraći mogući put – njegova dužina je 1200, dok je dužina puta Budimpešta-Beograd-Skoplje-Sofija jednaka 1050,

Pretpostavimo sada da je Skoplje polazni čvor, da je ciljni čvor (ponovo) Sofija, ali i da ne postoji direktan put između ova dva grada. Ocena Skoplja je 170 i ona je veća nego ocene za direktno dostupne gradove — Beograd (328) i Podgoricu (334), te pohlepna pretraga ne može da napravi nijedan korak.

Navedeni primer ilustruje opšti problem pohlepne pretrage: pohlepni algoritmi ne garantuju optimalnost, pa ni potpunost procesa rešavanja. Naime, moguće je da pohlepni algoritam vrati rešenje koje nije najbolje ili da ne nađe put do ciljnog čvora i ako on postoji. Dobra strana algoritama zasnovanih na pohlepnoj pretrazi je to što su veoma jednostavni, često veoma efikasni i mogu da daju dovoljno dobre rezultate. Pohlepna pretraga obično se ponaša dobro u slučaju problema kod kojih kvalitet odluke u nekom stanju pretrage ne zavisi od budućih odluka. Pohlepni algoritmi nikad se ne vraćaju u stanje u kojem su već bili. To svojstvo, u slučaju konačnih grafova, obezbeđuje zaustavljanje algoritma.

Primer 4.2. *U slučaju Lojdove slagalice, kao ocena rastojanja od tekućeg do ciljnog stanja može se koristiti zbir* Menhetn rastojanja *svakog od 15 polja slagalice do njegovog ciljnog mesta. Menhetn rastojanje između*

dva polja A i B definiše se kao najmanji broj polja koji je potrebno preći kako bi se došlo od A do B, krećući se isključivo horizontalno ili vertikalno (ovo rastojanje zove se Menhetn, jer podseća na kretanje ulicama Menhetna koje su međusobno normalne ili paralelne: od jednog do drugog bloka moguće je kretati se ulicama, ali nije moguće prolaziti blokove dijagonalno). U slučaju stanja slagalice u korenu levog stabla na narednoj slici, Menhetn rastojanje polja 1 do njegovog pravog mesta je 3, zato što je na tom putu potrebno preći preko dva polja krećući se naviše, a potom jedno polje krećući se ulevo. Mogući su i drugi putevi, ali njihova dužina nije manja. Ukupna ocena tog stanja slagalice je 0+0+0+1+1+2+1+0+3+2+2+0+0+2+2=16. Algoritam pohlepne pretrage prikazan na slici 4.1. traži potez kojim ocena stanja može da se popravi.



Ukoliko je polazna konfiguracija stanje slagalice prikazane u korenu levog stabla na slici, pohlepna pretraga ne može se da nastavi. Naime, dato stanje predstavlja tačku lokalnog minimuma, jer se bilo kojim potezom ta ocena uvećava za 1.

Ukoliko je polazna konfiguracija stanje slagalice prikazane u korenu desnog stabla na slici, ocena stanja se smanjuje pomeranjem polja 4 naviše a zatim i pomeranjem naviše polja 8 i 11. No, nakon toga se dolazi do stanja u kojem se dostiže lokalni minimum i pretraga se zaustavlja.

Navedeni primeri pokazuju da predloženom jednostavnom pohlepnom pretragom nije moguće rešiti Lojdovu slagalicu.

4.1.1 Penjanje uzbrdo

Slični po duhu pohlepnoj pretrazi su algoritmi *penjanja uzbrdo*. Oni imaju svojih specifičnosti, a ponekad se smatraju vrstom pohlepne pretrage.

Algoritmi penjanja uzbrdo pripadaju grupi algoritama *lokalne pretrage* za matematičku optimizaciju. Zadatak je pronaći argument $\mathbf{x}$ koji daje maksimalnu vrednost neke funkcije cilja $f(\mathbf{x})$. Moguće vrednosti tog argumenta čine *skup dopustivih rešenja*, a tražena vrednost argumenta naziva se *tačkom optimuma* ili *optimalnim rešenjem* problema optimizacije. Algoritam treba da kao rezultat uvek vrati neko dopustivo rešenje (iako ne može da tvrdi da je to rešenje zaista maksimalno, tj. optimalno). Inicijalno rešenje obično se bira slučajno. Nastavak postupka sprovodi se u iteracijama i to samo dok se popravlja kvalitet tekućeg rešenja. U svakoj iteraciji novo tekuće rešenje bira se iz specifične *okoline* tekućeg rešenja, tj. iz nekog skupa suseda. Onaj od tih suseda za koji je najveća vrednost funkcije f biće kandidat za novo tekuće rešenje. Ukoliko vrednost funkcije f za to rešenje nije veća od vrednosti za tekuće rešenje, postupak se obustavlja i kao rezultat se vraća tekuće rešenje. Funkcija $f(\mathbf{x})$ ne mora biti neprekidna, već može biti i diskretna i tada se penjanje uzbrdo ponaša kao opisana pohlepna pretraga u grafu.

Algoritmi penjanja uzbrdo obično su veoma jednostavni i često dovoljno uspešni, ali imaju i slabosti. Penjanje uzbrdo ne garantuje pronalaženje optimalnog rešenja. Na primer, algoritam može doći do *tačke lokalnog maksimuma* (kada je u svakom susedu vrednost funkcije cilja manja od tekuće), tada završava rad i vraća tekuću vrednost $\mathbf{x}$, bez ikakve garancije da ne postoji bolji lokalni maksimum ili drugačiji globalni maksimum. Slično, algoritam može da se nađe na *platou* (kada je u svakom susedu vrednost funkcije cilja jednaka tekućoj), tada završava rad jer ne može da odredi sledeću iteraciju i vraća tekuću vrednost $\mathbf{x}$.

Postoje razne varijacije osnovnog penjanja uzbrdo koje pokušavaju da se izbore sa navedenim problemima. Takvo je, na primer, *stohastičko penjanje uzbrdo* u kojem je dozvoljeno u nekim iteracijama birati i rešenje koje nije susedno i nije nužno bolje od tekućeg, a kako bi se povećale šanse da se pređe u deo prostora pretrage u kojem se nalazi tražena tačka maksimuma ciljne funkcije.

Potpuno analogni algoritmima koji na opisani način traže tačku maksimuma ciljne funkcije su algoritmi koji traže tačku minimuma ciljne funkcije.

4.1.2 Metode gradijentnog uspona i spusta

Metode pretrage i matematičke optimizacije često pokušavaju da iskoriste neku zakonitost u strukturi prostora pretrage, odnosno prostora dopustivih rešenja. U slučaju diferencijabilne ciljne funkcije, često se koristi metod koji je sličan opštem metodu penjanja uzbrdo, a koristi gradijent ciljne funkcije.

Definicija 4.1 (Gradijent)**.** *Ukoliko je data diferencijabilna funkcija* $f : \mathbb{R}^n \to \mathbb{R}$, gradijent *je vektor parcijalnih izvoda te funkcije:*

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

Gradijent izračunat u određenoj tački $\mathbf{a} \in \mathbb{R}^n$ predstavlja vektor u prostoru $\mathbb{R}^n$ u čijem smeru funkcija f najbrže raste u okolini tačke $\mathbf{a}$. Stoga se pravljenjem koraka u ovom smeru može približiti tački lokalnog maksimuma, a kretanjem u suprotnom smeru – tački lokalnog minimuma. S obzirom na to da se ne radi o diskretnom prostoru, u praksi se obično ne očekuje pronalaženje sâme tačke lokalnog maksimuma, te se opisani postupak zaustavlja kada, recimo, razlika u vrednosti funkcije f u odnosu na njenu prethodnu vrednost postane dovoljno mala. Ako se primeni kretanje u smeru najbržeg rasta, može se desiti da se „ode predaleko“ i da je vrednost funkcije f u novoj iteraciji lošija od vrednosti u tekućoj iteraciji, te metode zasnovane na gradijentu nisu, u strogom smislu, vrsta penjanja uzbrdo. Na slici 4.2 prikazan je opšti algoritam Gradijentni uspon (ili *najstrmiji uspon*), koji je zasnovan na opisanoj ideji i najjednostavniji među metodama lokalne optimizacije za diferencijabilne funkcije.

Algoritam: Gradijentni uspon

Ulaz: Diferencijabilna funkcija $f(\mathbf{x})$, polazna tačka $\mathbf{a_0}$ i preciznost ε

Izlaz: Aproksimacija tačke maksimuma funkcije f

1: postavi n na 0;
2: **ponavljaj**
3: izračunaj vrednost $\nabla f(\mathbf{a_n})$;
4: izvrši kretanje u smeru gradijenta do sledeće tačke $\mathbf{a_{n+1}}$;
5: uvećaj n za 1;
6: **dok nije ispunjen** uslov $|f(\mathbf{a_n}) - f(\mathbf{a_{n-1}})| \leq \varepsilon |f(\mathbf{a_{n-1}})|$;
7: vrati $\mathbf{a_n}$ kao rešenje.

Slika 4.2: Algoritam Gradijentni uspon.

Navedeni opšti algoritam potrebno je precizirati. Poznavanje gradijenta i proizvoljno kretanje u njegovom smeru ne garantuje nalaženje tačke maksimuma, jer je u zavisnosti od dužine koraka moguće preći preko te tačke, nastaviti dalje i doći do rešenja goreg od tekućeg. Stoga je u svakom koraku potrebno imati pogodnu vrednost α_n koja se koristi u izboru nove tačke (tj. koja određuje koliko daleko će se vršiti kretanje u smeru gradijenta):

$$\mathbf{a_{n+1}} = \mathbf{a_n} + \alpha_n \nabla f(\mathbf{a_n})$$

Vrednosti α_n mogu se definisati na različite načine. Dovoljan uslov za konvergenciju dat je Robins-Monroovim uslovima koji kažu da se za vrednosti α_n mogu uzeti bilo koji brojevi koji zadovoljavaju sledeće uslove:

$$\sum_{n=0}^{\infty} \alpha_n = \infty \qquad \sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

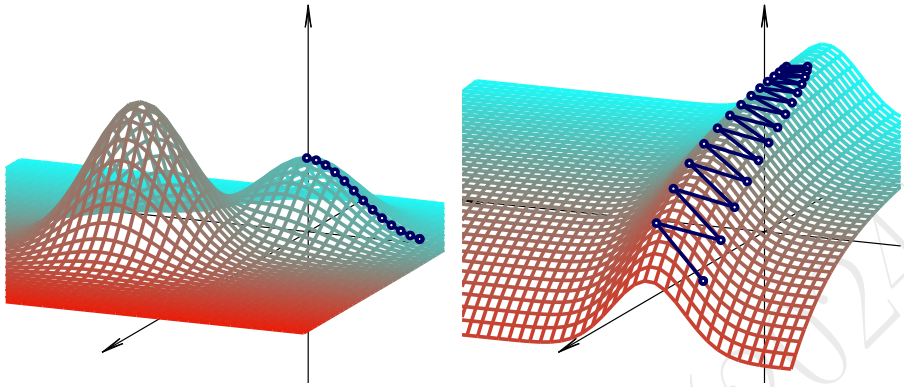
Intuitivno, prvi uslov garantuje da su koraci pretrage dovoljno veliki da pretraga ne uspori prerano i da stoga uopšte ne stigne do tačke maksimuma, dok drugi garantuje da su koraci dovoljno mali da optimizacioni proces ne divergira. Jedan izbor koji zadovoljava ove uslove je $\alpha_n = \frac{1}{n+1}$.

Prikazani algoritam ne garantuje pronalaženje tačke globalnog maksimuma. Naime, rešenje koje algoritam daje može da zavisi od izabrane polazne tačke i može se desiti da u njoj funkcija dostiže samo lokalni (a ne i globalni) maksimum.

Algoritam Gradijentni uspon veoma je jednostavan i često dovoljno uspešan, ali ima i sledeće slabosti (slično kao i penjanje uzbrdo):

Opasnost od lokalnih maksimuma: Algoritam Gradijentni uspon može doći do tačke lokalnog maksimuma i tada će je vratiti kao rezultat. Algoritam nema načina da utvrdi da li se radi o globalnom ili samo lokalnom maksimumu (slika 4.3, levo).

Opasnost od platoa: Platoi predstavljaju oblasti u kojima funkcija cilja ima konstantnu vrednost. Zbog toga je gradijent jednak nuli, te je nemoguće odrediti vrednost za sledeću iteraciju.



Slika 4.3: Situacija u kojoj Gradijentni uspon može da vrati tačku lokalnog maksimuma (levo) i situacija sa grebenom u kojoj Gradijentni uspon otežano dolazi do rešenja (desno).

Neefikasnost u slučaju grebena: Grebeni predstavljaju uske staze koje opadaju ili rastu duž nekog pravca (slika 4.3, desno). U takvim situacijama, kretanje ka većim vrednostima može biti usporeno. Naime, moguće je da se pravac najbržeg rasta funkcije preskače (zbog neodgovarajuće dužine kretanja u smeru gradijenta) i umesto bolje usmerenog rasta (duž grebena) primenjuje se veliki broj cik-cak koraka.

Spora konvergencija: Konvergencija algoritma često je spora. Brže alternative su ili komplikovanije ili imaju dodatne pretpostavke o svojstvima ciljne funkcije (poput jačih varijanti konveksnosti) ili zahtevaju dodatne informacije o ciljnoj funkciji (poput parcijalnih izvoda drugog reda).

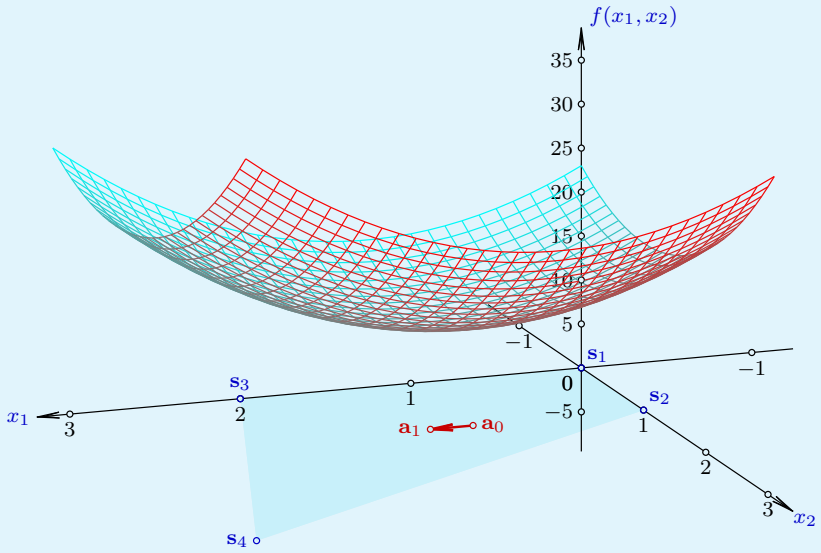
U problemima matematičke optimizacije često se traži i tačka minimuma date funkcije. Za ove probleme može se koristiti opšti algoritam Gradijentni spust (ili *najstrmiji spust*). Jedina razlika u odnosu na algoritam Gradijentni uspon je u tome što se kretanje ne vrši u smeru gradijenta, već u suprotnom smeru, tj. nova tačka bira se na sledeći način: $\mathbf{a_{n+1}} = \mathbf{a_n} - \alpha_n \nabla f(\mathbf{a_n})$. Delovanje ovog algoritma može se intuitivno ilustrovati sledećim primerom: ukoliko je površina zemlje aproksimirana glatkom površinom, onda bi se primenom algoritma Gradijentni spust od tačke izvora neke reke stiglo do njenog ušća.

Primer 4.3. *Potrebno je izgraditi medicinsku stanicu koja bi opsluživala četiri sportske lokacije čije su koordinate* $\mathbf{s}_1 = (0,0)$, $\mathbf{s}_2 = (0,1)$, $\mathbf{s}_3 = (2,0)$ *i* $\mathbf{s}_4 = (3,3)$. *Stanica bi trebalo da bude relativno blizu svim lokacijama. Jedan povoljan izbor njene lokacije je tačka* $\mathbf{x} = (x_1, x_2)$ *takva da je zbir*

$$f(\mathbf{x}) = \sum_{i=1}^{4} \|\mathbf{x} - \mathbf{s}_i\|^2$$

minimalan. Zapisano drugačije, funkcija f *jednaka je:*

$$\begin{aligned} f(\mathbf{x}) &= (x_1-0)^2 + (x_2-0)^2 + (x_1-0)^2 + (x_2-1)^2 + \\ &\quad (x_1-2)^2 + (x_2-0)^2 + (x_1-3)^2 + (x_2-3)^2 \\ &= 4x_1^2 + 4x_2^2 - 10x_1 - 8x_2 + 23 \end{aligned}$$



Gradijent funkcije f jednak je

$$\nabla f(\mathbf{x}) = (8x_1 - 10, 8x_2 - 8)$$

Neka je polazna tačka $\mathbf{a_0} = (1,1)$ i $\varepsilon = 10^{-6}$. Vrednost gradijenta $\nabla f(\mathbf{a_0})$ u prvoj iteraciji je $(-2,0)$. Neka je vrednost parametra α_n jednaka $\frac{1}{n+1}$. Sledeća tabela prikazuje kako se menjaju relevantne vrednosti prilikom primene gradijentnog spusta:

n	$\mathbf{a_n}$	α_n	$\nabla f(a_n)$
0	$(1,1)$	1	$(-2,0)$
1	$(3,1)$	$1/2$	$(14,0)$
2	$(-4,1)$	$1/3$	$(-42,0)$
3	$(10,1)$	$1/4$	$(70,0)$
4	$(-7.5,1)$	$1/5$	$(-70,0)$
5	$(6.5,1)$	$1/6$	$(42,0)$
6	$(-0.5,1)$	$1/7$	$(-14,0)$
7	$(1.5,1)$	$1/8$	$(2,0)$
8	$(1.25,1)$	$1/9$	$(0,0)$
9	$(1.25,1)$	-	–

Primetno je da algoritam pravi nekoliko velikih koraka, što je posledica relativno velike vrednosti α_n i strmosti funkcije. Međutim, kako se vrednosti α_n smanjuju, a tačka $\mathbf{a_n}$ bliži rešenju u čijoj je okolini funkcija manje strma, koraci postaju manji i dolazi se do tačnog rešenja. U opštem slučaju, retko se dešava zaustavljanje sa tačnim rešenjem. U ovom konkretnom slučaju, rešenje je moglo da bude pronađeno i analitički — rešavanjem jednačina $\nabla f(\mathbf{x}) = 0$, ali to u opštem slučaju nije moguće.

4.2 Pretraga Prvo najbolji

Pristup pretrage *prvo najbolji* (eng. *best-first search*) predstavlja osnovu za različite algoritme pretrage, pri čemu je prostor stanja i akcija za neki problem opisan u vidu grafa. Rešenjem se smatra niz čvorova (tj. put) od polaznog do ciljnog čvora u grafu. Kao i u drugim algoritmima informisane pretrage, koristi se funkcija evaluacije koja ocenjuje kvalitet čvorova i usmerava pretragu. Osnovna ideja je da se u pretrazi prednost daje elementima sa boljom ocenom.

U toku primene algoritma, svakom čvoru stabla pretrage pridružuje se informacija o njegovom prethodniku (roditelju) u mogućem rešenju, isto kao u Dajkstrinom algoritmu. Jednostavnosti radi, u nastavku će se često isto označavati čvor stabla pretrage i stanje koje mu je pridruženo.

Da bi se izbegle beskonačne petlje (tj. beskonačno obrađivanje istog čvora), da bi se obezbedila potpunost i da bi se omogućilo popravljanje već nađenih puteva, održava se spisak svih posećenih čvorova i to u vidu dve liste:

- *otvorena lista* (ili *lista otvorenih stanja*) – lista već dosegnutih čvorova koji treba da budu obrađeni;

- *zatvorena lista* (ili *lista zatvorenih stanja*) – lista čvorova koji su već obrađeni.

Na početku je u otvorenoj listi samo polazni čvor, a zatvorena lista je prazna. U svakoj iteraciji analizira se element otvorene liste sa najboljom ocenom (implementacija otvorene liste treba da omogućava efikasan pristup takvom elementu) i obrađuju se iz njega neposredno dostupni čvorovi. Ukoliko se naiđe na ciljni čvor – zadatak je rešen i algoritam završava rad. Precizniji opis algoritma dat je na slici 4.4.

Algoritam: Prvo najbolji

Ulaz: Graf G, polazni čvor i ciljni čvor

Izlaz: Niz koraka od polaznog do ciljnog čvora (ako postoji put između ova dva stanja)

1: postaviti da je inicijalna vrednost funkcije g za polazni čvor jednaka 0, a za sve ostale čvorove jednaka $+\infty$ (beskonačno);
2: zatvorenu listu postavi na praznu listu, a u otvorenu listu stavi samo polazni čvor (sa izračunatom vrednošću funkcije f);
3: **dok god** ima elemenata u otvorenoj listi **radi**
4: izaberi čvor n (*tekući čvor*) iz otvorene liste koji ima najbolju ocenu $f(n)$;
5: **ako** je n ciljni čvor **onda**
6: izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora (idući unazad — od ciljnog čvora);
7: **za** svaki čvor m koji je direktno dostupan iz n **radi**
8: **ako** m nije ni u otvorenoj ni u zatvorenoj listi **onda**
9: dodaj ga u otvorenu listu i označi n kao njegovog roditelja;
10: izbaci n iz otvorene liste i dodaj ga u zatvorenu listu;
11: izvesti da traženi put ne postoji (otvorena lista je prazna i uspeh nije prijavljen).

Slika 4.4: Algoritam Prvo najbolji.

Algoritam Prvo najbolji ne daje nužno optimalno rešenje. Ako je broj čvorova i grana u grafu konačan, algoritma se zaustavlja i ima svojstvo potpunosti, o čemu govori naredna teorema.

Teorema 4.1. *Ako je broj stanja i akcija konačan, algoritam Prvo najbolji se zaustavlja i nalazi traženi put ako on postoji.*

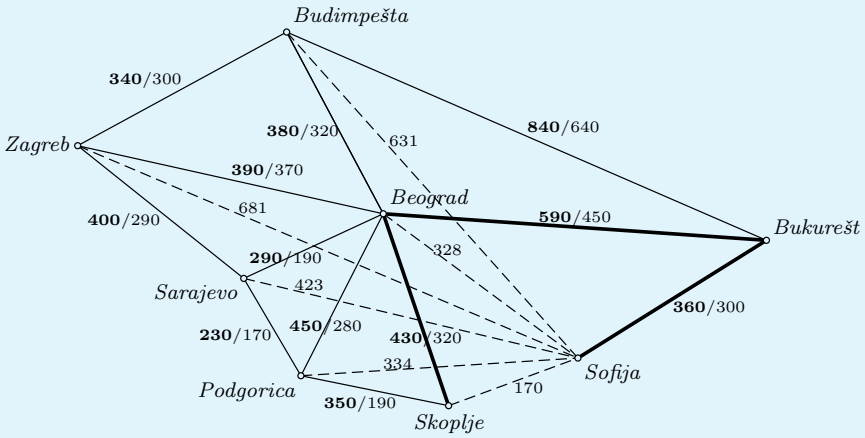
Ako funkcija $f(n)$ vraća dubinu čvora n u BFS obilasku grafa počev od polaznog čvora, onda se navedeni algoritam ponaša kao algoritam obilaska u širinu (za neki poredak čvorova). Ako funkcija $f(n)$ vraća zbir cena grana od polaznog čvora do čvora n, onda se navedeni algoritam ponaša kao Dajkstrin algoritam.

Opšti algoritam Prvo najbolji predstavlja bitnu modifikaciju algoritma jednostavnog pohlepnog pristupa. Iako oba u jednom čvoru koriste slično navođenje i biraju (najpre) najbolji susedni čvor (tj. čvor n sa najboljom vrednošću $f(n)$), algoritam Prvo najbolji, za razliku od jednostavnog pohlepnog pristupa, omogućava vraćanje na čvorove koji su posećeni a nisu ispitani (jer je neka od alternativa obećavala više). Ovim pristupom, zahvaljujući alternativama u otvorenoj listi, omogućava se uspešan nastavak pretrage i u slučajevima kada bi pohlepna pretraga naišla na plato ili na tačku lokalnog optimuma (tj. kada ne može jednim korakom da se popravi ocena tekućeg stanja).

Primer 4.4. *Razmotrimo ponovo primer 4.1 i zadatak nalaženja puta od Budimpešte do Sofije. Neka se ponovo kao funkcija evaluacije f koristi vazdušno rastojanje do ciljnog grada. U otvorenoj listi je najpre samo Budimpešta. To nije ciljni čvor i u otvorenu listu dodaju se direktno dostupni gradovi – Zagreb, Beograd, Bukurešt. Budimpešta prelazi u zatvorenu listu. Najbolje ocenjeni čvor iz otvorene liste je sada Bukurešt. U otvorenu listu dodaje se Sofija a Bukurešt prelazi u zatvorenu listu. Najbolje ocenjeni čvor iz otvorene liste je sada Sofija, to je ciljni čvor i vraća se put Budimpešta-Bukurešt-Sofija, u ovom konkretnom slučaju, to je isti put koji je vratila i pohlepna pretraga (i on nije najbolji mogući).*

Razmotrimo ponovo i drugi zadatak iz primera 4.1: pretpostavimo da je Skoplje polazni čvor, da je ciljni čvor (ponovo) Sofija, ali i da ne postoji direktan put između ova dva grada. U otvorenu listu se najpre dodaje Skoplje i to je prvi tekući čvor. U otvorenu listu dodaju se njemu direktno dostupni gradovi Beograd i Podgorica čije su ocene 328 i 334 (sa Skopljem kao roditeljem). Skoplje prelazi u zatvorenu listu. Sledeći

tekući čvor je Beograd (jer ima najbolju ocenu u otvorenoj listi) i u otvorenu listu ulaze Budimpešta, Zagreb, Sarajevo i Bukurešt (sa Beogradom kao roditeljem). Beograd prelazi u zatvorenu listu. Sledeći tekući čvor je Bukurešt i u otvorenu listu ulazi Sofija (sa Bukureštom kao roditeljem). Sledeći tekući čvor je Sofija i pretraga se završava, vrativši put Skoplje-Beograd-Bukurešt-Sofija. Podsetimo se da je u ovoj situaciji pohlepna pretraga bila nemoćna da pronađe put od Skoplja do Sofije (videti primer 4.1). Naredna slika ilustruje dobijeno rešenje.



Primer 4.5. *U slučaju primera slagalice diskutovanog u primeru 4.2, situacija prikazana levo predstavlja tačku lokalnog minimuma, zbog čega se pohlepna pretraga zaustavlja. Algoritam Prvo najbolji će odabrati jedan od mogućih poteza, ali će alternativno stanje čuvati u otvorenoj listi i možda ga obraditi kasnije. U situaciji prikazanoj desno, pohlepnom pretragom se prazno polje spušta do donjeg desnog ugla, čime se dolazi do tačke lokalnog optimuma i pohlepna pretraga ne može da nastavi. Međutim, u slučaju algoritma Prvo najbolji, stanja koja su bila alternative ispitanim stanjima su i dalje u otvorenoj listi i ispituju se dalje. Stoga je algoritam Prvo najbolji u stanju da reši slagalicu, ali ne garantuje nalaženje rešenja koje se sastoji od najmanjeg broja poteza.*

4.3 Algoritam A*

Algoritam A* pretraga ili, kraće, algoritam A* (čita se „a zvezda“, eng. „*a star*“) za određivanje puta između dva čvora grafa, jedan je od fundamentalnih i najpopularnijih algoritama veštačke inteligencije. Zasnovan je na korišćenju heuristika za usmeravanje pretrage, ali ipak ima, pod određenim uslovima, svojstvo potpunosti i optimalnosti. Prvu verziju algoritma A* razvili su Hart (Peter Hart), Nilson (Nils Nilson) i Rafael (Bertram Raphael) 1968. godine, a u narednim godinama uvedeno je nekoliko modifikacija.

Algoritam A* je varijanta algoritma Prvo najbolji u kojoj se koristi funkcija evaluacije f koja ima sledeću specifičnu formu:

$$f(n) = g(n) + h(n),$$

gde je $g(n)$ cena puta od polaznog čvora do čvora n, a $h(n)$ je *procenjena* (*heuristička*) cena najjeftinijeg puta od čvora n do ciljnog čvora. Dok se traga za najkraćim putem, za vrednost $g(n)$ koristi se trenutno poznata minimalna cena od polaznog čvora do čvora n i ona se može menjati tokom primene algoritma. S druge strane, vrednost $h(n)$ zasnovana je na proceni i za svaki čvor njena vrednost je nepromenljiva tokom primene algoritma. Od kvaliteta heuristike u velikoj meri zavisi ponašanje i efikasnost algoritma. Izbor kvalitetne heuristike jedan je od najvažnijih izazova u dizajniranju konkretnih implementacija algoritma A*.

Opis algoritma A* dat je na slici 4.5. Prilikom dodavanja čvora m u otvorenu listu, vrednost $g(m)$ se može izračunati na inkrementalan i efikasan način: vrednost $g(m)$ jednaka je zbiru vrednosti funkcije g za roditelja čvora m i ceni puta od roditelja do m.

Ako algoritam naiđe na čvor m koji je već u otvorenoj ili zatvorenoj listi, to znači da je pronađen *novi put* do već posećenog čvora m. Tada se proverava da li je put od polaznog čvora do čvora m preko čvora n bolji od postojećeg puta. Ako jeste, potrebno je ažurirati vrednost $g(m)$. To može da se desi i za čvor m koji pripada zatvorenoj listi: ako to jeste slučaj, potrebno je čvor m vratiti u otvorenu listu (jer je moguće da se mogu popraviti i neki putevi preko čvora m).

Algoritam: A*

Ulaz: Graf G, polazni čvor i ciljni čvor, heuristička funkcija h

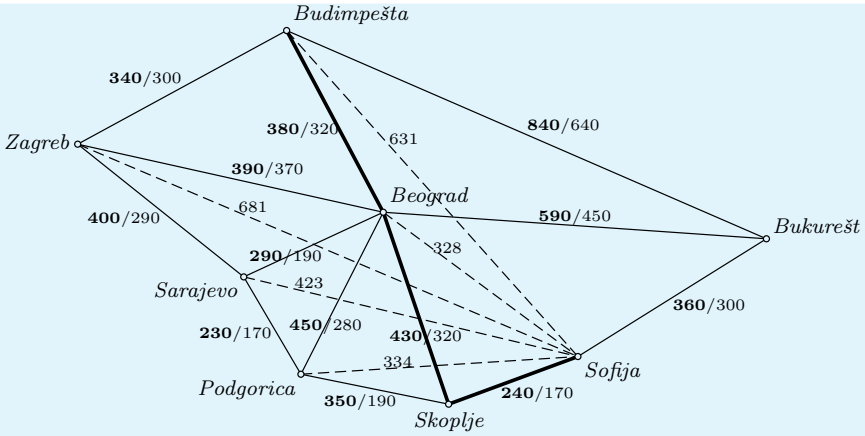
Izlaz: Put od polaznog do ciljnog čvora ili neuspeh

1: zatvorenu listu postavi na praznu listu, u otvorenu listu stavi samo polazni čvor (sa izračunatom vrednošću funkcije f);
2: **dok god** ima elemenata u otvorenoj listi **radi**
3: izaberi *tekući čvor* n iz otvorene liste koji ima najbolju ocenu $f(n)$;
4: **ako** je n ciljni čvor **onda**
5: izvesti o uspehu i vrati rešenje konstruišući put od polaznog do ciljnog čvora (idući unazad — od ciljnog čvora);
6: **za** svaki čvor m koji je direktno dostupan iz n **radi**
7: **ako** m nije ni u otvorenoj ni u zatvorenoj listi **onda**
8: dodaj ga u otvorenu listu i označi n kao njegovog roditelja; izračunaj i pridruži vrednosti $g(m)$ i $f(m)$ čvoru m;
9: **inače**
10: **ako** je cena puta od polaznog čvora do čvora m preko čvora n niža od cene postojećeg puta do m (koja je trenutno jednaka $g(m)$) **onda**
11: promeni informaciju o roditelju čvora m na čvor n i ažuriraj vrednosti $g(m)$ i $f(m)$;
12: **ako** je čvor m bio u zatvorenoj listi **onda**
13: prebaci ga u otvorenu;
14: izbaci n iz otvorene liste i dodaj ga u zatvorenu listu;
15: izvesti da traženi put ne postoji.

Slika 4.5: Algoritam A*.

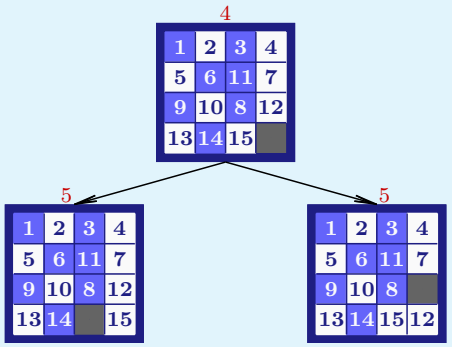
Primer 4.6. *U primeru pronalaženja najkraćih puteva između gradova, ako su poznata rastojanja između gradova vazdušnim putem, algoritam A* može kao heurističku vrednost $h(n)$ da koristi vazdušno rastojanje od čvora n do ciljnog čvora, kao u primeru 4.1. Naredna tabela i slika (kopnena rastojanja ispisana su podebljanim, a vazdušna običnim ciframa) ilustruju izvršavanje algoritma A* na primeru nalaženja najkraćeg puta od Budimpešte do Sofije.*

tekući čvor	*stanje otvorene liste [čvor(roditelj,g+h)]*	*u zatvorenu listu se dodaje*
	Bu(-,0+631)	
Bu	*Bg(Bu,380+328), Zg(Bu,340+681), Bk(Bu,840+300),*	*Bu(-)*
Bg	*Sk(Bg,810+170), Zg(Bu,340+681), Bk(Bu,840+300), Sa(Bg,670+423), Pg(Bg,830+334)*	*Bg(Bu)*
Sk	*Zg(Bu,340+681), So(Sk, 1050+0), Bk(Bu,840+300), Sa(Bg,670+423), Pg(Bg,830+334)*	*Sk(Bg)*
Zg	*So(Sk, 1050+0), Bk(Bu,840+300), Sa(Bg,670+423), Pg(Bg,830+334)*	*Zg(Bu)*
So	*Bk(Bu,840+300), Sa(Bg,670+423), Pg(Bg,830+334)*	



Pronađeni put je Budimpešta-Beograd-Skoplje-Sofija i on je kraći od puta Budimpešta-Bukurešt-Sofija koji pronalazi pohlepna pretraga i pretraga Prvo najbolji (štaviše, ovaj pronađeni put je najkraći mogući).

Primer 4.7. *U slučaju Lojdove slagalice, kao u primeru 4.2, za heuristiku se koristi zbir Menhetn rastojanja svakog od 15 polja slagalice do njegovog ciljnog mesta. Naredna slika prikazuje stanje slagalice i dva moguća naslednika, pri čemu oba imaju ocenu veću od ocene polaznog stanja. Stoga, kako se u polaznom stanju dostiže lokalni minimum, pristup čiste pohlepne pretrage sa izabranom heuristikom nemoćan je već na početku.*



Za isto polazno stanje, algoritam A pronalazi rešenje od šest poteza – gore, levo, gore, desno, dole, dole. Stablo pretrage vršene algoritmom A* prikazano je na slici 4.6.*

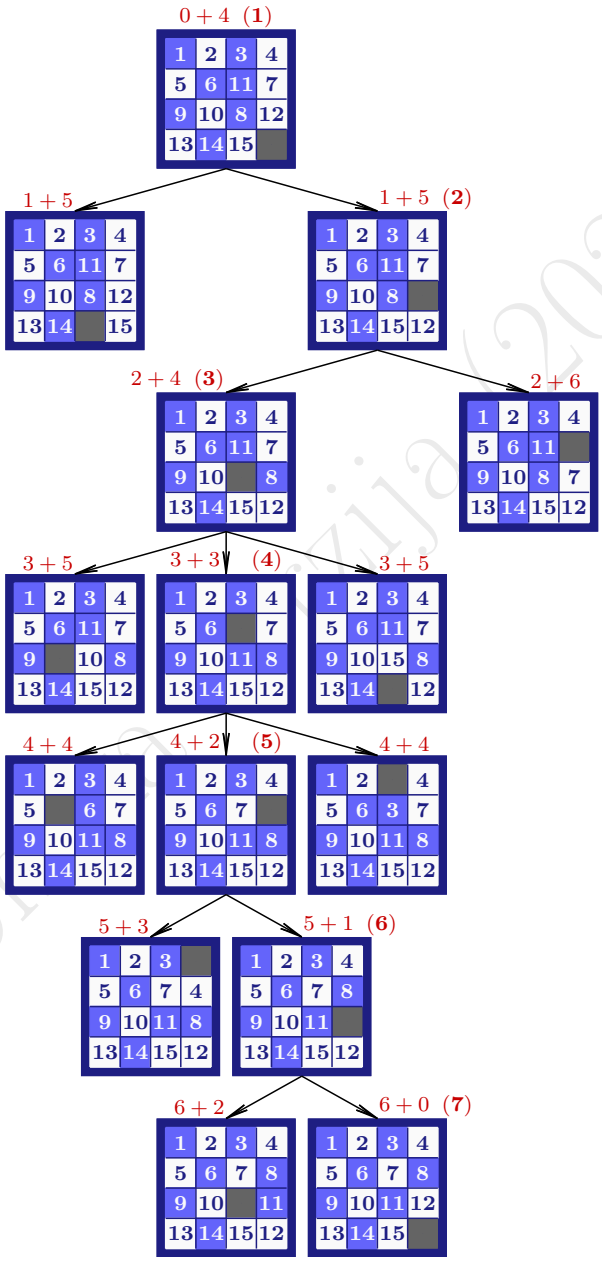
Korišćenje algoritma A* nije uvek jednostavno. Često je algoritam potrebno prilagoditi specifičnom problemu a uvek je, u kontekstu aplikacija koje rade u realnom vremenu, važno imati u vidu vremensku složenost, prostornu složenost, upravljanje memorijom i različite dodatne faktore. Neki od dodatnih, specifičnih zahteva mogu da iziskuju dodatno matematičko znanje i izračunavanja i specifične implementacione tehnike i strukture. Svi ti moduli treba da budu uklopljeni u kompaktan i efikasan sistem za nalaženje puta.

4.3.1 Svojstva algoritma A*

Pod određenim pretpostavkama, može se dokazati da se algoritam A* zaustavlja, da je potpun i da je optimalan.

Zaustavljanje: Ako su broj čvorova i broj grana konačni, algoritam A* se zaustavlja, kao i svaki algoritam tipa Prvo najbolji. Na grafovima sa beskonačnim brojem čvorova, sa konačnim stepenom i sa cenama grana većim od neke pozitivne konstante, A* se zaustavlja ako postoji rešenje.

Potpunost: Ako su broj čvorova i broj grana konačni, ako postoji put između dva čvora, algoritam A* će, kao i svaki algoritam tipe Prvo najbolji, naći jedan takav (ukoliko je raspoloživo dovoljno vremena i memorijskog prostora). Čak i ako je heuristička funkcija veoma loša, ciljni čvor će biti dostignut u konačnom broju koraka.



Slika 4.6: Stablo pretrage vršene algoritmom A*, na primeru slagalice u kojem se u polaznom stanju dostiže lokalni minimum. U prikazu stabla, među naslednicima svakog stanja su samo stanja koja nisu već dodata u zatvorenu listu. U zagradama su navedeni redni brojevi pod kojim stanja postaju tekuća (poredak obilaska potomaka mogao je da bude i drugačiji).

Optimalnost: Algoritam A* u opštem slučaju nema svojstvo optimalnosti, ono važi samo ukoliko heuristika koja se koristi zadovoljava neke uslove.

Definicija 4.2 (Dopustiva heuristika)**.** *Heuristika h je dopustiva (eng.* admissible*) ako nikada ne precenjuje stvarno rastojanje između tekućeg čvora i ciljnog čvora, tj. ako za svaki čvor važi:*

$$h(n) \leq h^*(n),$$

gde je $h^*(n)$ cena najkraćeg puta od čvora n do ciljnog čvora (tj. h^* je idealna, optimalna heuristika).

Tvrđenje da dopustiva heuristika obezbeđuje optimalnost navodimo bez dokaza:

Teorema 4.2. *Ako se u algoritmu A* koristi dopustiva heuristika, ako je pronađen put do ciljnog čvora, onda je on sigurno optimalan.*

Ukoliko heuristika h nije dopustiva, ali ne precenjuje stvarnu cenu za više od d, onda je cena puta koji će pronaći algoritam A* manja ili jednaka od cene najkraćeg puta uvećane za d.

Definicija 4.3 (Konzistentna heuristika)**.** *Heuristika h je* konzistentna *(eng.* consistent*) ako ima vrednost* 0 *za ciljni čvor i za bilo koja dva susedna čvora n i m važi:*

$$c(n, m) + h(m) \geq h(n)$$

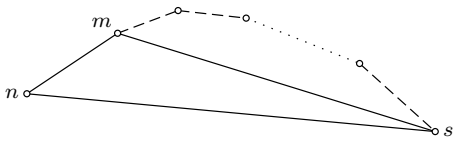
gde je $c(n, m)$ cena pridružena (moguće usmerenoj) grani (n, m).

Ako je funkcija h konzistentna, onda je ona i dopustiva (kao što tvrdi naredna teorema). Obratno ne važi nužno: funkcija h može da bude dopustiva, a da ne bude konzistentna.

Teorema 4.3. *Ako je h konzistentna heuristika, onda je ona i dopustiva.*

Dokaz: Neka je $h^*(n)$ jednako najkraćem rastojanju od čvora n do ciljnog čvora s (tj. neka je h^* optimalna heuristika). Dokažimo da, ako je h konzistentna heuristika, onda za svaki čvor n važi $h(n) \leq h^*(n)$. Dokaz izvedimo matematičkom indukcijom po broju čvorova na najkraćem putu između n i ciljnog čvora s.

Ako između n i s na najkraćem putu nema čvorova, onda iz uslova konzistentnosti važi $c(n, s)+h(s) \geq h(n)$, pa kako je $h^*(n) = c(n, s)$ i $h(s) = 0$, važi $h^*(n) \geq h(n)$.



Pretpostavimo da tvrđenje važi za svaki čvor za koji je broj čvorova do ciljnog čvora na najkraćem putu manji od k, za $k > 0$. Ako između n i s na najkraćem putu ima k čvorova, gde je $k > 0$, neka je m prvi čvor na koji se naiđe posle čvora n na najkraćem putu do s. Od čvora m do čvora s na najkraćem putu ima $k - 1$ čvorova, pa na osnovu induktivne hipoteze ($h^*(m) \geq h(m)$) i na osnovu svojstva konzistentnosti važi

$$h^*(n) = c(n, m) + h^*(m) \geq c(n, m) + h(m) \geq h(n),$$

što je i trebalo dokazati. □

Na osnovu teorema 4.2 i 4.3, sledi da algoritam A* ima svojstvo optimalnosti ako se koristi konzistentna heuristika h. Štaviše, u tom slučaju algoritam A* može da se pojednostavi: nije potrebno proveravati da li je put preko tekućeg čvora do jednom zatvorenog čvora bolji od postojećeg (jer sigurno nije). Dokaz ovog tvrđenja, kao i dokaz optimalnosti algoritma u slučaju korišćenja konzistentne heuristike (koji se ne oslanja na teoremu 4.2), dati su u nastavku.

Lema 4.1. *Ako je h konzistentna heuristika, onda su, u svakom trenutku primene algoritma, vrednosti $f(n)$ duž puta od polaznog do tekućeg čvora neopadajuće.*

Dokaz: Ako je u nekom trenutku primene algoritma čvor m tekući i ako je njegov roditelj čvor n, onda važi:

$$f(m) = g(m) + h(m) = g(n) + c(n, m) + h(m) \geq g(n) + h(n) = f(n)$$

Tvrđenje leme onda sledi na osnovu jednostavnog induktivnog argumenta. □

Lema 4.2. *Ako je h konzistentna heuristika, za niz čvorova n redom proglašenih za tekuće, niz vrednosti $f(n)$ čini neopadajući niz.*

Dokaz: U svakoj iteraciji, algoritam bira za tekući čvor čvor iz otvorene liste sa najmanjom vrednošću $f(n)$ (te svi preostali čvorovi u skupu otvorenih čvorova imaju veće ili jednake vrednosti f). Svi budući tekući čvorovi su preostali čvorovi iz otvorene liste, ili njihovi potomci. Na osnovu prethodne leme, onda sledi da svi tekući čvorovi nakon tekućeg čvora n imaju vrednosti f veće ili jednake $f(n)$. Kako ovo važi za svaki tekući čvor n, sledi tvrđenje leme, tj. algoritam proglašava čvorove tekućim u neopadajućem poretku po $f(n)$. □

Lema 4.3. *Ako je h konzistentna heuristika, kada neki čvor n postane tekući po prvi put, do njega je već pronađen optimalan put.*

Dokaz: Kada algoritam proglasi neki čvor n tekućim po prvi put, on ima neke vrednosti $g(n) = g_0$ i $f(n) = f_0$. Pretpostavimo da $g(n)$ nije optimalna cena puta od polaznog čvora i pretpostavimo da je optimalan put do istog čvora moguće dostići u nekoj kasnijoj iteraciji, kada će mu biti pridružene vrednosti g_1 i f_1. Kako je g_1 cena optimalnog puta do n, važi $g_0 > g_1$, pa i $g_0 + h(n) > g_1 + h(n)$, tj. $f_0 > f_1$. S druge strane, na osnovu prethodne leme, važi $f_0 \leq f_1$, što daje kontradikciju. □

Teorema 4.4. *Ako se u algoritmu A* koristi konzistentna heuristika h, onda:*

- *ako je pronađen put do ciljnog čvora, onda je on sigurno optimalan;*
- *za čvorove dostupne iz tekućeg čvora koji su već zatvoreni, ne treba da se proverava da li njihova vrednost g može da se ažurira.*

Dokaz: Algoritam vraća nađeni put čim ciljni čvor po prvi put postane tekući. Na osnovu leme 4.3, ako je h konzistentna heuristika, kad ciljni čvor postane tekući, do njega je već pronađen optimalan put, što daje prvi deo teoreme. Iz leme 4.3 neposredno sledi i drugi deo teoreme. □

Navedena teorema u slučaju konzistentne heuristike obezbeđuje jednostavniju i efikasniju, a pritom optimalnu verziju algoritma A*.

Složenost: Složenost algoritma A* (i vremenska i prostorna) bitno zavisi od heuristike. Ako je heuristika jednaka nuli, i broj otvorenih čvorova, i prostorna i vremenska složenost algoritma A* jednake su kao za Dajkstrin algoritam. Često je važno i razmatranje složenosti algoritma A* u zavisnosti od dužine najkraćeg puta. Ukoliko svaki čvor ima b susednih čvorova, a najkraći put od nekog čvora do ciljnog je dužine d, onda je složenost u najgorem slučaju jednaka $O(b^d)$, tj. broj obrađenih čvorova eksponencijalno zavisi od d. Može se dokazati da broj obrađenih čvorova polinomski zavisi od d ako heuristika h zadovoljava sledeći uslov:

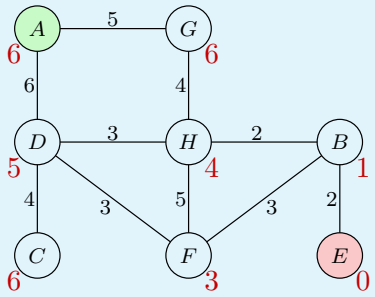
$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

gde je h^* idealna heuristika, tj. funkcija koja vraća cenu najkraćeg puta od čvora n do ciljnog čvora.

Ukoliko je c najkraće rastojanje od polaznog čvora n do ciljnog čvora i ukoliko se koristi dopustiva heuristika, može se dokazati da će algoritam A* posetiti sve čvorove za koje važi $f(m) < c$, kao i neke čvorove m za koje važi $f(m) = c$.

Za algoritam A* se kaže i da je *optimalno efektivan*, jer je dokazano da ne postoji algoritam koji je potpun, a uvek posećuje manje od ili jednako čvorova kao algoritam A*.

Primer 4.8. *Zadatak je naći najkraći put od čvora A do čvora E u sledećem grafu (pored čvorova grafa zapisane su procenjene dužine puta do čvora E, tj. vrednosti funkcije h).*



Naredna tabela ilustruje primenu algoritma A (odgovarajuće stablo pretrage prikazano je na slici 4.7).*

tekući čvor	*stanje otvorene liste [čvor(roditelj,g+h)]*	*u zatvorenu listu se dodaje*
	A(-, 0+6)	
A(6)	*D(A,6+5), G(A,5+6)*	*A(-)*
D(11)	*G(A,5+6), F(D,9+3), H(D,9+4), C(D,10+6)*	*D(A)*
G(11)	*F(D,9+3), H(D,9+4), C(D,10+6)*	*G(A)*
F(12)	*B(F,12+1), H(D,9+4), C(D,10+6)*	*F(D)*
B(13)	*H(D,9+4), E(B,14+0), C(D,10+6)*	*B(F)*
H(13)	*B(H,11+1), E(B,14+0), C(D,10+6)*	*H(D)*
B(12)	*E(B,13+0), C(D,10+6)*	*B(H)*
E(13)	*C(D,10+6)*	

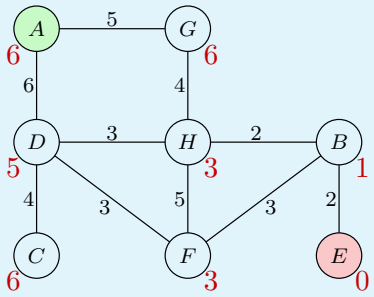
Na kraju primene algoritma, kada je čvor E postao tekući čvor, konstruiše se traženi put – koristeći informacije o roditeljima za čvorove iz zatvorene liste: $A-D-H-B-E$. Korišćena heuristika je dopustiva, pa je pronađeni put optimalan.

Za neke heuristike (kao što je, na primer, euklidsko rastojanje) postoji jednostavan, opšti argument da je dopustiva i konzistentna, a u nekim situacijama, kao što je i graf u ovom primeru, dopustivost i konzistentnost mogu da se ispitaju samo neposrednim proveravanjem za sve čvorove. Upotrebljena heuristika h nije konzistentna jer važi $c(H,B) + h(B) = 2 + 1 < 4 = h(H)$ (primetimo da vrednosti f za čvorove koji postaju tekući nisu neopadajuće). Zato je nužno i za zatvorene čvorove proveravati da li se put do njih može popraviti. To i jeste bio slučaj za čvor B: u koraku u kojem se H briše iz zatvorene liste, u nju se dodaje čvor B jer je do njega pronađen bolji put (preko H) od ranije postojećeg. Ukoliko to ne bi bilo rađeno, algoritam bi se ponašao na sledeći način:

tekući čvor	*stanje otvorene liste [čvor(roditelj,g+h)]*	*u zatvorenu listu se dodaje*
	A(-, 0+6)	
A(6)	*D(A,6+5), G(A,5+6)*	*A(-)*
D(11)	*G(A,5+6), F(D,9+3), H(D,9+4), C(D,10+6)*	*D(A)*
G(11)	*F(D,9+3), H(D,9+4), C(D,10+6)*	*G(A)*
F(12)	*B(F,12+1), H(D,9+4), C(D,10+6)*	*F(D)*
B(13)	*H(D,9+4), E(B,14+0), C(D,10+6)*	*B(F)*
H(13)	*E(B,14+0), C(D,10+6)*	*H(D)*
E(14)	*C(D,10+6)*	

Na kraju primene algoritma, kada je čvor E postao tekući čvor, konstruiše se put: A−D−F−B−E. Ovo jeste put od čvora A do čvora E, ali nije najkraći mogući. Ovo ponašanje posledica je činjenice da funkcija h nije konzistentna: kada heuristika nije konzistentna, neophodno je proveravati i zatvorene čvorove.

Ukoliko se za isti problem koristi konzistentna heuristika, rezultat će biti optimalan put od A do E, a neće biti potrebno proveravati jednom zatvorene čvorove. U narednom primeru koristi se konzistentna heuristika h čija se vrednost razlikuje u odnosu na prethodnu samo za čvor H i daje optimalni put A−D−H−B−E.



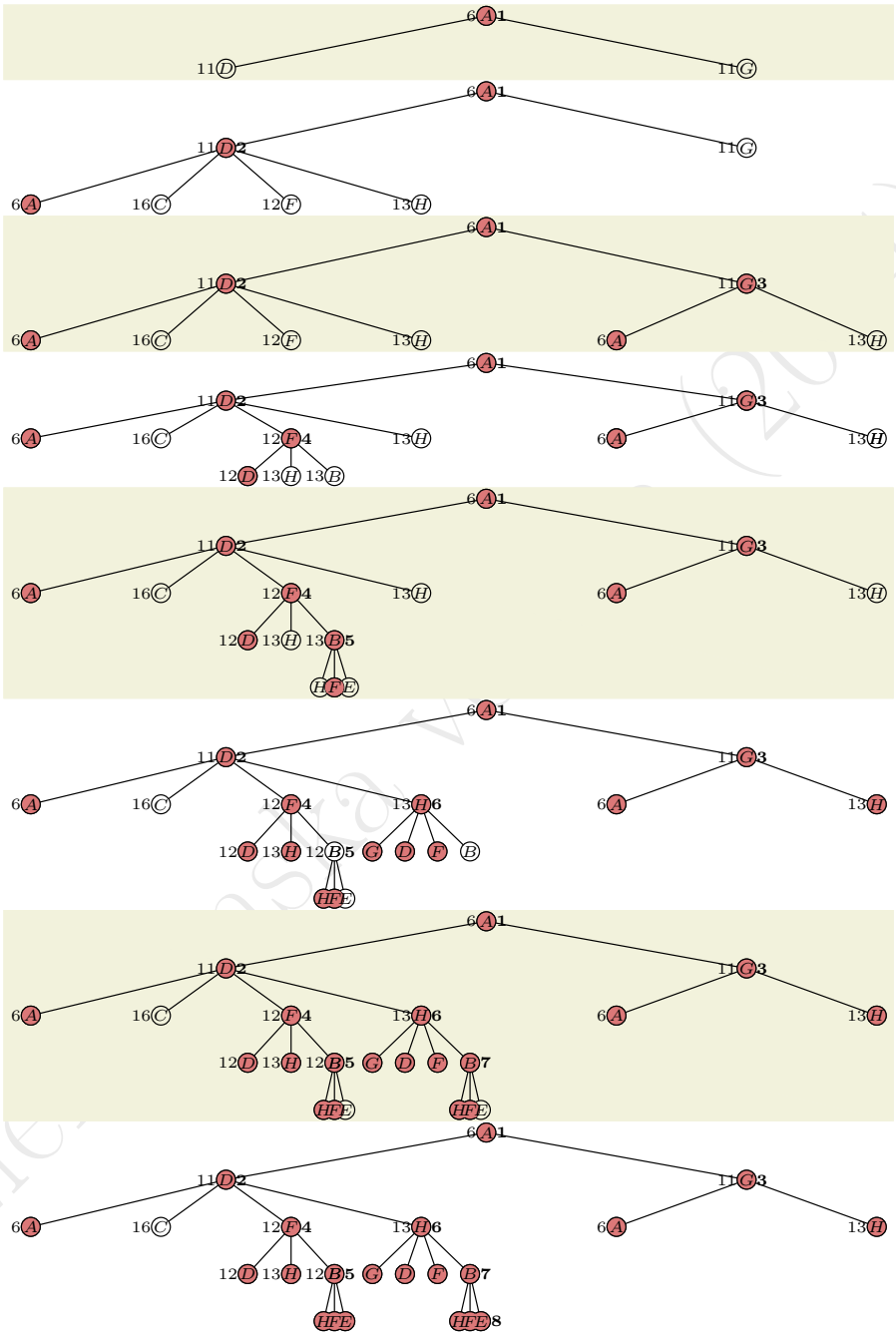
tekući čvor	*stanje otvorene liste [čvor(roditelj,g+h)]*	*u zatvorenu listu se dodaje*
	A(-, 0+6)	
A(6)	*D(A,6+5), G(A,5+6)*	*A(-)*
D(11)	*G(A,5+6), H(D,9+3), F(D,9+3), C(D,10+6)*	*D(A)*
G(11)	*H(D,9+3), F(D,9+3), C(D,10+6)*	*G(A)*
H(12)	*B(H,11+1), F(D,9+3), C(D,10+6)*	*H(D)*
B(12)	*F(D,9+3), E(B,13+0), C(D,10+6)*	*B(H)*
F(12)	*E(B,13+0), C(D,10+6)*	*F(D)*
E(13)	*C(D,10+6)*	

4.3.2 Odnos algoritma A* sa drugim algoritmima pretrage grafova

Obilasci grafa u dubinu i širinu mogu se realizovati kao specijalni slučajevi algoritma A*. Algoritam A* je specijalan slučaj metoda Prvo najbolji (ako se u algoritmu Prvo najbolji vrši ažuriranje puteva za čvorove i iz otvorene i iz zatvorene liste).

Dajkstrin algoritam je specijalni slučaj algoritma A* u kojem je $h(n) = 0$ za svaki čvor n. Ovakva funkcija h je konzistentna i garantuje nalaženje optimalnog puta. Skup otvorenih čvorova tada se širi ravnomerno, slično koncentričnim krugovima oko polaznog čvora, baš kao kod Dajkstrinog algoritma. Sa boljom heuristikom, skup otvorenih čvorova će se brže širiti ka ciljnom čvoru i biće posećeno manje čvorova nego primenom Dajkstrinog algoritma. Ključna razlika između dva algoritma je upravo u korišćenju heuristike: Dajkstrin algoritam (kao algoritam neinformisane pretrage) uzima u obzir samo cenu od polaznog do tekućeg čvora — vrednost $g(n)$, a A* (kao algoritam informisane pretrage) koristi vrednost funkcije evaluacije $f(n) = g(n) + h(n)$.

Za $g(n) = 0$, algoritam A* predstavlja specijalnu varijantu pristupa Prvo najbolji, koja najpre obrađuje čvorove sa najboljom heurističkom vrednošću. Ova varijanta algoritma nije nužno optimalna.



Slika 4.7: Stablo pretrage tokom primene algoritma A* na problem iz primera 4.8 (prva varijanta). Levo od čvora zapisana je njegova f vrednost, a desno redni broj u nizu tekućih čvorova, crvenom bojom označeni su čvorovi koji se nalaze u zatvorenoj listi.

4.3.3 Dizajniranje dopustivih heuristika

Svojstva algoritma A* govore da on najbolje performanse (najmanji broj obrađenih čvorova) daje kada je funkcija heuristike bliska idealnoj funkciji heuristike. S druge strane, optimalnost je garantovana samo ako funkcija heuristike nikada ne precenjuje stvarnu cenu puta. Zajedno, to govori da dobra funkcija heuristike mora da bude veoma pažljivo konstruisana — treba da bude što bliža idealnoj funkciji, ali da je nikada ne premašuje.

Ne postoji opšti pristup koji obezbeđuje kvalitetnu heuristiku. Ipak, postoje neke smernice koje mogu pomoći u dizajniranju heuristike. Ukoliko je raspoloživo nekoliko dopustivih heuristika h_1, h_2, ..., h_k, onda je preporučljivo koristiti heuristiku h koja je za svaki čvor n definisana na sledeći način: $h(n) = \max\{h_1(n), h_2(n), \ldots, h_k(n)\}$ i koja je očigledno takođe dopustiva.

Dopustiva heuristika može biti zasnovana i na ceni optimalnog rešenja nekog potproblema problema koji se rešava. U slučaju Lojdove slagalice, to, na primer, može biti problem postavljanja (samo) dva ili tri polja na svoje pozicije. Optimalno rešenje takvog potproblema sigurno ne precenjuje cenu rešavanja čitavog problema, pa cena tog optimalnog rešenja može da posluži kao dopustiva heuristika. Sličan pristup je da se kao heuristika koristi tačna cena neke relaksirane verzije problema. Relaksirana verzija problema je neki novi problem koji se od polaznog dobija zanemarivanjem nekih ograničenja. Primera radi, u slučaju Lojdove slagalice, ako se zanemari ograničenje da na svakom polju može u jednom trenutku biti samo jedan deo slagalice, moguće je tačno izračunati cenu optimalnog rešenja (takve) slagalice, a to je zbir Menhetn rastojanja svih delova do njihovih pozicija u završnom stanju slagalice. Takva cena ne može premašiti cenu slaganja slagalice pri svim ograničenjima, te se može koristiti kao dopustiva heuristika. Ukoliko se ukloni i ograničenje da se delovi slagalice moraju premeštati samo na susedna polja, cena rešavanja takvog problema jednaka je broju delova slagalice koji nisu na svojim mestima. Ovakva ocena, međutim, lošija je od prethodne jer se zanemaruje previše ograničenja izvornog problema i stoga nije dobar izbor za heuristiku. Dodatno, može se kreirati više heuristika u ovom stilu, pa se njihovim kombinovanjem, kao što je već objašnjeno, može dobiti još kvalitetnija heuristika.

Dopustive heuristike ponekad se dobijaju i korišćenjem tehnika mašinskog učenja – rešava se mnoštvo instanci problema i, na osnovu dobijenih informacija, kreira se heuristička funkcija.

4.3.4 Primena algoritma A* na uniformnoj mreži

Opšti algoritam A* često se primenjuje za pronalaženje puta na uniformnoj, kvadratnoj mreži čvorova (koja odgovara, na primer, diskretizovanoj ili rasterizovanoj mapi). Tada on dobija specifičnu formu. Pretpostavimo da je mreža pravilna (sačinjena od kvadrata) i da ima pravougaonu formu. Dodatno, pretpostavljamo da neki čvorovi (tj. neki kvadrati, neka polja mreže) nisu dostupni i da oni predstavljaju *prepreke*. Svako polje povezano je sa svakim susednim poljem osim sa preprekama, te ima najviše četiri susedna polja. Svakom horizontalnom ili vertikalnom kretanju između dva susedna polja obično se pridružuje cena 1. Funkcija heuristike h može se zadati na različite načine. Kada se izračunava vrednost h, obično se, jednostavnosti i efikasnosti radi, ignorišu sve prepreke jer vrednost $h(n)$ je procenjeno a ne stvarno rastojanje, a ignorisanjem prepreka biće (ne nužno, ali moguće) *potcenjeno* stvarno rastojanje. Jedna mogućnost za heuristiku h je euklidsko rastojanje između dva polja:

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Ova funkcija je konzistentna i dopustiva te obezbeđuje optimalnost, ali je zahtevna što se tiče vremena izračunavanja (što može biti kritično za mape sa milionima čvorova). Drugi primer funkcije heuristike je Menhetn rastojanje u kojem se broji ukupan broj polja pređenih horizontalno ili vertikalno da bi se došlo od jednog do drugog polja:

$$d((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|.$$

Ova heuristika je konzistentna (i dopustiva), te garantuje pronalaženje optimalnog puta. Ukoliko su na mreži dozvoljena i dijagonalna kretanja, onda se svakom horizontalnom ili vertikalnom koraku obično pridružuje cena 1, a svakom dijagonalnom potezu cena $\sqrt{2} \approx 1.414$ (ovakva cena odgovara euklidskom rastojanju između središta polja; ove vrednosti često se množe nekom konstantom, na primer 10 i zaokružuju na ceo broj). U ovom slučaju, Menhetn rastojanje može da precenjuje rastojanje do ciljnog čvora, te nije dopustiva heuristika i ne garantuje pronalaženje najkraćeg puta. No, ovo rastojanje u praksi često daje dobre rezultate i pronađeni putevi su obično dovoljno dobri, čak i ako nisu najkraći. U slučaju da su dozvoljena i dijagonalna kretanja, kao heuristika koja je konzistentna (i dopustiva) može se koristiti Čebiševljevo rastojanje:

$$d((x_1, y_1), (x_2, y_2)) = max(|x_2 - x_1|, |y_2 - y_1|).$$

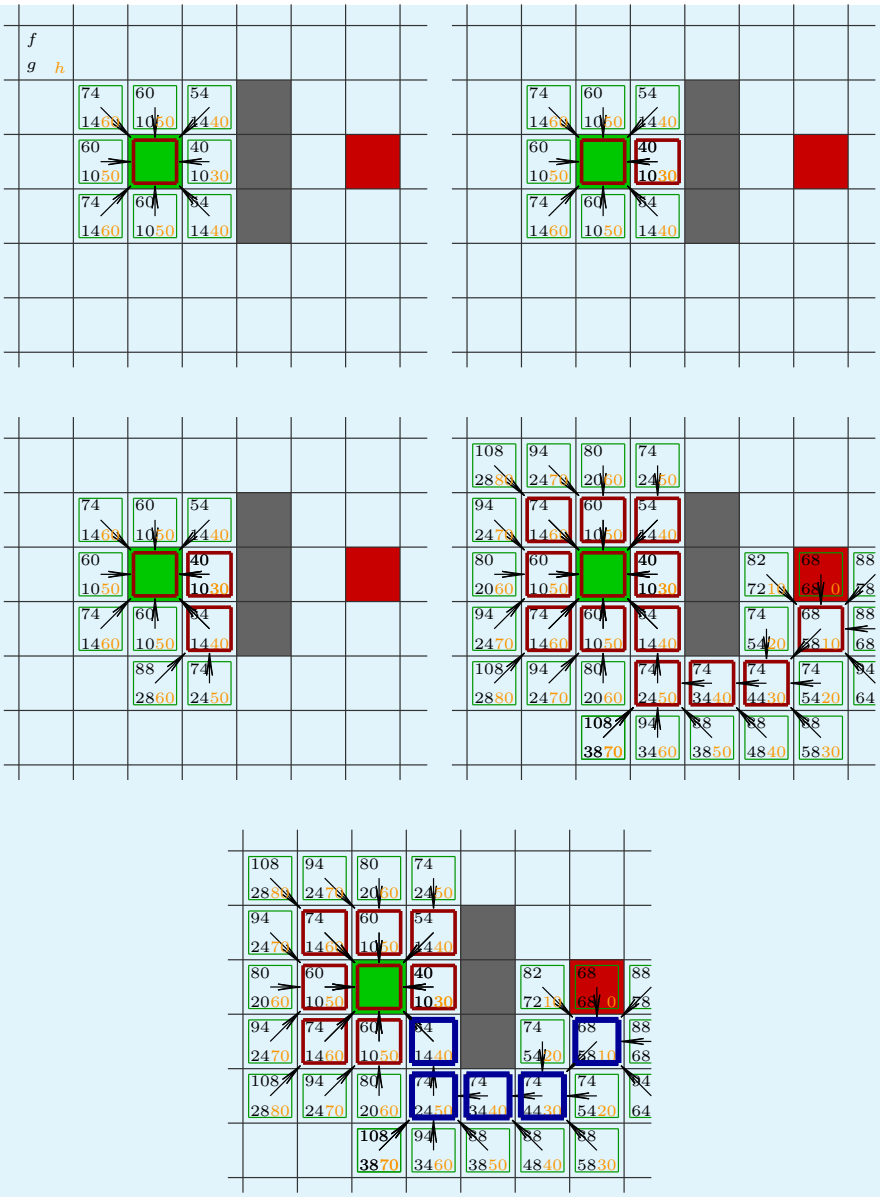
I kada heuristika nije konzistentna, mogu da se ne ažuriraju (i otvaraju ponovo) zatvoreni čvorovi. I ovakav pristup često daje dovoljno dobra i efikasna rešenja, iako ne nužno optimalna.

Primer 4.9. *Primenom algoritma A* potrebno je pronaći put od polaznog do ciljnog čvora na uniformnoj mreži prikazanoj na narednoj slici. Polazni čvor označen je zelenom, a ciljni crvenom bojom. Dozvoljeni su horizontalni, vertikalni i dijagonalni potezi. Nije moguće dijagonalno kretanje ka polju gore-desno ako je desno polje prepreka, ni u drugim analognim situacijama. Ovakvo ograničenje zavisi od prirode konkretne primene (na primer, ovakvo ograničenje može da opisuje moguće kretanje vozila).*

Vrednosti funkcija f, g i h zapisane su u poljima uniformne mreže: vrednost funkcije f zapisana je gore-levo, vrednost funkcije g dole-levo, a vrednost funkcije h dole-desno. Vrednost funkcije f za svako polje je, kao i uvek, zbir vrednosti funkcija g i h. Za rastojanje između dva polja susedna horizontalno ili vertikalno uzima se vrednost 10*, a za rastojanje između dva polja susedna dijagonalno uzima se vrednost* 14*. Kao heuristika h koristi se Menhetn rastojanje do ciljnog polja pomnoženo sa* 10 *(prepreka se zanemaruje). Ova heuristika nije dopustiva jer su dozvoljeni i dijagonalni potezi.*

Otvorena polja označena su dodatnim tankim (zelenim) kvadratima u okviru polja, zatvorena polja označena su dodatnim unutrašnjim (crvenim) kvadratima srednje debljine, a polja koja čine pronađeni put označena su dodatnim unutrašnjim debljim (plavim) kvadratima. Strelice ukazuju na tekućeg roditelja polja.

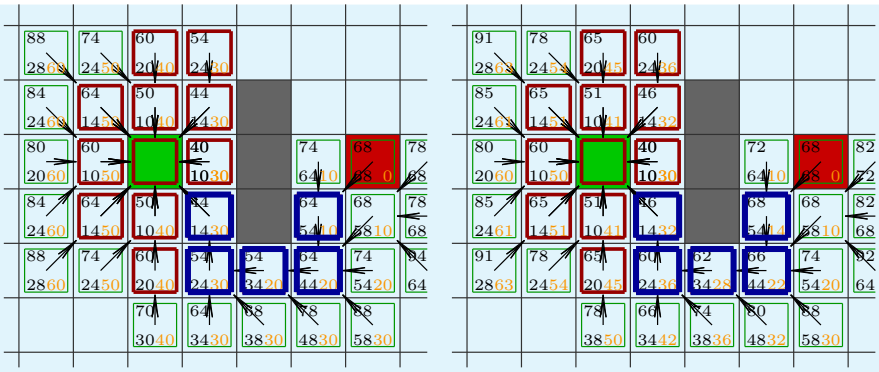
*Pretraga kreće od polaznog polja jer je na početku samo ono u otvorenoj listi. Polazno polje briše se iz otvorene liste i dodaje u zatvorenu listu. U otvorenoj listi je onda njegovih osam susednih polja. Od svih njih, bira se ono sa najmanjom vrednošću funkcije f (*40*), to je polje neposredno desno od polaznog polja i ono će biti sledeće tekuće polje. Za to, novo tekuće polje proveravaju se njegova susedna polja (nakon čega će to tekuće polje biti izbačeno iz otvorene liste i dodato u zatvorenu listu). Njegova četiri susedna polja već su u otvorenoj listi a jedno je u zatvorenoj listi, pa je potrebno proveriti da li put preko tekućeg čvora popravlja njihove trenutne ocene. Razmotrimo, na primer, polje neposredno iznad tekućeg polja: vrednost funkcije g za njega je* 14*. Ukoliko bi se do njega dolazilo preko tekućeg polja, vrednost funkcije g bila bi* 20 *(*10 *je cena od polaznog do tekućeg čvora i* 10 *je cena prelaska od tekućeg polja). Dakle, na ovaj način se ne može popraviti vrednost funkcije g u polju iznad i ona ostaje nepromenjena. Pretraga se i u nastavku sprovodi kao kod opšteg algoritma. Kada ciljno polje postane tekuće, traženi put od polaznog čvora konstruiše se jednostavno: kreće se od ciljnog čvora i prelazi na roditeljski sve dok se ne dođe do polaznog čvora. Ovako određen niz polja u suprotnom poretku daje traženi put.*



Primetimo kako vrednost funkcije f za neko polje može da se promeni tokom primene algoritma. Nakon nekoliko iteracija, vrednosti funkcija g i f, kao i roditeljsko polje, promenili su se za polje koje se nalazi dva polja ispod polaznog polja. Ranije je ovo polje imalo vrednost funkcije g jednaku 28 *(i vrednost funkcije f jednaku* 88*) i roditeljsko polje je bilo gore-desno. Kasnije, ovo isto polje ima vrednost funkcije g jednaku* 20 *(i vrednost funkcije f jednaku* 80*), a roditeljsko polje je gore. Ova izmena dogodila se u nekoj iteraciji u međuvremenu.*

Ponovimo da Menhetn rastojanje koje je korišćeno ne daje dopustivu heuristiku. Na primer, za drugo polje ispod prepreke, ocena heuristike je 50 *i ona je veća od stvarnog rastojanja do ciljnog čvora koje je jednako* 38*. Kako su dozvoljeni dijagonalni potezi, a za heuristiku je korišćeno Menhetn rastojanje, ne može se garantovati da će uvek biti dobijen optimalni put. Optimalnost puta bila bi garantovana da je korišćeno Čebiševljevo rastojanje. Uprkos ovome, i u ovakvim situacijama može da se koristi Menhetn rastojanje jer može* brže *da vodi cilju (iako možda ne dajući optimalan put).*

Rad algoritma A na istom grafu, ali za heuristike zasnovane na Čebiševljevom rastojanju i na euklidskom rastojanju ilustrovan je narednoj slici, levo i desno redom:*



U ova dva slučaja, kako su ove heuristike konzistentne, duž pronađenog puta niz vrednosti f od početnog ka ciljnog čvoru je neopadajući niz. To ne važi kada se za heuristiku koristi rastojanje Menhetn (jer ona nije konzistentna kada su dozvoljeni i dijagonalni potezi).

4.3.5 Implementaciona pitanja

Algoritam A* obično se primenjuje u aplikacijama koje rade u realnom vremenu, te je neophodno da je efikasno implementiran. Otvorena lista često se implementira kao binarni min-hip (kako bi se brzo dolazilo do elementa sa najmanjom vrednošću funkcije f), a zatvorena lista kao heš tabela. Korišćenjem ovih struktura, operacije za dodavanje i brisanje elemenata iz otvorene liste zahtevaju, u najgorem slučaju, vreme $O(\log |V|)$, gde je V skup čvorova grafa, a operacije proveravanja da li je element u zatvorenoj listi, dodavanja u zatvorenu listu, kao i brisanja iz zatvorene liste zahtevaju prosečno vreme $O(1)$.

Zahtevi za memorijskim prostorom su za algoritam A* često još veći problem nego vremenska složenost. Ipak, ukoliko broj čvorova grafa nije preveliki, može da bude isplativo i statičko alociranje potrebnog prostora (ili dinamičko alociranje većih blokova) koji onda može da se koristi u savezu sa min-hip strukturom, kako bi se izbegle česte i skupe operacije dinamičkog alociranja (i dealociranja) za pojedinačne čvorove.

Najgori slučaj za algoritam A* je kada ne postoji put između polaznog i ciljnog čvora. U tu svrhu može se, za neusmerene grafove, implementirati brza provera da li uopšte postoji put između dva čvora: dva čvora su povezana ako i samo ako pripadaju povezanim delovima grafa. Ako se za svaki čvor može lako proveriti kom delu grafa pripada, onda je i navedena provera jednostavna (posebno ako mapa može biti obrađena unapred i podeljena na povezane delove).

Kada se algoritam A* koristi za pronalaženje puta na uniformnoj mreži, dobijeni putevi ne izgledaju uvek prirodno (posebno ako treba da ih prati neko vozilo). Takve puteve potrebno je zameniti sličnim putevima koji su glatkiji i izgledaju prirodnije.

4.3.6 Primer rešavanja problema korišćenjem algoritma A*

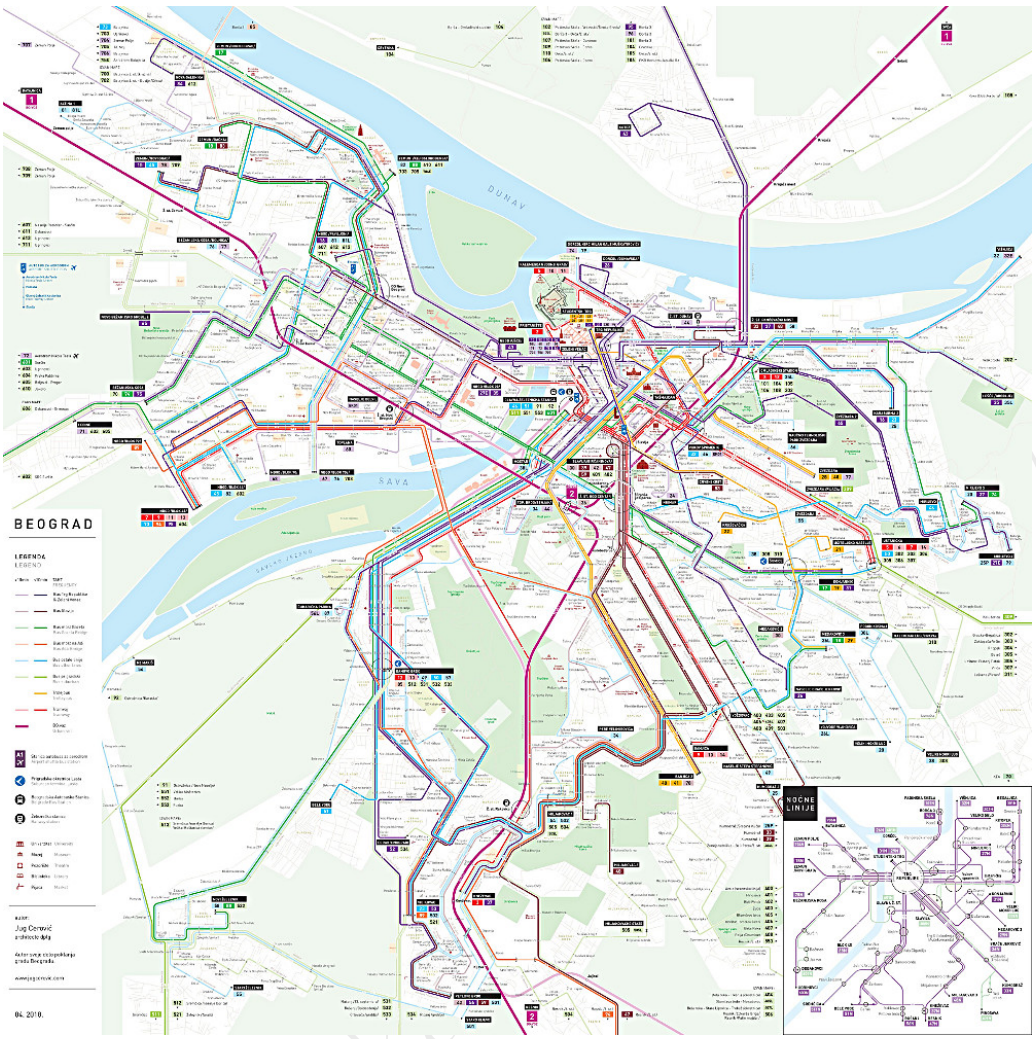
Razmotrimo sledeći realan problem: potrebno je za neki grad napraviti aplikaciju koja predlaže najbolji put od jednog do drugog stajališta javnog prevoza (Beograd, na primer, ima oko 160 linija javnog prevoza i oko 2400 stajališta, slika 4.8). Cena puta može biti njegova dužina ili (na primer, prosečno) vreme potrebno za njegovo prelaženje (u ovom drugom slučaju, bolji rezultati će se dobijati ako su raspoložive informacije za različita doba dana, za svaki dan u nedelji i slično).

Opisani zadatak može se efikasno rešiti primenom algoritma A*, sa skupom stajališta koji čini skup čvorova grafa. Ukoliko je cena puta definisana kao njegova dužina, pogodna heuristika može biti zasnovana na euklidskom rastojanju i ona je očigledno dopustiva. Ukoliko je cena puta definisana kao potrebno vreme, onda kao dopustiva heuristika može da se koristi euklidsko rastojanje podeljeno maksimalnom brzinom vozila.

U najjednostavnijoj varijanti, skup čvorova (usmerenog) grafa jednak je skupu stajališta. Grana između dva čvora postoji ako i samo ako neka linija povezuje ta dva stajališta direktno (bez stajališta između). To stvara sledeći problem: algoritam ne može da razlikuje kvalitet putanje sa presedanjima i bez presedanja.

Alternativa je da svako stajalište daje onoliko čvorova grafa koliko linija staje na tom stajalištu. Na primer, ako na stajalištu 10 staju linije 22 i 29, postojaće čvorovi 10-22 i 10-29. Granama će biti spojena sva susedna stajališta neke linije. Dodatno, granama će biti spojeni čvorovi koji odgovaraju istom stajalištu a različitim linijima. Cena ovakvih grana može da se bira slobodno i ona kontroliše koliko su presedanja prihvatljiva. Kao i cene puteva u grafu, i funkcije f, g i h će odgovarati ili dužini puta ili utrošenom vremenu.

U Beogradu, na jednom stajalištu staje prosečno po 5 različitih linija, pa bi ukupan broj čvorova grafa bio oko 12000. Za ovakav graf, kvalitetna implementacija trošiće za jedan upit samo delić sekunde.



Slika 4.8: Mapa linija javnog prevoza Beograda.

Opisani pristup ne razmatra (realnu) opciju da putnik pešači između dva (relativno bliska) stajališta. Da bi i to bilo omogućeno, grafu treba dodati i „pešačke čvorove“ za svako stajalište, grane koje povezuju svaki postojeći čvor stajališta sa tim novim čvorom, kao i grane koje povezuju „pešačke čvorove“ bliskih stajališta (na primer, na rastojanju manjem od 300m). Cena takvih grana može biti grubo procenjena na bazi euklidskog rastojanja. Ukoliko je cena puta bazirana na vremenu, onda se za cenu ovakve grane može uzeti euklidsko rastojanje podeljeno prosečnom brzinom hodanja. Moguće su i mnoge druge modifikacije osnovnog rešenja koje mogu dati kvalitetnije i upotrebljivije rezultate.

Glava 5

Igranje strateških igara

Automatsko igranje strateških igara kao što je šah davnašnji je izazov. Još početkom dvadesetog veka španski pronalazač Tores Kevedo (Torres y Quevedo) konstruisao je (i prikazao na svetskoj izložbi u Parizu 1914. godine) elektro-mehanički uređaj *El Ajedrecista* („Šahista“) koji je, kao beli, igrao šahovsku završnicu „kralj i top protiv kralja“ i iz svake pozicije nepogrešivo pobeđivao (iako ne u najmanjem mogućem broju poteza). Razvoj teorije igara započeo je Džon fon Nojman (John von Neumann) postavljanjem opšteg problema (1928. godine): *Igrači* $S_1, S_2, \ldots, S_n$ *igraju datu igru* Γ. *Kako treba da igra igrač* S_m *da bi ostvario najbolji mogući rezultat?* Već od polovine dvadesetog veka, problemi ove vrste bili su važan i često pokretački, motivišući izazov za oblast u nastajanju – veštačku inteligenciju. Neki od najvećih (ili makar najšire poznatih) uspeha veštačke inteligencije ostvareni su upravo na polju strateških igara: računari su već odavno pobedili svetske šampione u igrama bekgemon, dame i šah (tada važeći svetski šampion Gari Kasparov partiju šaha izgubio je od računara 1997. godine), a 2016. godine i u igri go. Iako su ovi programi veoma uspešni, njihovi principi odlučivanja kvalitativno su (po pitanjima apstrahovanja, analogija, pravljenja planova i sl.) veoma različiti od ljudskih. Većina najznačajnijih pristupa za igranje strateških igara zasnovana je na namenskim algoritmima pretrage, a odnedavno i na kombinacijama takvih algoritama sa tehnikama mašinskog učenja (videti poglavlje 13.4.1). Pretraga koja se koristi u igrama je specifična vrsta pretrage – *suparnička pretraga* (eng. *adversarial search*), u kojoj postoji i „suparnička strana“ koja takođe menja stanje pretrage, obično suprotno željama strane koja vrši pretragu (primeri u kojima je primenljiva ovakva pretraga su, pored igara, modeli ratovanja, trgovanja i slično).

U nastavku neće biti upuštanja u analize pojedinačnih igara, već će biti opisani opšti pojmovi i algoritmi koji mogu da se koriste za širok spektar strateških igara. Biće razmatrani algoritmi za takozvane igre nulte sume bez nepoznatih informacija za dva igrača. To su igre kod kojih igrači, grubo rečeno, imaju analogne, simetrične mogućnosti – ono što je dobro za jednog igrača loše je za drugog i svaki igrač zna koje poteze na raspolaganju ima protivnik. U ovu kategoriju spadaju, na primer, igre šah, dame, go, reversi, iks-oks, četiri u nizu, mankala, a ne spadaju, na primer, igre u kojima igrač ne zna karte koje ima protivnik, nepoznati broj koji treba pogoditi itd.

5.1 Opšte strategije za igranje igara

Moderna istorija programiranja strateških igara počinje člankom *Programming a digital computer for playing Chess* Kloda Šenona iz 1950. godine. U tom tekstu, Šenon je opisao dve opšte strategije za izbor poteza:

A: „Minimaks“ procedurom vrši se pretraživanje stabla igre sa određenom funkcijom evaluacije i ocenjivanje legalnih poteza; bira se potez sa najboljom ocenom (videti poglavlje 5.4.2).

B: Potez se bira na osnovu trenutne pozicije/situacije u igri i na osnovu odgovarajuće, unapred pripremljene tabele.

Pristup zasnovan na Šenonovoj strategiji **A** naziva se „sistematskim“ ili „dubinskim pretraživanjem“ a i pristupom „gruba sila plus jednostavna vrednosna funkcija“. Ako bi se pretraživanje stabla igre vršilo do završnih stanja, efektivno bi bili ispitivani svi mogući tokovi nastavka partije i mogao bi da bude izabran zaista najbolji potez (završna stanja bila bi ocenjivana jednostavnom „trovrednosnom funkcijom“: mogući ishodi su pobeda prvog igrača, pobega drugog igrača i nerešeno). Međutim, taj pristup za netrivijalne igre nije praktično ostvariv. Zbog toga, efikasna primena Šenonove strategije **A** svodi se na pretraživanje stabla igre do relativno male dubine algoritmima koji su usmereni heuristikama i uz dobro osmišljenu, ali jednostavnu funkciju evaluacije za ocenu nezavršnih pozicija. Ovakvim pristupom ne dobija se nužno zaista najbolji potez, a obim pretrage ostaje,

najčešće, i dalje veoma veliki. Precizniji opisi pojmova stabla igre, funkcije evaluacije i algoritama minimaks tipa dati su u poglavlju 5.4.

Šenonova strategija **B** zasniva se na jednostavnoj, unapred pripremljenoj tabeli koja zamenjuje izračunavanje u toku izvršavanja. U ovom pristupu, znanje o igri ne nalazi se u programu koji igra, već u programu koji je tabelu generisao. Tabela ima dve kolone: u jednoj su moguće pozicije/stanja igre, a u drugoj preporučeni (ponekad optimalni) potezi. Jedan od „klasičnih" primera ovog pristupa je program za igranje šahovske završnice *kralj i kraljica protiv kralja i topa* koji je 1977. godine kreirao Kenet Tompson (Kenneth Thompson, tvorac operativnog sistema UNIX). Tabela koju je koristio program sadržavala je sve moguće pozicije i optimalne poteze za sve te pozicije (pri čemu se pod optimalnim potezom za igrača koji ima kralja i kraljicu smatra potez koji vodi pobedi u najmanjem broju poteza, a za slabijeg – potez koji maksimalno odlaže poraz). Tabela je imala oko tri miliona vrsta i program koji se na njoj zasnivao bio je, naravno, nepogrešiv. Tabela je kreirana korišćenjem *retrogradne analize* koja se za igrača koji ima kraljicu (pretpostavimo da je to beli) sprovodi na sledeći način. Obrađuju se samo pozicije u kojima je na potezu beli i svakoj se, inicijalno, pridružuje vrednost ∞; dalje, prepoznaje se svaka pozicija u kojoj crni može neposredno biti matiran (mat u jednom potezu belog), pridružuje joj se taj matni potez i vrednost 1; dalje, prepoznaje se svaka pozicija i u tabelu se upisuje potez nakon kojeg, ma šta da odigra crni dolazi se u poziciju označenu brojem 1 (za takve pozicije važi da postoji mat u dva poteza belog) i toj poziciji pridružuje se vrednost 2; dalje, prepoznaje se svaka pozicija i u tabelu se upisuje potez nakon kojeg, ma šta da odigra crni dolazi se u poziciju označenu brojem ≤ 2 (za takve pozicije važi da postoji mat u najviše tri poteza belog) i toj poziciji pridružuje se vrednost 3; ... dalje, prepoznaje se svaka pozicija i u tabelu se upisuje potez nakon kojeg, ma šta da odigra crni dolazi se u poziciju označenu brojem $\leq d$ (za takve pozicije važi da postoji mat u najviše $d+1$ poteza belog) i toj poziciji pridružuje se vrednost $d+1$; ... Ovaj postupak nastavlja se dok god postoji neka pozicija označena sa ∞. Tabela za optimalnu igru crnog (koji ima topa) kreira se korišćenjem tabele za belog: neka u nekoj poziciji p crni ima na raspolaganju n poteza i neka u tabeli za belog pozicijama u koje vode ovi potezi odgovaraju vrednosti v_1, v_2, ..., v_n (to su maksimalni brojevi poteza do mata ako beli igra optimalno). U tabelu za crnog, za poziciju p kao optimalan potez upisuje se potez koji vodi ka najvećoj vrednosti (čime crni maksimalno odlaže neminovni poraz). Korišćenjem istog pristupa, na moskovskom univerzitetu su 2012. godine kreirane *tabele Lomonosov* optimalnih poteza za sve šahovske završnice sa najviše sedam figura na tabli. Tabela sadrži više od 500 triliona pozicija (pri čemu se u tabeli ne čuvaju pozicije koje se mogu dobiti od drugih pozicija simetrijama i rotacijama).

Šenonova strategija **A** za izbor poteza zahteva malo memorije i mnogo izračunavanja, a strategija **B** malo izračunavanja i mnogo memorije. Na toj skali odnosa količine podataka i obima izračunavanja, čovekov način zaključivanja je između ovih krajnosti i bitno se od njih razlikuje po svojoj prirodi.

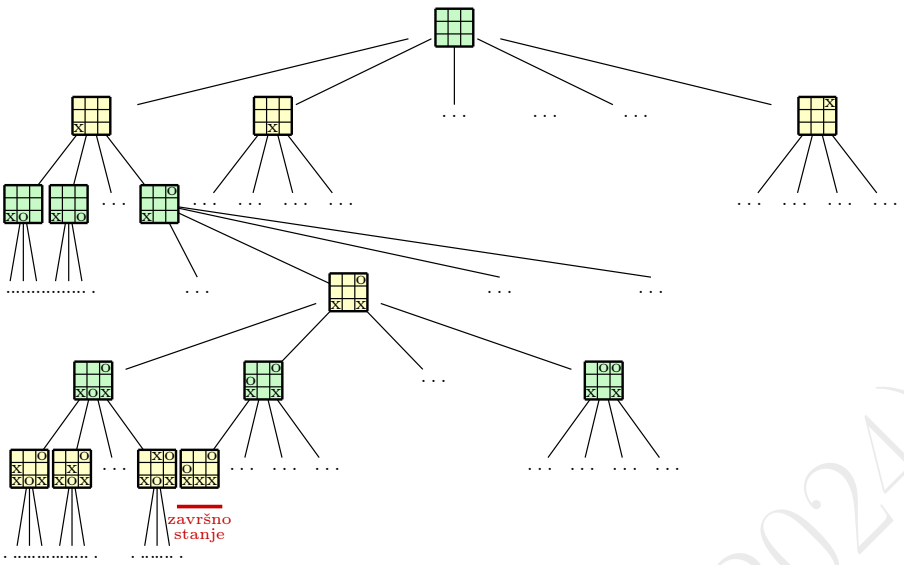
U dvadeset prvom veku, na popularnosti dobija i treća opšta strategija za igranje strateških igara zasnovana na Monte Karlo pretrazi i velikom broju simulacija kojima se, u sadejstvu sa mašinskim učenjem, ocenjuju mogući potezi.

5.2 Legalni potezi i stablo igre

Pravila konkretne igre definišu *legalna stanja* (tj. legalne pozicije) i *legalne poteze* za svaku legalnu poziciju.[1] Za svaku legalnu poziciju može se efektivno odrediti skup legalnih poteza. Neke legalne pozicije mogu biti *početne pozicije*, a neke *završne*. U nekim igrama, legalni potez može biti i *dalje*, u situaciji kada igrač koji je na redu nema na raspolaganju legalnih poteza i preskače svoj red (takve situacije ne postoje u šahu, ali postoje, na primer, u igri reversi).

Prostor stanja igre može se opisati grafom čiji su čvorovi legalne pozicije, a grane legalni potezi. Taj graf, graf prostora stanja, je usmeren jer nije nužno da postoje potezi u oba smera koji povezuju dva stanja (na primer, u igri šah, pešak može da se kreće samo napred, ne i nazad). *Stablo igre* je stablo u čijim su čvorovima legalne pozicije i za svaki čvor njegova deca su sve pozicije do kojih se iz tog čvora može doći po jednim legalnim potezom. Od korena do bilo kog lista naizmenično se, dakle, smenjuju grane koje odgovaraju potezima prvog i drugog igrača (nije nužno da je u korenu stabla početna pozicija niti da su u svim listovima završne pozicije). *Potpuno stablo igre* je stablo igre u čijem je korenu početna pozicija igre, a svi listovi su završne pozicije igre i svakom listu pridružen je ishod – pobeda prvog igrača, nerešeno ili pobeda drugog igrača. Potpuno stablo igre ima onoliko listova koliko data igra ima različitih mogućih tokova. Taj broj je kod većine igara (čak i kod veoma jednostavnih) ogroman i onemogućava kompletno pretraživanje u cilju izbora poteza. Na slici 5.1 ilustrovan je deo potpunog stabla igre za igru iks-oks.

[1] Nekad se ono što odigra jedan igrač naziva *polupotez* (eng. *ply*), a par takvih uzastopnih polupoteza naziva se *potez* (eng. *move*). U ovom tekstu će se smatrati da je potez ono što odigra jedan igrač.



Slika 5.1: Deo potpunog stabla igre za igru iks-oks.

5.3 Otvaranje

U strateškim igrama, umesto da se na samom početku igre program upusti u proces pretrage, obično se koriste *knjige otvaranja* (eng. *opening book*) koje su zasnovane na ljudskom iskustvu i koje sadrže informacije o poznatim i kvalitetnim potezima koji se često javljaju u otvaranju.[2] Knjiga otvaranja koristi se, ne zahtevajući mnogo vremena za izvršavanje, dok god je to moguće (u šahu, na primer, obično za prvih desetak poteza) sa očekivanjem da se dođe do bolje pozicije nego da je program koristio druga znanja. Kada više nije moguće koristiti knjigu otvaranja, prelazi se na druge strategije izbora poteza.

Ukoliko protivnik odigra neki neuobičajen potez koji ne postoji u knjizi otvaranja, program je primoran da promeni pristup i da počne da vrši pretragu nastavka stabla igre. Takav neuobičajeni potez može, međutim, da bude rizičniji za igrača koji ga je odigrao nego za program koji u daljem toku može da kompenzuje rano napuštanje knjige otvaranja i iskoristi slabosti protivnika.

Ukoliko za neku poziciju postoji u knjizi otvaranja više mogućih poteza, izbor može da se načini po verovatnoćama koje oslikavaju kvalitet tih poteza. Pomenuti pristup može da se realizuje, na primer, na sledeći način: neka je, na osnovu knjige otvaranja, u nekom trenutku na raspolaganju n poteza. Svakom od njih neka je pridružena neka empirijska nenegativna ocena kvaliteta m_i, $1 \leq i \leq n$ (na primer, na osnovu ishoda partija u knjizi koje uključuju taj potez). Što je neki potez bolji, to je njegova ocena veća. Tada se, u toj poziciji, i-ti potez ($1 \leq i \leq n$) bira sa verovatnoćom

$$p_i = \frac{m_i}{\sum_{j=1}^{n} m_j}.$$

Na taj način izbegava se determinističko ponašanje programa u otvaranju: bolji potezi (u smislu neke procene) biraju se češće, ali ne uvek. Ocene m_i mogu se tokom vremena i korigovati.

Knjiga otvaranja može biti statička (sadržati određen, konačan broj varijanti u svakom potezu i informacije o potezima samo do određene dubine) ili se modifikovati tokom samog izvršavanja programa.

5.4 Središnjica

Savremeni programi za strateške igre u središnjici najčešće koriste Šenonovu strategiju **A** — koriste pretraživanje stabla igre do neke dubine i pogodnu funkciju evaluacije. Funkcija evaluacije izračunava se samo za čvorove na nekoj izabranoj dubini, a čvorovima na manjim dubinama ocena se određuje na osnovu ocena njihovih potomaka. Očekuje se da su potezi dobijeni na ovaj način dovoljno dobri (ali ne nužno i zaista najbolji mogući, sem ako se ne vrši pretraživanje potpunog stabla igre). Pored kvalitetne funkcije evaluacije, od ključne važnosti su algoritmi koji se koriste za pretraživanje stabla igre vođeni raznovrsnim heuristikama. Pretraživanjem stabla igre nastaje *stablo pretrage*. Za konkretnu poziciju, stablo pretrage obično čini samo mali deo potpunog stabla igre, tj. algoritam posećuje samo mali deo skupa pozicija u stablu igre. Algoritam je efikasniji što je taj deo

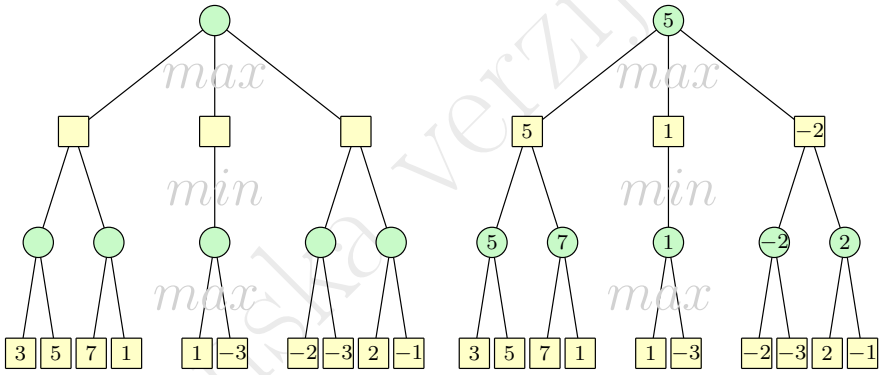
[2] U evropskoj istoriji šaha, otvaranja se sistematično proučavaju već više od četiristo godina.

manji. Kao i u drugim vrstama pretrage, u procesu traženja poteza, stablo pretrage ne kreira se eksplicitno, kao struktura u memoriji računara, već samo implicitno, kroz proces izvršavanja algoritma.

5.4.1 Funkcija evaluacije i statička ocena pozicije

Funkcija evaluacije (eng. *evaluation function*) dodeljuje pozicijama, tj. čvorovima stabla igre, neke ocene na osnovu kojih one mogu da se porede po kvalitetu. Ocena pozicije formira se u skladu sa specifičnostima konkretne igre, pri čemu se ne ispituju ni pozicije iz kojih se došlo u tu poziciju niti mogući nastavci, te se zato za ovu ocenu kaže da je *statička ocena pozicije*. Gotovo sve znanje o igri koje se koristi u središnjici partije sadržano je u funkciji evaluacije i u velikoj meri od nje zavisi kvalitet igre programa. Potrebno je da ona koristi što više relevantnih informacija ali, s druge strane, kako se izračunava mnogo puta, potrebno je da se izvršava što brže. Funkcija evaluacije obično preslikava skup svih mogućih pozicija u simetričan segment $[-M, M]$. Tada se vrednost M dodeljuje samo završnim stanjima igre u kojima je pobednik prvi igrač, a $-M$ samo završnim stanjima igre u kojima je pobednik drugi igrač. U igrama nulte sume, smisao funkcije evaluacije za protivnike u igri za dva igrača je suprotan — ono što je najbolje stanje za jednog igrača najlošije je za drugog i obratno. Dakle, funkcija evaluacije za simetrične pozicije (za zamenjene uloge igrača) treba da daje vrednosti koje se razlikuju samo po znaku. Najjednostavnija je trovrednosna funkcija evaluacije: ona se primenjuje samo na završne pozicije igre i ima samo tri različite vrednosti – za pobedu prvog, za pobedu drugog igrača i za nerešen ishod (na primer, 10, -10 i 0). Trovrednosna funkcija zahteva pretraživanje stabla igre do završnih čvorova, pa je, zbog velike dubine pretraživanja, ova funkcija za većinu igara praktično neupotrebljiva.

Funkcija evaluacije u šahu obično uključuje vrednost „materijala“ (beloj kraljici, može da bude pridružena vrednost 100, belom topu 50, belom lovcu i konju 30, a belom pešaku 10; crnim figurama onda se pridružuju iste vrednosti, ali suprotnog znaka), pokretljivost figura, pešačku strukturu, rokade i slično.



Slika 5.2: Ilustracija primene algoritma Minimaks.

5.4.2 Algoritam Minimaks

Algoritam Minimaks ključni je element Šenonove strategije **A** i on je u osnovi skoro svih algoritama za izbor poteza sistematičnim pretraživanjem stabla igre. Ovaj algoritam prvi je opisao Fon Nojman još 1928. godine.

Algoritam Minimaks pretraživanjem stabla igre za igrača koji je na potezu određuje najbolji mogući potez u datoj poziciji. Naglasimo da je obično ekstremno teško ili nemoguće znati šta je *zaista* najbolji potez u nekoj poziciji. Zato se u ovom kontekstu (u kontekstu primene algoritma Minimaks) pod „najboljim potezom“ podrazumeva najbolji potez za zadati čvor, za zadatu maksimalnu dubinu pretraživanja i za izabranu funkciju evaluacije. Pretpostavimo da funkcija evaluacije za igrača koji je na potezu ima pozitivan smisao, tj. bolji je potez ako obezbeđuje veću vrednost funkcije. Ocene se funkcijom evaluacije dodeljuju čvorovima na zadatoj, fiksnoj dubini (ti čvorovi ne moraju da predstavljaju završna stanja igre) ili na manjoj, samo ako je tada dostignuto završno stanje igre. Dalji postupak je rekurzivan: kao ocena čvoru dodeljuje se minimum ocena njegove dece ako je u tom čvoru na potezu protivnik, a maksimum ocena njegove dece, u suprotnom (slika 5.2). Korišćenje minimuma ocena kada je na potezu protivnik oslikava pretpostavku da i protivnik igra koristeći istu logiku (tj. igra koristeći isti stil pretrage i funkciju evaluacije). Ocena početnog čvora je maksimum ocena njegove dece i bira se potez kojem odgovara taj maksimum. Dakle, algoritam karakteriše minimizovanje ocene kada je na potezu protivnik i maksimizovanje ocene kada je na potezu sâm igrač, pa otuda i ime algoritma. Postupak naizmeničnog minimizovanja i maksimizovanja zovemo i *minimaksimizacija*. Algoritam Minimaks dat je na slici 5.3. Primetimo da glavna funkcija algoritma ne vraća ocenu čvora, već najbolji pronađeni potez. To znači da

Algoritam: Minimaks

Ulaz: Funkcija evaluacije f, pozicija, dozvoljena dubina pretraživanja d

Izlaz: Potez

1: $r := Max(f, \text{pozicija}, d)$;
2: vrati potez kojem odgovara vrednost r.

Algoritam: Max

Ulaz: Funkcija evaluacije f, pozicija, dozvoljena dubina pretraživanja d

Izlaz: Vrednost pozicije

1: **ako** je pozicija završna ili je d jednako 0 **onda**
2: vrati $f(\text{pozicija})$;
3: $a := -\infty$;
4: **za** svaku poziciju s do koje se može doći u jednom potezu **radi**
5: $m := Min(f, s, d-1)$;
6: **ako** $m > a$ **onda**
7: $a := m$;
8: vrati a.

Algoritam: Min

Ulaz: Funkcija evaluacije f, pozicija, dozvoljena dubina pretraživanja d

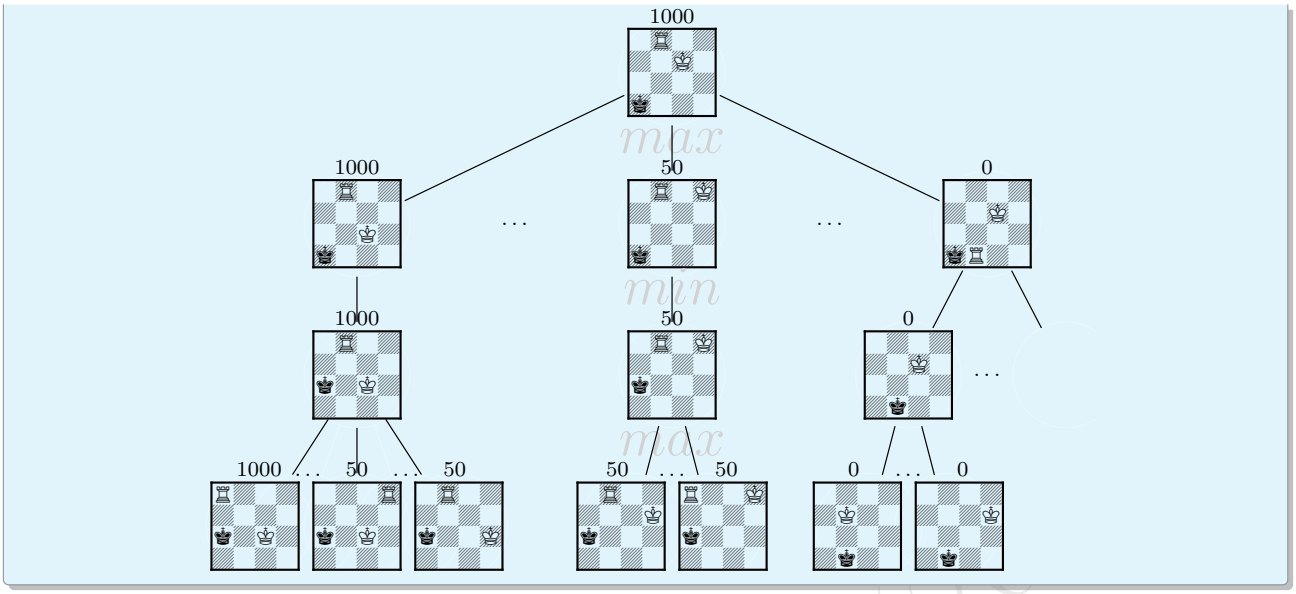
Izlaz: Vrednost pozicije

1: **ako** je pozicija završna ili je d jednako 0 **onda**
2: vrati $f(\text{pozicija})$;
3: $b := +\infty$;
4: **za** svaku poziciju s do koje se može doći u jednom potezu **radi**
5: $m := Max(f, s, d-1)$;
6: **ako** $m < b$ **onda**
7: $b := m$;
8: vrati b.

Slika 5.3: Algoritam Minimaks.

je potrebno osigurati da funkcija Max na najvišem nivou takođe vraća potez koji vodi najboljoj oceni, ali to svojstvo smatramo implementacionim detaljem te ono nije pokriveno prikazanim algoritmom.

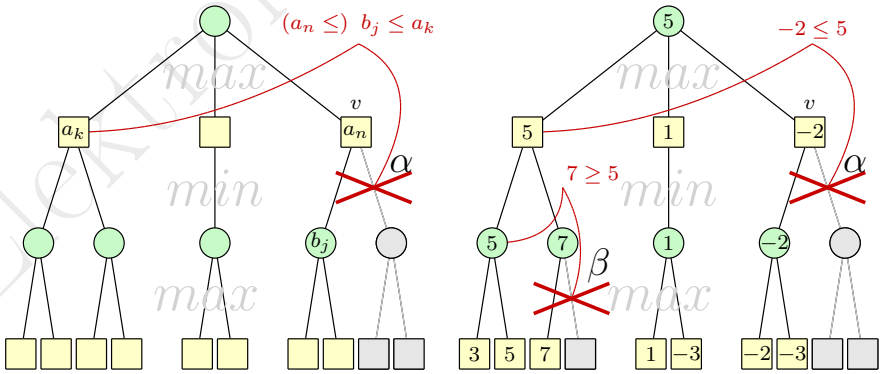
Primer 5.1. *Na narednoj slici prikazan je primer primene algoritma Minimaks na pojednostavljenu verziju šaha na tabli 4×4. Koristi se funkcija evaluacije koja topu daje vrednost 50, a pobedi vrednost 1000. Kao najbolji potez za koreni čvor bira se prvi potez naveden u sledećem redu – potez koji vodi u mat u dva poteza.*



Algoritam Minimaks (kao i ostali algoritmi zasnovani na minimaksimizaciji) vrši izbor poteza samo na osnovu vrednosti koje su pridružene čvorovima na maksimalnoj dubini pretraživanja. To znači da se ne ispituju potezi koji dalje slede (a to ispitivanje često bi promenilo odluku o izabranom potezu). Dakle, pri pretraživanju stabla igre razmatraju se („vide se“) čvorovi do neke fiksne dubine, ali ne i oni posle njih. Za ovakvo ponašanje često se kaže da ima *efekat horizonta* (eng. *horizon effect*). Dodatno, kada je neki potez izabran na osnovu dubljih čvorova i odigran, informacija o tome ne koristi se ubuduće u procesu izbora narednog poteza (na primer, ako je neki potez izabran na osnovu poteza p koji obećava i koji je na dubini tri, u sledećem potezu istog igrača pretraživanje kreće iznova i često neće biti izabran potez koji vodi potezu p).

5.4.3 Algoritam Alfa-beta

Algoritam Alfa-beta (ili $\alpha-\beta$) otkriven je sredinom dvadesetog veka nezavisno od strane nekoliko istraživača. Semjuel (Arthur Samuel) sa jedne i Edvards (Daniel Edwards), Hart (Timothy Hart) i Levin (Michael Levin) sa druge strane, formulisali su nezavisno ranu verziju algoritma sredinom pedesetih godina. Makarti je slične ideje predstavio 1956. godine, tokom znamenite konferencije u Darmutu. Brudno (Alexander Brudno) je, takođe nezavisno, otkrio algoritam Alfa-beta i objavio ga 1963. godine.



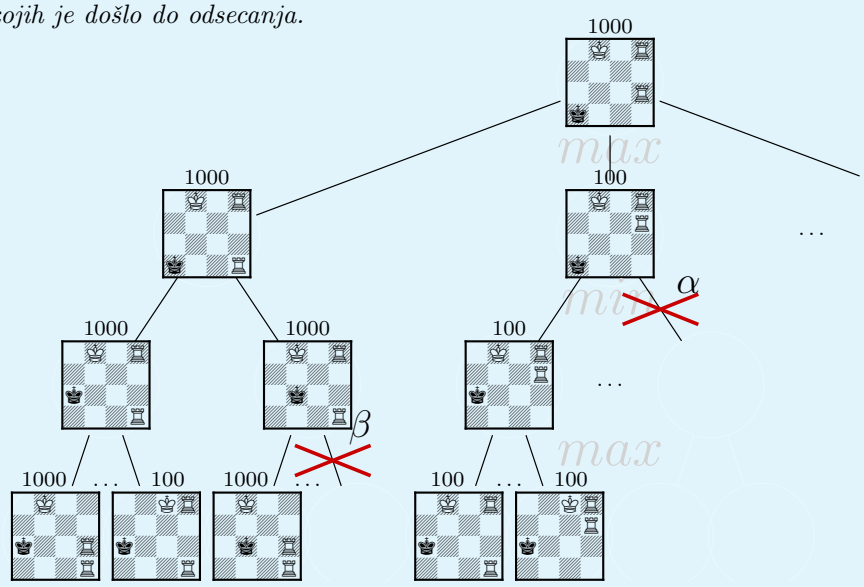
Slika 5.4: Ilustracija primene algoritma Alfa-beta.

Algoritam Alfa-beta zasnovan je na algoritmu Minimaks i heuristikama alfa i beta koje ubrzavaju pretragu odsecanjem delova stabla igre. Osnovni postupak ocenjivanja čvorova je minimaks tipa: funkcijom evaluacije ocenjuju se samo čvorovi na nekoj odabranoj dubini, a zatim se rekurzivnim postupkom (minimaksimizacijom) ocenjuju čvorovi prethodnici. Postupak „alfa odsecanja“ biće opisan pretpostavljajući da funkcija evaluacije za igrača A koji je na potezu ima pozitivan smisao (bolje su veće ocene). Neka je ocenjeno $n-1$ njegovih legalnih poteza, neka su dobijene ocene $a_1, a_2, \ldots, a_{n-1}$ i neka je a_k najveća od njih. Razmatramo n-ti legalni potez, neka je v čvor u koji taj potez vodi i neka je a_n njegova ocena koja tek treba da bude određena kao minimum ocena dece čvora v (slika 5.4). Nakon tog poteza, protivnik (igrač B) ima više mogućnosti i traži se ona sa najmanjom

ocenom. Za ocenu b_i bilo kog deteta čvora v važi da je ona veća od ili jednaka zajedničkom minimumu a_n, tj. važi $a_n \leq b_i$. Ako se u pretraživanju čvora v dođe do čvora sa ocenom b_j za koju važi $b_j \leq a_k$, sigurno je da je i ocena a_n čvora v manja od ili jednaka oceni a_k, jer važi $a_n \leq b_j \leq a_k$. Kako se u početnom čvoru traži maksimum ocena mogućih poteza, vrednost a_n, dakle, ne može da utiče na ocenu početnog čvora. Zato se dalje pretraživanje poteza protivnika u čvoru v može prekinuti bez uticaja na rezultat pretraživanja — može da se izvrši „alfa odsecanje stabla“ (slika 5.4). „Beta odsecanje“ potpuno je analogno i primenjuje se na čvorove u kojima je na potezu protivnik. Naravno, s obzirom na smisao funkcije evaluacije, maksimumi pominjani u „alfa odsecanju“ zamenjuju se minimumima i obratno (slika 5.4).

Algoritam Alfa-beta prikazan je na slici 5.5. Svaka instanca funkcije Max analizira po jedan čvor stabla igre i ima po jednu promenljivu a. Promenljiva a služi za određivanje maksimuma u čvoru i ona u svakom trenutku sadrži tekuću najveću pronađenu vrednost (tj. trenutno donje ograničenje za traženi maksimum). Vrednost α je trenutno najveća od vrednosti a za sve aktivne instance funkcije Max i ona se koristi za odsecanje u pozivima funkcije Min (u okviru naredbe „**ako** $b \leq \alpha$ **onda** vrati b;“). Vrednost α koja se koristi za odsecanje u nekom pozivu funkcije Min ne mora biti pronađena u funkciji Max koja je neposredno poziva, već i u nekom od poziva predaka. Naime, opravdano je odsecanje na osnovu tekuće vrednosti a u bilo kojem čvoru-pretku. Zato se vrednost α prosleđuje, rekurzivnim pozivima, dubljim čvorovima, gde će možda biti dodatno uvećana (što bi omogućilo još više odsecanja). Vrednost β u funkciji Max ima ulogu analognu ulozi vrednosti α u funkciji Min: ako je tekuća ocena a veća od ili jednaka β, onda se vrši odsecanje, tj. prekida se izvršavanje funkcije.

Primer 5.2. *Na narednoj slici prikazan je primer primene algoritma Alfa-beta na šah (pojednostavljenu verziju na tabli 4×4; funkcija evaluacije topu daje vrednost 50, a pobedi vrednost 1000). Označeni su delovi stabla igre kod kojih je došlo do odsecanja.*



Primetimo da je u ovom primeru moglo da se primeni i sledeće jednostavno odsecanje: ako neki čvor dobije maksimalnu moguću ocenu (1000 u ovom slučaju), onda nije potrebno ispitivati preostalu decu njegovog roditelja.

Kako je stablo igre obično ogromno, ubrzavanje Minimaks algoritma heuristikama „alfa-odsecanje“ i „beta-odsecanje“ ima izuzetan značaj. Pri tome, i algoritam Alfa-beta nalazi najbolji potez za zadati čvor (najbolji za zadatu maksimalnu dubinu pretraživanja i za izabranu funkciju evaluacije), što znači da heuristike koje se primenjuju ne narušavaju tu osobinu algoritma Minimaks.

Ukoliko se u svakom čvoru potezi ispituju od najlošijeg ka najboljem (u kontekstu tekućeg čvora)[3], tada nema nijednog alfa ili beta odsecanja, pa se algoritam Alfa-beta svodi na algoritam Minimaks. S druge strane, najviše alfa i beta odsecanja ima (i ispituje se najmanji broj čvorova stabla) kada se najpre ispituje potez najbolji u kontekstu tekućeg čvora (za korišćenu funkciju evaluacije i za zadatu dubinu). Pri tome, poredak preostalih poteza nije bitan. Naravno, nije moguće unapred znati koji je potez najbolji u datom čvoru, ali se i dobrim procenama (izborom jednog od boljih poteza) postižu dobri efekti. Upravo na toj ideji zasnivaju se i neke varijacije algoritma Alfa-beta.

[3] Da li je neki potez dobar zavisi od igrača iz čije se perspektive potez razmatra: ono što je dobro za jednog igrača nije za drugog i obratno. Kada se kaže „potez je dobar u kontekstu tekućeg čvora“, to znači da je dobar iz perspektive igrača koji je na potezu u tom čvoru.

Algoritam: Alfa-beta

Ulaz: Funkcija evaluacije f, pozicija, dozvoljena dubina pretraživanja d

Izlaz: Potez

1: $r := Max(f, pozicija, -\infty, +\infty, d)$;
2: vrati potez kojem odgovara vrednost r.

Algoritam: Max

Ulaz: Funkcija evaluacije f, pozicija, tekuće donje ograničenje α, tekuće gornje ograničenje β, dozvoljena dubina pretraživanja d

Izlaz: Vrednost pozicije

1: **ako** je pozicija završna ili je d jednako 0 **onda**
2: vrati $f(pozicija)$;
3: $a := -\infty$;
4: **za** svaku poziciju s do koje se može doći u jednom potezu **radi**
5: $m := Min(f, s, \alpha, \beta, d-1)$;
6: **ako** $m > a$ **onda**
7: $a := m$;
8: **ako** $a \geq \beta$ **onda**
9: vrati a;
10: **ako** $a > \alpha$ **onda**
11: $\alpha := a$;
12: vrati a.

Algoritam: Min

Ulaz: Funkcija evaluacije f, pozicija, tekuće donje ograničenje α, tekuće gornje ograničenje β, dozvoljena dubina pretraživanja d

Izlaz: Vrednost pozicije

1: **ako** je pozicija završna ili je d jednako 0 **onda**
2: vrati $f(pozicija)$;
3: $b := +\infty$;
4: **za** svaku poziciju s do koje se može doći u jednom potezu **radi**
5: $m := Max(f, s, \alpha, \beta, d-1)$;
6: **ako** $m < b$ **onda**
7: $b := m$;
8: **ako** $b \leq \alpha$ **onda**
9: vrati b;
10: **ako** $b < \beta$ **onda**
11: $\beta := b$;
12: vrati b.

Slika 5.5: Algoritam Alfa-beta.

5.4.4 Heuristika *kiler*

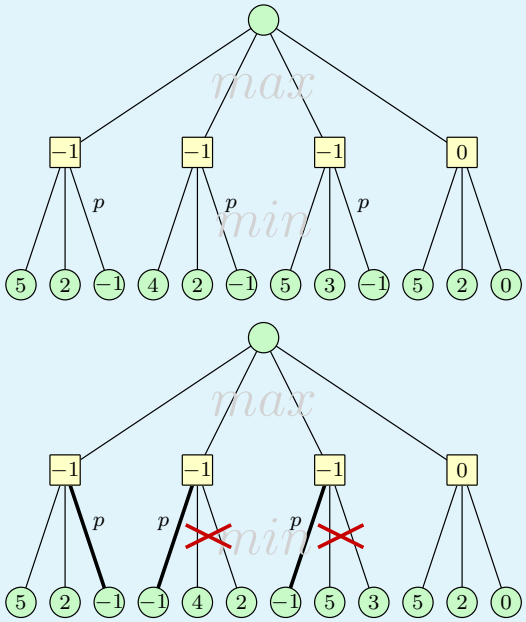
Tokom pretraživanja stabla igre, u svakom čvoru raspoloživi legalni potezi mogu se ispitivati u bilo kom poretku. Međutim, neki poredak može biti bolji od drugog, u smislu da omogućava veći broj alfa i beta odsecanja i time efikasniju pretragu. Heuristika *kiler* (eng. *killer*) ima za cilj da se u mnogim čvorovima razmatra najpre najbolji (ili makar veoma dobar) potez za taj čvor, kako bi bilo što više alfa i beta odsecanja. I ova heuristika je opšta i ne koristi specifična znanja o igri.

Neka se tokom pretraživanja stabla igre algoritmom Alfa-beta ocenjuje neki čvor, prvi na dubini d i neka je p pronađeni najbolji potez za taj čvor. Taj potez postaje *kiler potez* za dubinu d. Kada se sledeći put naiđe na neki čvor dubine d, ispitivanje poteza počinje od kiler poteza za tu dubinu.[4] Ukoliko se kasnije pokaže da je za taj nivo bolji neki drugi potez, onda on postaje kiler potez za dubinu d. Opisana heuristika primenjuje se za sve dubine do maksimalne dubine pretraživanja, tj. za svaku dubinu ažurira se po jedan kiler potez.

[4] Ukoliko u nekom čvoru tekući kiler potez nije legalan, ispitivanje poteza počinje od prvog sledećeg legalnog poteza (u postojećem uređenju poteza).

Kiler heuristika zasniva se na sledećem rasuđivanju: ukoliko je u jednoj grani stabla na dubini d najbolji potez p, ima izgleda da je on najbolji ili veoma dobar (ako je legalan) i u drugim čvorovima na istoj dubini. Ilustrujmo to na primeru šaha: neka igraču koji je na redu preti jedan te isti matni potez u sledećem potezu i neka nijedan njegov potez ne može da spreči taj matni potez protivnika. Pretraživanjem stabla u čvoru u kojem je na potezu protivnik, otkriva se pomenuti matni potez i on postaje kiler potez (za dubinu 1). Pri daljem pretraživanju stabla, na dubini 1 najpre se ispituje taj potez i kako on vodi pobedi protivnika, alfa odsecanje čini nepotrebnim dalje ispitivanje poteza u tom čvoru. Time se broj čvorova stabla koje u ovakvoj situaciji treba ispitati drastično smanjuje.

Primer 5.3. *Razmotrimo primer ilustrovan na narednoj slici – gornje stablo prikazuje pretraživanje algoritmom* Alfa-beta *bez kiler heuristike (nije bilo alfa i beta odsecanja), a donje stablo sa primenom kiler heuristike.*



Tokom ocenjivanja prvog čvora na dubini 1*, ocenjivana su njegova deca i kao najbolji pronađen je potez* p *(kojem odgovara ocena* -1*). Taj potez postaje* kiler potez *za dubinu* 1*. U naredna dva čvora na dubini* 1*, ispitivanje poteza počinje od tog istog poteza (a preostali legalni potezi uređeni su postojećim uređenjem). U oba slučaja dolazi do značajnog broja odsecanja. U četvrtom čvoru na dubini* 1 *tekući kiler potez* p *nije legalan, te ispitivanje poteza počinje od prvog sledećeg legalnog poteza (u nekom postojećem uređenju).*

Algoritam Alfa-beta proširen kiler heuristikom zovemo Alfa-beta/kiler. Primenom kiler heuristike ne menja se rezultat algoritma Alfa-beta (za istu funkciju evaluacije i istu dubinu pretraživanja) u smislu da se dobija potez sa istom ocenom (ne nužno i isti potez) kao primenom algoritama Alfa-beta ili Minimaks. Pri tome, takav potez dobija se najčešće sa bitno manjim brojem ispitanih čvorova stabla.

5.4.5 Iterativni Alfa-beta/kiler algoritam

U osnovnoj verziji kiler heuristike, kiler potez ne postoji na početnom nivou stabla igre, a upravo taj nivo bi mogao da omogući najveći broj odsecanja. Jedno rešenje za ovaj i još nekoliko drugih problema nudi iterativni Alfa-beta/kiler algoritam.

Za zadatu maksimalnu dubinu pretraživanja d_{max}, iterativni Alfa-beta/kiler algoritam realizuje se na sledeći način: algoritam Alfa-beta/kiler primenjuje se za zadati čvor u iteracijama: za maksimalnu dubinu pretraživanja 1, pa za maksimalnu dubinu pretraživanja 2, i tako dalje, sve do zadate maksimalne dubine d_{max}. Kad god se završi jedna iteracija, najbolji pronađeni potez postaje kiler potez za početni čvor (za nivo 0) i u sledećoj iteraciji ispitivanje svih legalnih poteza kreće od njega. Kiler potezi za dubine veće od 1 biraju se kao i obično (poglavlje 5.4.4). Finalno, kao najbolji potez za zadati čvor bira se onaj dobijen završnom primenom algoritma Alfa-beta/kiler — primenom za dubinu d_{max}.

Primer 5.4. *Ako bi iterativni Alfa-beta/kiler algoritam bio primenjen na stablo iz primera 5.3, u prvoj iteraciji bi se vršilo pretraživanje do dubine 1 i pozicije dostupne neposredno iz tekuće pozicije bile bi ocenjene statički. Potez koji vodi najboljoj oceni bio bi odabran za kiler potez za drugu iteraciju. Ponovimo da ocena dobijena pretraživanjem do dubine 1 možda nije jednaka oceni* 0 *(kao kada se vrši pretraživanje do dubine 2) i, takođe, izabrani potez možda nije onaj koji vodi oceni* 0 *iz primera 5.3 (gde je vršeno pretraživanje do dubine 2).*

Efekti ovog algoritma su, u svakoj iteraciji, slični efektima Alfa-beta/kiler algoritma, s tim što u iterativnom algoritmu postoji i kiler potez za početni čvor. Ima izgleda da je u svakoj iteraciji taj kiler potez bolje odabran i da daje sve bolje i bolje rezultate (veći broj alfa i beta odsecanja). Druga dobra osobina iterativnog algoritma je to što u slučaju prekida pretraživanja, praktično u svakom trenutku, ima smisleni rezultat kao najbolji pronađeni potez za neku kompletno završenu iteraciju (videti poglavlje 5.4.7).

Ono što izgleda kao mana algoritma — višestruko pretraživanje nekih čvorova — ne utiče bitno na performanse algoritma. Naime, u odnosu na vreme utrošeno za završnu iteraciju, vreme utrošeno na sve prethodne iteracije obično je zanemarljivo. Štaviše, ukoliko je odabran dobar kiler potez za početni čvor, završna iteracija će zahtevati ispitivanje znatno manjeg broja čvorova od običnog algoritma Alfa-beta/kiler za istu dubinu.

Primenom iterativnog Alfa-beta/kiler algoritma dobija se potez sa najboljom ocenom za datu funkciju evaluacije i datu dubinu pretraživanja, isto kao i primenom algoritama Minimaks, Alfa-beta i Alfa-beta/kiler (možda ne isti potez, ali potez sa istom minimaks ocenom).

5.4.6 Stabilno pretraživanje

Nedostatak pristupa u kojem se pretraživanje stabla igre vrši do fiksne dubine je u tome što funkcija evaluacije, koja se primenjuje na čvorove na najvećoj dubini, ne razmatra moguće nastavke za pozicije na najvećoj dubini. Te ocene, ma koliko funkcija evaluacije bila dobra, mogu da budu varljive i da vode lošem izboru poteza (u šahu se, na primer, može izabrati potez zbog nekog, naizgled dobrog, završnog čvora u kojem se zarobljava protivnikov top, ali se ne zna da nakon toga može da bude izgubljena kraljica ili da sledi mat). Zbog toga se primenjuje „stabilno pretraživanje“ (eng. *quiscence searching*): vrši se pretraživanje do neke fiksne dubine, ali se pretraživanje nastavlja i dalje ukoliko je, po nekom kriterijumu, završni čvor „nestabilan“. Maksimalna dubina dodatnog pretraživanja takođe treba da bude ograničena. Stabilno pretraživanje može se primenjivati u kombinaciji sa svakom od ranije opisanih tehnika.

Kriterijumi stabilnosti poteza određuju se u skladu sa specifičnostima konkretne igre. U šahu, na primer, pozicija se može smatrati stabilnom ako igrač koji je na potezu nije pod šahom, ako ne postoji nijedna napadnuta a nebranjena figura i ako ne preti neposredno izvođenje nekog protivnikovog pešaka. Savremeni programi za šah često pretražuju stablo igre do dubine desetak poteza, a sa dodatnim, stabilnim pretraživanjem do dubine dvadesetak poteza.

5.4.7 Prekidi i vremenska ograničenja

Vremenska ograničenja su važna u programiranju strateških igara: potrebno je da program izabere smislen potez i ukoliko se pretraživanje stabla igre prekine pre nego što se izvrši kompletan algoritam. Prekidi mogu biti izazvani akcijom korisnika ili ograničenjima vremena raspoloživog za jedan potez ili za celu partiju[5]. Kod do sada opisanih algoritama, ako algoritam nije kompletno izvršen (i, na primer, ispitani su samo neki, povoljni odgovori protivnika), odabrani potez može biti veoma loš. Izuzetak je iterativni Alfa-beta/kiler algoritam, jer praktično u svakom trenutku (trajanje prve iteracije je zanemarljivo) ima neku kompletno završenu iteraciju, te će biti izbegnuti makar najjednostavniji previdi.

5.4.8 Svojstva algoritama Minimaks i Alfa-beta

Delotvornost algoritama za pretragu stabla igre obično se analizira u terminima prosečnog i efektivnog faktora grananja. Time se procenjuje koliko neki algoritam ispituje čvorova u odnosu na potpunu pretragu stabla (do neke dubine). Pojmovi prosečnog i efektivnog faktora grananja u nastavku su definisani grubo i neformalno.

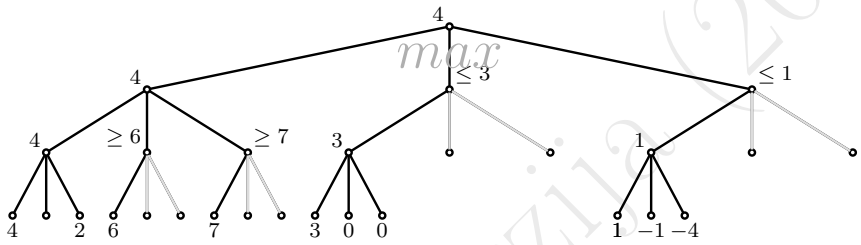
Prosečan faktor grananja je prosečan broj dece čvorova u stablu igre (stablo igre obično nije uniformno, tj. njegovi čvorovi obično imaju različit broj dece). Ta vrednost jedna je od ključnih karakteristika svake igre. Prosečan faktor grananja igre ekstremno je teško izračunati egzaktno za bilo koju netrivijalnu igru, te se ova

[5] Ukoliko je ograničeno vreme raspoloživo za celu partiju, program mora i da ga ekonomično deli na procenjeni broj poteza.

vrednost obično samo procenjuje na osnovu eksperimenata. Na primer, za šahovsku središnjicu procenjuje se da je prosečan faktor grananja oko 35, a za igru go oko 250.

Efektivni faktor grananja za igru G i algoritam A, u oznaci $B(G, A)$, prosečan je broj dece koje algoritam A ispituje u čvorovima stabla igre. Kao i prosečan faktor grananja, efektivni faktor grananja je teško izračunati, te se ova vrednost obično samo procenjuje na osnovu prosečnog faktora grananja i svojstava datog algoritma i to u nekim specijalnim slučajevima. Efektivni faktor grananja može biti upotrebljen za procenu koliko će čvorova stabla igre G primenom nekog algoritma A biti ispitano na nekoj dubini d: $B(G, A)^d$. Ta vrednost može da služi kao mera efikasnosti algoritma A.

Ako je prosečan faktor grananja igre G jednak b, onda je, trivijalno, $B(G, \textsf{Minimaks}) = b$. Problem određivanja vrednosti $B(G, \textsf{Alfa-beta})$ znatno je teži. Ukoliko se u svakom čvoru stabla igre ispituju potezi od najlošijeg ka najboljem, onda, očigledno, nema nikakvih odsecanja i algoritam Alfa-beta se ponaša kao algoritam Minimaks. S druge strane, algoritam Alfa-beta ponaša se najbolje ukoliko se u svakom čvoru najpre ispituje najbolji potez (poredak ostalih poteza nije bitan). Razmotrimo u nastavku taj slučaj. Tokom primene algoritma Alfa-beta, nekim čvorovima izračunava se egzaktna minimaks ocena za zadatu dubinu, a nekim čvorovima ocena se računa samo onoliko koliko je neophodno za odsecanja. Na primer, na slici 5.6, za čvorove sa ocenom 4, ova ocena je egzaktna (ista ocena bila bi izračunata i da se uopšte ne primenjuju odsecanja), a čvoru sa ocenom ≥ 7, ocena 7 nije nužno egzaktna (možda je veća), jer je primenjeno odsecanje i nisu ispitani svi njegovi potomci.



Slika 5.6: Ilustracija izračunavanja egzaktne ocene čvora i granice ocene čvora.

Neka $T(d)$ označava broj listova koje treba ispitati za određivanje egzaktne ocene čvora na nivou d (za zadato stablo), pri čemu je u ovom konkretnom kontekstu pogodno da se nivoi broje od listova – njima odgovara nivo 0. Neka $L(d)$ označava broj listova koje treba ispitati za određivanje (samo) ograničenja ocene čvora na nivou d dovoljnog da obezbedi odsecanja u nastavku primene algoritma. Važi $T(0) = 1$ i $L(0) = 1$. Kako se u svakom čvoru najpre ispituje najbolji potez, onda važi (slika 5.6):

$$T(d) = T(d-1) + (b-1)L(d-1)$$

$$L(d) = T(d-1)$$

odakle sledi

$$T(d) = T(d-1) + (b-1)T(d-2)$$

Karakteristična jednačine ove rekurentne veze je $t^2 - t - (b-1) = 0$ i njena rešenja su

$t_1 = \frac{(1-\sqrt{1+4(b-1)}}{2} = 1/2 - \sqrt{b-3/4}$

$t_2 = \frac{(1+\sqrt{1+4(b-1)}}{2} = 1/2 + \sqrt{b-3/4}$

Dakle, važi:

$$T(d) = \lambda_1 \cdot \left(1/2 - \sqrt{b-3/4}\right)^d + \lambda_2 \cdot \left(1/2 + \sqrt{b-3/4}\right)^d$$

i $T(d) = \Theta(b^{d/2})$, što govori da je efektivni faktor grananja za algoritam Alfa-beta za ovaj slučaj jednak $b^{1/2}$. Kiler heuristika u iterativnoj verziji često postiže dobar poredak poteza u svakom čvoru, pa takav efektivni faktor grananja nije nerealno očekivati.

Problem određivanja efektivnog faktora grananja algoritma Alfa-beta sa slučajnim poretkom grana još je složeniji i odgovor, bez dokaza, navodimo u okviru naredne teoreme.

Teorema 5.1. *Za igru G sa uniformnim stablom i faktorom grananja b, za efektivni faktor grananja važi da je*

- *jednak b, za algoritam Minimaks, kao i za algoritam Alfa-beta, u slučaju da se potezi ispituju od najlošijem ka najboljem;*

- *približno jednak $b^{3/4}$, za algoritam Alfa-beta, u slučaju da se potezi ispituju u slučajnom poretku;*
- *približno jednak $b^{1/2}$, za algoritam Alfa-beta, u slučaju da se potezi ispituju počev od najboljeg.*

Navedeno tvrđenje govori i da je, uz idealan poredak poteza, algoritam Alfa-beta moguće primeniti, uz približno isto vreme izvršavanja, na dva puta veću maksimalnu dubinu nego algoritam Minimaks. Mnoge igre ne ispunjavaju uslov teoreme da je stablo igre uniformno, ali ako je prosečan faktor grananja b, teorema daje dobru procenu i za te slučajeve.

Važno je i sledeće srodno tvrđenje.

Teorema 5.2. *Algoritam Alfa-beta je asimptotski optimalan algoritam za pretraživanje stabla igre.*

Navedeno tvrđenje znači da ne postoji algoritam za pretraživanje stabla igre koji, u opštem slučaju, asimptotski ispituje manje završnih čvorova nego algoritam Alfa-beta. Modifikacije algoritma Alfa-beta opisane u prethodnom delu veoma često u praksi daju bolje rezultate nego osnovni algoritam. Ipak, efektivni faktori grananja ovih algoritama ne razlikuju se od efektivnog faktora grananja algoritma Alfa-beta i u opštem slučaju oni ne garantuju asimptotski manje ispitanih završnih čvorova stabla nego algoritam Alfa-beta.

5.4.9 Monte Karlo pretraga stabla igre

Pored algoritama koji vrše sistematičnu pretragu dela stabla igre ili vrše pretragu koja je ekvivalentna sistematičnoj pretrazi, postoje i algoritmi koji su zasnovani na uzorkovanju putanja kroz stablo igre. I oni se najčešće koriste u središnjici.

Heuristička pretraga Monte Karlo zasniva se na uzorkovanju prostora pretrage a primenjuje se još od četrdesetih godina prošlog veka u mnogim oblastima. Od 2006. godine primenjuje se i u pretrazi stabala igara, kada se naziva *Monte Karlo pretraga stabla* (eng. *Monte Carlo tree search*). U ovom kontekstu, simulacijom odigravanja partija obilazi se samo jedan deo odgovarajućeg stabla pretrage. Time se aproksimira njegov (puni) obilazak i donose zaključci na osnovu nepotpunih informacija.

Do najznamenitijih rezultata Monte Karlo pretrage u strateškim igrama došlo se tek u prethodnih nekoliko godina. Programi AlphaGo i AlphaZero (kompanije DeepMind), koji koriste ovu vrstu pretrage u sadejstvu sa neuronskim mrežama (poglavlje 11.5), pobedili su 2016. i 2017. godine vodeće svetske igrače i programe u igrama go i šah. Mehanizam koji je u daljem tekstu pojednostavljeno opisan, program AlphaZero koristio je za samotreniranje, na nekoliko hiljada procesora, tokom perioda od dvadeset četiri sata. Ishod je superiorna pobeda nad jednim od najboljih šahovskih programa u tom trenutku (Stockfish 8) od 64:36 i osvojena znanja o mnogim otvaranjima za koja su ljudima bili potrebni vekovi. Uprkos ovim uspesima, postoji i uverenje da će najbolji programi za strateške igre u budućnosti (vodeći računa i o raspoloživim resursima) kombinovati Monte Karlo pretragu i neuronske mreže sa algoritmima minimaks tipa opisanim ranije.

Monte Karlo pretraga koristi se u strateškim igrama bez ikakvog domenskog znanja, tj. iskustvenog znanja o konkretnoj igri i jedino što je potrebno to je poznavanje pravila igre, tj. skupa legalnih poteza, završnih stanja i mogućih ishoda igre (na primer: pobeda, remi i poraz). Nepotpuno stablo igre formira se i modifikuje u iteracijama, postepenim popunjavanjem novim čvorovima. Čvorovima stabla (tj. pozicijama) pridružene su ocene koje treba da odražavaju njihov kvalitet. Ocene pripadaju intervalu $[0, 1]$ i ako je (u nekom trenutku) ocena nekog čvora 1 ili skoro 1, onda je ta pozicija veoma dobra za igrača koji je na potezu, a ako je ocena 0 ili skoro 0, onda je ta pozicija veoma dobra za igrača koji nije na potezu. Ocene čvorova ažuriraju se u iteracijama. U svakoj iteraciji dodaje se novi čvor iz kojeg se odigravaju partije simulacijama – do kraja partije, slučajnim odigravanjem poteza, bez ikakvog netrivijalnog rasuđivanja. Na osnovu takvih simulacija i dobijenih ocena, postepeno se profinjuju i ocene čvorova koji su preci novog čvora. Taj postupak nastavlja se sve dok to dozvoljavaju raspoloživi resursi (računarsko vreme i memorija). Ocene čvorova obično su jednake w/n, gde je n broj simulacija, a w zbir ishoda svih simulacija iz tog čvora. Ishodi mogu, na primer, biti vrednost 1 za pobedu igrača koji je na potezu, 0.5 za remi i 0 za izgubljenu partiju. Ako su ocene dece nekog čvora jednake w_i/n_i $i = 1, \ldots, k$ onda je ocena tog čvora jednaka w/n, gde je $n = \sum_{i=1}^{k} n_i$ i $w = n - \sum_{i=1}^{k} w_i$ (i ta veza se održava za sve čvorove kada se ažuriraju ocene čvorova tokom propagacije). Primetimo da je, za razliku od minimaks ocena, smisao ocena w/n na svim dubinama stable igre isti — na svakoj dubini, za igrača koji je na potezu veće ocene su bolje.

Svaka iteracija opisanog postupka ima sledeće faze:

- selekcija: bira se čvor M koji nije završno stanje igre i iz kojeg se jednim potezom može dobiti pozicija m koja još uvek nije posećena (primetimo da M nije nužno list tekućeg stabla);
- razgranavanje: čvor sa pozicijom m pridodaje se stablu i pridružuje mu se ocena 0/0;

- simulacija: pokreće se simulacija nastavka partije iz čvora m, nasumičnim izborom poteza i određuje se ocena te simulacije (na primer 1, ako je u simulaciji pobedio igrač koji je na potezu u čvoru m);
- propagacija: ažuriraju se ocene svih čvorova od m do korena novodobijenim rezultatom.

U svakoj iteraciji, dakle, dodaje se po jedan čvor (m) koji će biti ocenjen korišćenjem simulacija. Načine na koje se bira čvor M u okviru selekcije zovemo *politikama selekcije*. Politika selekcije primenjuje se rekurzivno, počevši od korena. Postoje mnoge politike selekcije. Najjednostavnija je da se čvor na tekućem nivou bira slučajno. Selekcija može da se sprovodi i tako što se od korena ka dubljim čvorovima uvek bira čvor koji ima najbolju ocenu, na primer, ocenu $\frac{w}{n}$. Time se obezbeđuje da se bolji potezi detaljnije analiziraju. Međutim, potrebno je ostavljati i mogućnost da se dodatno analiziraju čvorovi sa lošijim ocenama kako ne bi došlo do previda nekog veoma dobrog poteza. To rade još naprednije politike selekcije koje koriste ocene poput:

$$\frac{w}{n} + \sqrt{\frac{c \ln N}{n}}$$

gde je N ukupan broj simulacija iz roditeljskog čvora, a c pogodan parametar koji se bira empirijski (obično je jednak 2).

Simulacija iz novokreiranog čvora ne mora da se sprovodi do završnih stanja, već i do nekih drugih predefinisanih stanja (u kojima se jasno procenjuje ishod partije). Dodatno, simulacije mogu da budu obogaćene nekim znanjima kako ne bi bile potpuno slučajne. Umesto jedne simulacije iz novog čvora, može da se izvrši više njih. Simulacije iz više čvorova mogu da se vrše nezavisno što omogućava razne varijante paralelizacije Monte Karlo pretrage. Ukupan broj simulacija može se smanjiti korišćenjem mašinskog učenja i generalizacijom na stanja koja uopšte nisu viđena.

Primer 5.5. *Na slici 5.7 ilustrovana je jedna iteracija primene Monte Karlo pretrage stabla igre. U čvorovima su ocene w/n, gde je n broj simulacija, w zbirni ishod svih simulacija iz tog čvora. U koraku selekcije bira se čvor koji ima ocenu $1/1$, on se razgranava – dodaje se jedno njegovo dete i tom detetu se pridružuje ocena $0/0$. Zatim se vrši simulacija igre od tog deteta i dobija se da igru gubi igrač koji igra u toj poziciji. Ocena deteta se ažurira na $0/1$. Kako gubi igrač koji igra u poziciji u čvoru-detetu, to znači da dobija igrač koji igra u čvoru-roditelju, te se ocena $1/1$ ažurira na $2/2$ (što znači da su dobijene dve partije od dve simulacije). Dobijena ocena se, zatim, analogno propagira do samog korena stabla.*

Stablo koje kreira Monte Karlo metod ne razvija se nužno simetrično. Time se obezbeđuje njegova efikasnost i praktična upotrebljivost, ali u nekim situacijama to može da bude i loše, jer u ograničenom vremenu može da se desi da se metod fokusira na neku lošu granu, previdevši postojanje neke mnogo bolje.

Složenost Monte Karlo pretrage proporcionalna je broju dozvoljenih simulacija. I nakon kratkog utrošenog vremena, metod ima nekakve upotrebljive ocene za raspložive poteze.

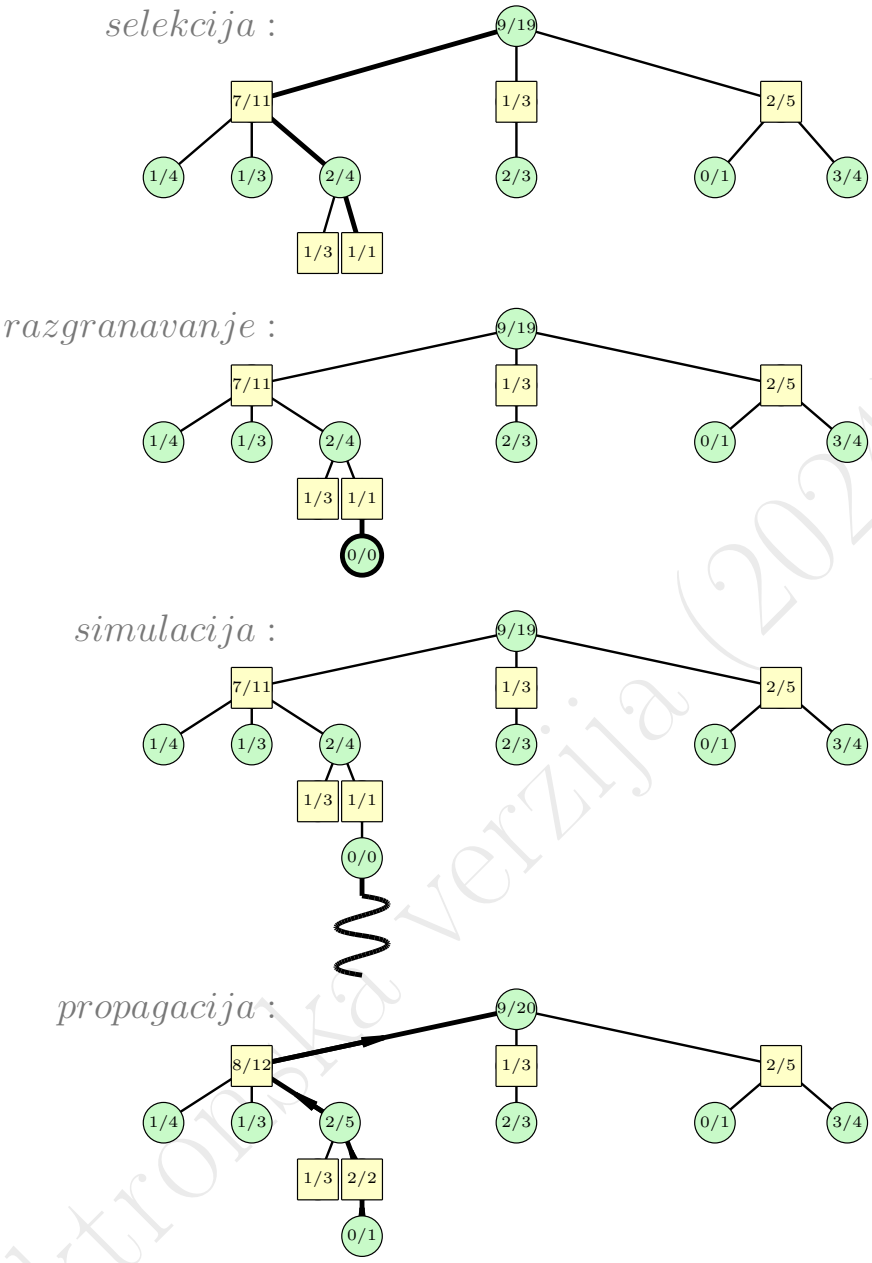
Način rada Monte Karlo pretrage stabla igre suštinski je drugačiji od načina rada algoritama Minimaks i Alfa-beta. Na primer, algoritam Minimaks i osnovni algoritam Alfa-beta treba da se izvrše do kraja da bi dali najbolji potez za zadatu dubinu, dok Monte Karlo pretraga može da se prekine u bilo kom trenutku, s tim da daje bolje poteze što duže radi. S druge strane, postoji i snažna veza između algoritma Minimaks i Monte Karlo pretrage stabla igre. Pretpostavimo da se algoritam Minimaks primenjuje do listova stabla igre, na primer, do ishoda -1, 0, 1 i da su minimaksimizacijom ocenjeni svi čvorovi stabla. Neka su te ocene transformisane u ocene 0, $1/2$, 1 na sledeći način: neka se na *max* nivoima stabla ocene x preslikavaju u ocene $(x+1)/2$, a na *min* nivoima stabla u ocene $(-x+1)/2$. Može se dokazati da najbolji potezi na osnovu Monte Karlo pretrage, sa opisanom naprednom selekcijom, konvergiraju najboljim potezima na osnovu algoritma Minimaks koji je primenjen nad transformisanim ocenama (preciznije, verovatnoća da Monte Karlo pretraga oceni kao najbolji potez potez koji je najbolji u smislu algoritma Minimaks teži 1).

5.5 Završnica

U igrama kao što je šah, završnica iziskuje specifične tehnike.[6] Naime, pristupi koji se zasnivaju na dubinskom pretraživanju ne daju dobre rezultate u završnici jer kvalitetna igra iziskuje jako veliku dubinu pretraživanja. Problem završnice još je teži ako se postavi zahtev za korektnom ili optimalnom igrom/taktikom.[7] Ukoliko se,

[6] Završnica u šahu se, pojednostavljeno, može opisati kao stanje igre u kojem je na tabli preostao samo neki mali broj figura, na primer, manje ili jednako sedam.

[7] Za taktiku kažemo da je *korektna* ukoliko iz svake pozicije garantovano vodi ka najboljem mogućem ishodu. Za taktiku kažemo da je *optimalna* ukoliko u dobijenoj poziciji vodi pobedi u najmanjem broju poteza i ukoliko u izgubljenoj poziciji poraz maksimalno odlaže. Očigledno, ako je taktika optimalna, onda je i korektna, ali ne važi obratno.



Slika 5.7: Ilustracija primene Monte Karlo pretrage stabla igre.

tehnikama koje se koriste u središnjici, sistematsko pretraživanje vrši do završnih čvorova time se obezbeđuje optimalna strategija (naravno, za većinu igara to je u praksi nemoguće izvesti). Optimalnu igru u šahovskoj završnici praktično je nemoguće obezbediti (za uobičajena vremenska ograničenja), jer su moguće završnice u kojoj igrač ima dobijenu poziciju, ali ne može da matira protivnika u manje od, na primer, četrdeset poteza,[8] pa takva završnica za optimalnu igru zahteva preveliku dubinu pretraživanja. Slični problemi važe i za korektnu igru. U daljem tekstu opisani su samo neki od pristupa koji se koriste u šahovskim završnicama.

Klase ekvivalencije pozicija. U ovom pristupu za šah (koji je opisao Max Bramer, 1975. godine), sve legalne pozicije podržanih završnica svrstane su u klase ekvivalencija, tako da istoj klasi pripadaju suštinski slične pozicije (na primer, sve pozicije „kralj i pešak protiv kralja“). Svakoj od tih klasa ekvivalencije pridružena je jedna konkretna ocena i jedna funkcija evaluacije. U toku igre, bira se, bez ikakvog pretraživanja u dubinu, potez kojem odgovara najveći zbir ocene klase i ocene konkretne pozicije u toj klasi u koju taj potez vodi. Ovaj pristup zahteva definisanje mnogih klasa završnica i kreiranje pratećih, vrlo specifičnih, funkcija evaluacije (na

[8] Tabele Lomonosov pokazale su da postoje pozicije u kojima beli dobija, ali ne nužno u manje od (čak) 545 poteza.

primer, u nekim završnicama važno je rastojanje između dva kralja, što je parametar koji nije relevantan u središnjici).

Završni saveti. Jedan od pristupa koji se primenjuje u šahovskim završnicama je pristup koji ćemo ovde zvati *završni saveti* ili *strategije za završnice* (a izvorni naziv je *advice texts*). Svaka konkretna vrsta završnice zahteva posebnu, specijalizovanu strategiju (i prateći algoritam). Navedimo, kao ilustraciju, jedan „završni savet“ za završnicu „kralj i top protiv kralja“ (autora Ivana Bratka).

1. „mat“: pokušaj da matiraš protivnika u dva poteza;
2. „stezanje“: ako to nije moguće, pokušaj da topom smanjiš prostor na tabli dostupan protivničkom kralju;
3. „približavanje“: ako to nije moguće, pronađi način da svog kralja približiš protivničkom;
4. „zadržavanje“: ako to nije moguće, pronađi potez koji zadržava trenutno stanje u smislu (2) i (3) (tj. odaberi „potez čekanja“);
5. „razdvajanje“: ako to nije moguće, pronađi potez kojim se dobija pozicija u kojoj top razdvaja dva kralja, bilo vertikalno ili horizontalno.

Nedostatak pristupa ilustrovanog navedenim primerom je u tome što iziskuje posebne „završne savete“ za sve suštinski različite završnice. Pored toga, za sve tipove završnica nije jednostavno (ili nije moguće) napraviti koncizan i efikasan „završni savet“. Prethodnih godina za generisanje ovakvih „završnih saveta“ koriste se i tehnike mašinskog učenja.

Glava 6

Genetski algoritmi

Heuristike koje se koriste u rešavanju problema pretrage dizajnirane su za konkretan problem, imajući u vidu njegove specifičnosti. Heuristika dizajnirana za jedan problem često je potpuno neupotrebljiva za drugi problem i rešavanje svakog novog problema korišćenjem heuristika može da bude veoma zahtevno. S druge strane, *metaheuristike* ili *metaheurističke metode* su metode koje opisuju opšte strategije pretrage za rešavanje optimizacionih problema i formulisane su nezavisno od konkretnog problema. U svom opštem obliku, metaheuristike u procesu rešavanja ne koriste specifičnosti nijednog konkretnog problema te se mogu koristiti za široke klase problema. Međutim, one ipak moraju biti prilagođene i instancirane za konkretan problem koji se rešava. Metaheuristike ne vrše sistematičnu pretragu skupa svih potencijalnih rešenja, već obično razmatraju samo njegov mali uzorak i ne garantuju pronalaženje najboljeg mogućeg rešenja. Međutim, rešenja koja daju metaheuristike često mogu biti dovoljno dobra, posebno u situacijama kada nije raspoloživa ili nije praktično upotrebljiva odgovarajuća egzaktna metoda (koja garantuje pronalaženje najboljeg mogućeg rešenja).

Genetski algoritmi pripadaju široj grupi metaheurističkih algoritama globalne optimizacije ili pretrage koji koriste tehnike inspirisane biologijom. Genetski algoritmi koriste pojmove kao što su selekcija, ukrštanje, nasleđivanje, mutacija, itd. U prirodi, evolucija je proces u kojem jedinke koje su najbolje prilagođene okolini preživljavaju i ostavljaju potomstvo, koje je najčešće isto tako ili bolje prilagođeno okolini. Svaka ćelija svakog živog organizma sadrži hromozome. Svaki hromozom sadrži skup gena — blokove DNK. Svaki gen određuje neku osobinu organizma. Familija gena često se naziva genotip, a familija osobina fenotip. Reprodukcija organizama uključuje kombinovanje gena roditelja i, pored toga, mali broj mutacija. Jedinka može biti manje ili više prilagođena okolini. Jedinka koja je bolje prilagođena okolini u kojoj živi ima veću verovatnoću preživljavanja i ostavljanja potomstva, a time i prenošenja svog genetskog materijala. Genetski materijal prilagođenih jedinki uglavnom opstaje, dok genetski materijal neprilagođenih jedinki uglavnom nestaje kroz generacije. Dakle, evolucioni procesi u prirodi su, u određenom smislu, optimizacioni procesi — procesi u kojima se kroz generacije optimizuje genetski materijal (tj. osobine organizama) tako da bude što bolje prilagođen okolini. Genetski algoritmi koriste se za nalaženje tačnog ili približnog rešenja nekog problema optimizacije ili pretrage.

Mada je još pedesetih godina dvadesetog veka bilo računarskih simulacija zasnovanih na evoluciji, smatra se da je moderne genetske algoritme uveo Džon Holand (John Holland) sedamdesetih godina dvadesetog veka, a postali su popularni kasnih osamdesetih godina. Tokom prethodnih tridesetak godina ostvaren je veliki napredak u razvoju genetskih algoritama. Genetski algoritmi mogu se koristiti za rešavanje širokog skupa problema, često NP-kompletnih ili još težih, za koje ne postoje efikasna egzaktna rešenja. Genetski algoritmi imaju uspešne primene u ekonomiji, tehnici, bioinformatici, hemiji, fizici itd. Genetski algoritmi mogu se uspešno koristiti i u rešavanju optimizacionih problema u kojima postoji više lokalnih ekstremuma. Popularnost genetskih algoritama potiče iz njihove uspešnosti, ali i jednostavnosti. Naime, ideje na kojima su genetski algoritmi zasnovani su jednostavne za razumevanje i implementiranje, a daju opšti sistem pretrage primenljiv na veliki broj problema. Pored toga, i u situacijama kada ne nalaze globalne ekstremume, rešenja koja daju često su dovoljno dobra. Genetski algoritmi mogu davati odlične rezultate kada se koriste u sadejstvu sa drugim optimizacionim metodama. Uporedo sa nalaženjem brojnih novih primena i unapređivanjem algoritma, razvijaju se i teorijske osnove genetskih algoritama, ali još uvek sa ograničenim uspesima. Na primer, iako često nalaze globalne ekstremume, genetski algoritmi ne pružaju informaciju o tome da li je u pitanju globalni ili lokalni ekstremum, niti o tome sa kolikom greškom je određeno rešenje.

6.1 Opšti genetski algoritam

Genetski algoritmi treba da, kao svoj rezultat, vrate tačno ili približno rešenje nekog problema optimizacije ili pretrage. Rezultat može biti numerička vrednost, matematička funkcija (na primer, polinom stepena 5 koji

najbolje aproksimira dati skup tačaka u ravni), put u grafu (na primer, najkraći put, dužine manje od 50, između dve tačke u uniformnoj mreži), itd.

Genetski algoritmi implementiraju se kao računarska simulacija u kojoj populacija apstraktno opisanih jedinki koje su kandidati za rešenje problema treba da se približava boljim rešenjima. Reprezentacija jedinke naziva se *hromozomom* ili *genotipom*. Cilj je naći vrednost za koju zadata *funkcija cilja* dostiže svoj ekstremum ili vrednost koja je dovoljno blizu ekstremuma. Potencijalna rešenja, tj. jedinke obično su predstavljene nizovima nula i jedinica, ali su moguće i druge reprezentacije za probleme u kojima binarna reprezentacija nije pogodna. Postupak se odvija kroz generacije. Obično u svakoj generaciji postoji isti broj jedinki. Početnu generaciju često čine slučajno generisane jedinke, ali ona može da sadrži i jedinke koje su (grubi) rezultat neke druge, jeftinije optimizacione metode.

Za svaku jedinku računa se njen kvalitet (koji odgovara prilagođenosti okolini). Funkcija koja pridružuje te vrednosti jedinkama naziva se *funkcija prilagođenosti* ili *funkcija kvaliteta*. Ova funkcija ima ključnu ulogu u algoritmu. Ona može ali ne mora da bude jednaka funkciji cilja.

Iz jedne generacije se, na osnovu vrednosti funkcije prilagođenosti, kroz proces *selekcije* biraju jedinke koje će biti iskorišćene za stvaranje novih jedinki (potomstva). One kvalitetnije biraju se sa većom verovatnoćom. Nad izabranim jedinkama primenjuju se genetski operatori *ukrštanja*[1] i tako se dobijaju nove jedinke. Ukrštanjem se od dve jedinke dobija nova (ili dve nove) sa genetskim materijalom koji je dobijen neposredno od roditelja, tj. od polaznih jedinki. Operatorom *mutacije* može da se modifikuje deo jedinke (i on oponaša mutacije koje se u prirodi javljaju pod uticajem spoljnih faktora). U svakoj generaciji, dakle, zbog rekombinacije gena mogu da se javljaju brojne sličnosti i različitosti između jedinki te generacije i njihovih potomaka.

Politika zamene generacija određuje kako se od postojećih jedinki i njihovog potomstva kreira nova generacija. Tako dobijena nova generacija koristi se za sledeću iteraciju algoritma. Neke jedinke u novoj generaciji mogu biti bolje, neke mogu biti i lošije od jedinki iz prethodne generacije, ali se očekuje da se prosečna prilagođenost popravlja.

Postupak se zaustavlja kada je dostignut zadati broj generacija, kada je dostignut željeni nivo kvaliteta populacije (na primer, prilagođenost najprilagođenije jedinke) ili kada je ispunjen neki drugi uslov. Ukoliko je dostignut zadati broj generacija, nema nikakvih garancija da tekuća najkvalitetnija jedinka ima zadovoljavajuću vrednost funkcije cilja.

Opšti genetski algoritam prikazan je na slici 6.1. U konkretnim primenama, on se dodatno precizira i prilagođava za rešavanje nekog problema tj. za rešavanje neke klase njegovih instanci. Na primer, genetski algoritam može da bude dizajniran tako da rešava neke slične instance problema pravljenja rasporeda časova: za neke moguće brojeve nastavnika, odeljenja, učionica, termina i slično. Ulaz za algoritam čini opis konkretne instance problema na osnovu kojeg treba izabrati funkciju prilagođenosti, ali i brojna podešavanja algoritma — izbor pogodne politike selekcije, politike zamene generacija, verovatnoće ukrštanja, verovatnoće mutacije i drugih parametara. Često je najteži deo primene genetskih algoritama upravo pravljenje ovih izbora, dok implementiranje opšteg genetskog algoritma obično čini samo mali deo ukupnog potrebnog truda. Izbor pogodnih parametara često se za određenu instancu ili klasu instanci bira empirijski, isprobavanjem raznih mogućih vrednosti.

Primetimo da funkcija cilja ne figuriše u opisu opšteg genetskog algoritma. Ipak, ona je prisutna implicitno, kroz funkciju prilagođenosti.

Algoritam: Opšti genetski algoritam

Ulaz: Podaci koji određuju funkciju cilja i podešavanja algoritma

Izlaz: Tačno ili približno rešenje problema određenog zadatom funkcijom cilja

1: generiši početnu populaciju jedinki;
2: izračunaj prilagođenost svake jedinke u populaciji;
3: **ponavljaj**
4: izaberi iz populacije skup jedinki za reprodukciju;
5: primenom operatora ukrštanja i mutacije kreiraj nove jedinke (i računaj njihovu prilagođenost);
6: na osnovu starih i novih jedinki, kreiraj novu generaciju;
7: **dok nije ispunjen** uslov zaustavljanja;
8: vrati najkvalitetniju jedinku u poslednjoj populaciji.

Slika 6.1: Opšti genetski algoritam.

[1] Ovaj termin nije sasvim u skladu sa značenjem koje ima u biologiji.

6.2 Komponente genetskog algoritma

Pored podataka koji određuju funkciju cilja, da bi navedeni opšti algoritam bio upotrebljiv potrebno je instancirati ga za konkretan problem – definisati reprezentaciju jedinki, funkciju prilagođenosti, politiku selekcije, politiku zamene generacija, itd.

6.2.1 Reprezentacija jedinki

Jedinke mogu biti predstavljene raznovrsnim strukturama podataka, na primer, nizovima binarnih cifara, stablima, matricama i slično. Neophodno je da izabrana reprezentacija jedinki može da opiše moguće rešenje razmatranog problema. Ukoliko je potencijalno rešenje sačinjeno od nekoliko vrednosti, onda i reprezentacija jedinke treba to da oslikava — u takvim situacijama, svaka jedinka obično je sačinjena od nekoliko delova objedinjenih u celinu, na primer, u niz binarnih cifara.

Izbor reprezentacije može bitno da utiče na performanse algoritma. Neophodno je da nad izabranom reprezentacijom mogu da se definišu genetski operatori (ukrštanje i mutacija). Poželjno je da genetski operatori budu definisani tako da se njima dobijaju samo jedinke koje su smislene u odabranoj reprezentaciji, tj. da se ne dobijaju jedinke koje ne predstavljaju moguća rešenja (na primer, nelegalni putevi u grafu), jer bi one narušavale performanse algoritma. Međutim, nekada se koriste i takvi operatori, ali se onda moraju definisati mehanizmi popravljanja neispravnih jedinki, tako da one opisuju moguća rešenja.

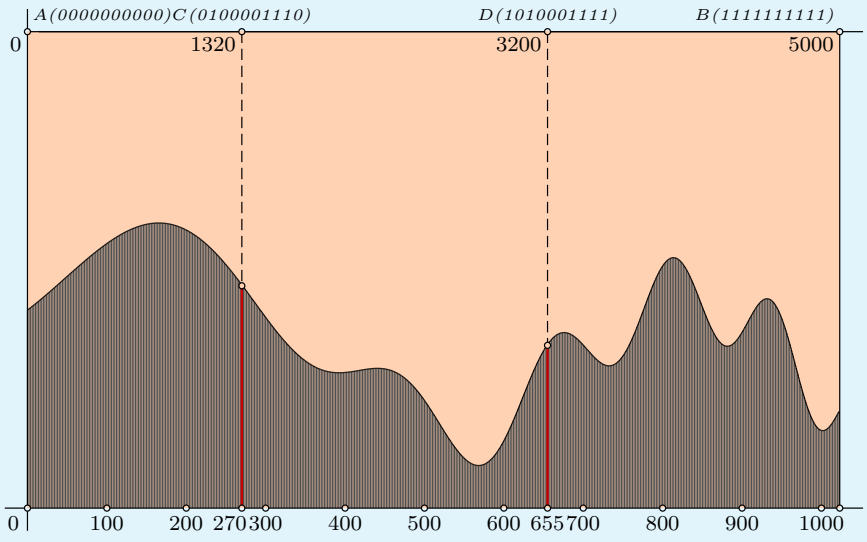
Najčešće korišćena reprezentacija jedinki je u vidu nizova binarnih cifara (analogno nizovima bitova u digitalnim računarima). Svaki deo hromozoma, tj. svaku cifru u takvoj reprezentaciji, zovemo *gen*. Dublja priroda binarne reprezentacije zavisi od konkretnog problema. Na primer, ako je dužina hromozoma n binarnih cifara (tj. bitova) i ako je prostor mogućih rešenja interval realnih brojeva $[a, b]$, onda je potrebno uspostaviti vezu (koja, naravno, nije bijektivna) između nizova n binarnih cifara i realnih brojeva iz datog intervala. Tako će binarna reprezentacija $\underbrace{000\ldots0}_{n}$ odgovarati broju a, a binarna reprezentacija $\underbrace{111\ldots1}_{n}$ broju b. Broju x sa binarnom reprezentacijom između $\underbrace{000\ldots0}_{n}$ i $\underbrace{111\ldots1}_{n}$ odgovara realni broj

$$a + \frac{x}{2^n - 1}(b - a)$$

S druge strane, realnom broju x iz intervala $[a, b]$ pridružujemo niz binarnih cifara koji predstavlja binarnu reprezentaciju broja

$$\left[\frac{x - a}{b - a}(2^n - 1)\right].$$

Primer 6.1. *Pretpostavimo da naftnu platformu treba postaviti na pogodnom mestu na putu između tačaka A i B, koji je dužine* 5000*m. Lokacija platforme je pogodnija ukoliko na tom mestu postoje veće rezerve nafte.*



U biranju lokacije platforme moguće je meriti postojeće rezerve nafte na bilo kojoj tački između A i B. Moguća rešenja mogu se predstaviti nizovima bitova dužine 10*, tj. brojevima od* 0 *do* 1023*. Tački A tada odgovara broj* 0 *i reprezentacija* 0000000000*, a tački B broj* 1023 *i reprezentacija* 1111111111*. Tački C na rastojanju* $1320m$ *od tačke A odgovara vrednost* $1023 \cdot (1320/5000) \approx 270$ *i reprezentacija* 0100001110*, a tački D na rastojanju* $3200m$ *od tačke A odgovara vrednost* $1023 \cdot (3200/5000) \approx 655$ *i reprezentacija* 1010001111*. Za vrednost funkcije prilagođenosti jedne tačke može se uzeti rezerva nafte izmerena u toj tački.*

6.2.2 Funkcija prilagođenosti

Funkcija prilagođenosti daje ocenu kvaliteta jedinke. Što je vrednost funkcije prilagođenosti za neku jedinku veća, to će biti veća i verovatnoća da se ta jedinka koristi za generisanje sledeće generacije. Očekuje se da, kroz generacije, ukupna prilagođenost bude sve bolja i bolja. Pogodan izbor funkcije prilagođenosti od izuzetne je važnosti za efikasnost algoritma.

Funkcija prilagođenosti treba da je definisana za sve moguće jedinke i da se relativno brzo izračunava. Sem ovih, ne postoje nikakvi opšti uslovi koje funkcija prilagođenosti treba da zadovoljava (na primer, da je diferencijabilna), mada je algoritam često efikasniji za funkcije koje zadovoljavaju neke specifične uslove. Često je pogodno da funkcija prilagođenosti bude nenegativna (na primer, zbog jednostavnijeg sprovođenja procesa selekcije, videti poglavlje 6.2.4).

Primer 6.2. *Potrebno je odrediti tačku maksimuma funkcije* $f(x)$ *na intervalu* $[a, b]$*. Funkcija je definisana za sve elemente datog intervala, ali nije nužno ni neprekidna, ni diferencijabilna. Genetski algoritam prirodno je primeniti tako da se za funkciju prilagođenosti koristi upravo funkcija* f*, a da se za reprezentaciju koristi binarna reprezentacija (na način opisan u poglavlju 6.2.1). Ukoliko je potrebno odrediti tačku minimuma funkcije* $f(x)$*, onda bi za funkciju prilagođenosti mogla da se koristi funkcija* $-f$*.*

Primer 6.3. *Potrebno je napraviti raspored časova za neki fakultet. Cilj je da raspored časova zadovoljava što više zadatih uslova koji se odnose na studentske grupe, nastavnike, raspoložive sale, itd. Genetski algoritam moguće je primeniti tako da se za funkciju prilagođenosti koristi broj zadovoljenih uslova. Ukoliko su neki uslovi važniji od drugih, u funkciji prilagođenosti oni mogu da imaju veće težinske faktore.*

Funkcija cilja i funkcija prilagođenosti ne moraju da budu iste. Na primer, funkcija prilagođenosti može jedinkama čija vrednost funkcije cilja prelazi neku granicu pridruživati vrednost 1, a ostalima vrednost 0. Tako se može pojednostaviti implementacija algoritma, ali se ovakve odluke moraju donositi oprezno kako ne bi došlo do smanjenja raznolikosti populacije. U nekim problemima, funkcija prilagođenosti je samo jednostavno transformisana funkcija cilja: na primer, ako se traži tačka maksimuma neke funkcije f koja ima i pozitivne i negativne vrednosti, za funkciju prilagođenosti, kako bi bila nenegativna, može se uzeti funkcija cilja uvećana za C, gde je C konstanta dovoljno velika da je taj zbir uvek nenegativan. Ponekad je teško ili nemoguće da funkcija prilagođenosti bude identična funkciji cilja (na primer, kada funkcija cilja ima negativne vrednosti ili nije zadata eksplicitno). Dodatno, funkcija cilja ne mora uvek biti eksplicitno zadata nekom matematičkom reprezentacijom, već samo implicitno ili nekakvim manje formalnim opisom (videti, na primer, funkciju cilja iz poglavlja 6.4.3 — ona nije zadata eksplicitno, već se njene vrednosti za konkretne instance dobijaju simulacijama).

6.2.3 Inicijalizacija

Populaciju jedinki jedne generacije, ukoliko se koristi binarna reprezentacija, čini skup nizova binarnih cifara. U toku rešavanja jednog problema, obično sve generacije imaju isti broj jedinki. Taj broj, veličina populacije, je parametar algoritma i on je obično nekoliko desetina, stotina ili, eventualno, nekoliko hiljada.

Proces inicijalizacije, tj. proces generisanja početne populacije, često je jednostavan. Najčešće se početna populacija generiše slučajno (tako da pokriva čitav prostor pretrage). Ukoliko se koristi binarna reprezentacija, jedinke početne generacije mogu se generisati kao slučajni brojevi u intervalu $[0, 2^n - 1]$, gde je n dužina hromozoma u izabranoj reprezentaciji. Dodatno, u početnu populaciju mogu biti dodate neke specifične jedinke (na primer, iz delova prostora pretrage za koje se veruje da sadrži optimalna rešenja) ili čitava početna populacija može biti generisana koristeći neki drugi optimizacioni metod. Kao što je rečeno u poglavlju 6.2.1, u nekim problemima mogu da postoje ograničenja za ispravne jedinke (na primer, jedinkama treba da odgovaraju legalni putevi u grafu) i njih onda treba uzeti u obzir pri generisanju inicijalne populacije.

6.2.4 Selekcija

Selekcija obezbeđuje čuvanje i prenošenje dobrih osobina populacije (tj. dobrog genetskog materijala) na sledeću generaciju. U svakoj generaciji, deo jedinki se izdvaja za reprodukciju i generisanje nove generacije. Izdvajanje jedinki koje će učestovavati u reprodukciji zasniva se na funkciji prilagođenosti i, generalno, prilagođenije jedinke imaju veću verovatnoću da imaju potomstvo. U najjednostavnijim pristupima biraju se jedinke sa najvećom vrednošću funkcije prilagođenosti. U drugim pristupima, jedinke se biraju slučajno, ali sa verovatnoćama koje su izvedene iz prilagođenosti, pri čemu je moguće da budu izabrane i neke lošije prilagođene jedinke (to može da pomogne u održavanju genetske raznolikosti i, dalje, u sprečavanju prerane konvergencije ka nekom lokalnom optimumu). Najpopularnije strategije selekcije su ruletska i turnirska selekcija.

Ruletska selekcija U ruletskoj selekciji (eng. *roulette wheel selection*), ako je $f(i)$ vrednost funkcije prilagođenosti za jedinku i, a N broj jedinki u populaciji, verovatnoća da će jedinka i biti izabrana da učestvuje u reprodukciji jednaka je

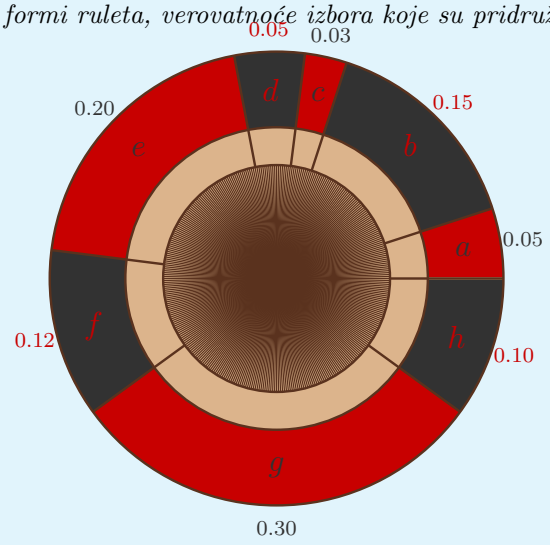
$$p_i = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$$

Naziv ruletske selekcije potiče od analogije koja se može napraviti sa ruletom. Ukoliko polja ruleta imaju širine proporcionalne verovatnoćama jedinki populacije, onda je proces biranja m jedinki za reprodukciju analogan odigravanju m partija ruleta.

Primer 6.4. *Pretpostavimo da populacija ima osam jedinki: a, b, c, d, e, f, g, h i da su njihove prilagođenosti redom* 0.10, 0.30, 0.06, 0.10, 0.40, 0.24, 0.60, 0.20. *Ukupna prilogođenost generacije jednaka je* 2.00. *Sledeća tabela prikazuje verovatnoće izbora jedinki u ruletskoj selekciji:*

jedinka	a	b	c	d	e	f	g	h
prilagođenost	0.10	0.30	0.06	0.10	0.40	0.24	0.60	0.20
verovatnoća	0.05	0.15	0.03	0.05	0.20	0.12	0.30	0.10

Naredna slika ilustruje, u formi ruleta, verovatnoće izbora koje su pridružene jedinkama.



U opisanom pristupu, pretpostavlja se da je funkcija prilagođenosti definisana tako da ima samo nenegativne vrednosti.

U ruletskoj selekciji moguće je da jedna jedinka više puta bude izabrana da učestvuje u reprodukciji. Prevelik broj ponavljanja istih jedinki loše utiče na performanse algoritma.

Turnirska selekcija U turnirskoj selekciji (eng. *tournament selection*), jedinke „odigravaju turnire“ u kojima veće šanse za pobedu (tj. za prelazak u narednu generaciju) imaju one sa boljom prilagođenošću.

Za jedan turnir bira se slučajno k jedinki iz populacije, gde je k parametar procesa turnirske selekcije. Nakon toga, u jednoj varijanti turnirske selekcije, pobednikom se smatra jedinka sa najvećom prilagođenošću. U drugoj varijanti, izabranih k jedinki sortira se po vrednosti funkcije prilagođenosti i i-ta jedinka u tako sortiranom nizu

bira se sa verovatnoćom[2] $p(1-p)^{i-1}$, gde je verovatnoća p drugi parametar procesa turnirske selekcije. Postoje i mnoge druge varijante turnirske selekcije.

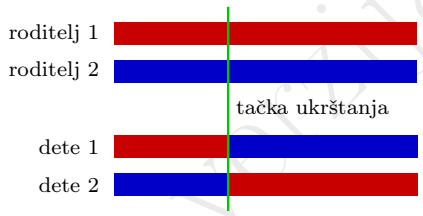
Ukoliko se u procesu selekcije koristi veća veličina turnira, onda nekvalitetne jedinke imaju manje šanse da budu izabrane. Selekcija sa veličinom turnira 1 ekvivalentna je slučajnoj selekciji. U determinističkoj turnirskoj selekciji ($p = 1$) bira se najbolja jedinka od onih koje učestvuju u turniru.

Jedinkama koje su jednom izabrane može se zabraniti učestvovanje u daljim turnirima.

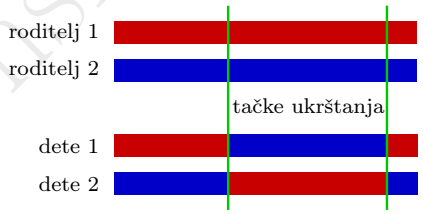
6.2.5 Reprodukcija

U procesu reprodukcije učestvuju jedinke koje su izabrane u procesu selekcije: biraju se dve različite i ukrštaju sa zadatom verovatnoćom (obično između 0.6 i 0.9). Dve jedinke koje učestvuju u ukrštanju (eng. *crossover*) nazivaju se roditelji. Rezultat ukrštanja je jedna nova jedinka ili dve nove jedinke koje nazivamo decom ili neposrednim potomcima. Očekivano je da deca nasleđuju osobine roditelja, pa otuda i nada da deca mogu da imaju prilagođenost kao svoji roditelji ili bolju (ako su nekako objedinila različite osobine koje daju dobru prilagođenost). Dobijeni potomci imaju šansu da uđu u sledeću generaciju, u skladu sa politikom zamene generacija.

Postoji više jednostavnih varijanti ukrštanja kada se koristi binarna reprezentacija. U *višepozicionom ukrštanju* potrebno je izabrati tačke ukrštanja i prekombinovati nizove binarnih cifara — jedno dete deo od jedne tačke ukrštanja do sledeće nasleđuje od jednog roditelja, a naredni deo od drugog. Ukrštanje može koristiti proizvoljan broj tačaka ukrštanja (s tim da je manji od dužine hromozoma). Slike 6.2 i 6.3 ilustruju ukrštanje sa jednom (jednopoziciono ukrštanje) i sa dve tačke ukrštanja (dvopoziciono ukrštanje) za binarnu reprezentaciju. Tačke ukrštanja biraju se slučajno iz skupa svih mogućih tačaka ukrštanja.



Slika 6.2: Jednopoziciono ukrštanje.



Slika 6.3: Dvopoziciono ukrštanje.

U *uniformnom ukrštanju* svaka binarna cifra prvog roditelja se sa verovatnoćom p prenosi na prvo dete i sa verovatnoćom $1 - p$ na drugo dete (a inače dete nasleđuje odgovarajuću binarnu cifru drugog roditelja). Verovatnoća p je obično jednaka 0.5, ali može biti i drugačija.

6.2.6 Mutacija

Mutacija se primenjuje nakon procesa ukrštanja. To je operator koji sa određenom (obično veoma malom) verovatnoćom menja jedan deo jedinke na određeni način. Operator mutacije može se realizovati na različite načine. Na primer, u binarnoj reprezentaciji mutacija menja jedan ili više slučajno odabranih gena. Od jedne jedinke dobija se jedna nova jedinka. Verovatnoća da će neka binarna cifra neke jedinke populacije biti promenjena je parametar algoritma i bira se eksperimentalno (a obično je manja od 1%).

Uloga mutacija u genetskim algoritmima je da spreči da jedinke u populaciji postanu suviše slične i da pomogne u obnavljanju izgubljenog genetskog materijala. Na primer, ukoliko u jednoj generaciji sve jedinke

[2] Zbir ovih verovatnoća za svaku konačnu vrednost k manji je od 1. Zato one treba da se skaliraju tako da im zbir bude jednak 1.

imaju istu vrednost jednog gena, onda taj gen samo ukrštanjem nikada ne bi mogao da se promeni. Kontrolisano podsticanje genetske raznolikosti mutacijom često omogućava izbegavanje lokalnih ekstremuma. Naime, često je dovoljno je da se jedna jedinka približi nekom lokalnom ekstremumu, pa da za nekoliko generacija sve jedinke budu u tom delu prostora pretrage. Mutacije omogućavaju pretraživanje novih delova prostora pretrage u nadi da će se naići na bolji lokalni ili na globalni ekstremum.

Ukoliko je verovatnoća mutacije velika, onda usmeravanje pretrage postaje preslabo i ona počinje da liči na slučajnu pretragu. Ukoliko je verovatnoća mutacije jednaka nuli, onda uopšte nema mutacije i algoritam će verovatno brzo dospeti do nekog lokalnog ekstremuma.

6.2.7 Politika zamene generacije

Politika zamene generacije opisuje kako se od tekuće generacije dobija nova. Osnovna podela po ovom kriterijumu je na *generacijske* genetske algoritme (eng. *generational genetic algorithm*) i genetske algoritme *stabilnog stanja* (eng. *steady state genetic algorithm*).

U slučaju generacijskih genetskih algoritama, nova generacija dobija se tako što se selekcijom bira dovoljno jedinki iz tekuće generacije da se napravi cela nova generacija. Izabrane jedinke se ukrštaju i mutiraju i tako dobijena generacija zamenjuje staru.

U slučaju genetskih algoritama stabilnog stanja, čim se izabere par roditelja, vrše se ukrštanje i mutacija i umetanje potomaka u populaciju u skladu sa nekom politikom zamene. Postoje raznovrsne politike zamene a neke od njih su:

- *zamena najgorih*, prema kojoj dobijeni potomci zamenjuju najmanje prilagođene jedinke u populaciji;
- *nasumična zamena*, prema kojoj dobijeni potomci zamenjuju nasumično izabrane jedinke iz populacije;
- *takmičenje roditelja i potomaka*, prema kojoj dobijeni potomci zamenjuju svoje roditelje ukoliko su oba potomka bolja od oba roditelja;
- *turnirska zamena*, prema kojoj se jedinka koju dobijeni potomci zamenjuju bira istim mehanizmom kao kod turnirske selekcije, s tim što se umesto najbolje prilagođenih jedinki biraju najgore.

Pored navedenih, za genetske algoritme stabilnog stanja, postoje i druge strategije zamene.

Elitizam je (opciona) strategija u okviru zamene generacije kojom se nekoliko najboljih jedinki (možda samo jedna) u generaciji štite od eliminisanja ili bilo kakvih izmena i takve prenose u sledeću generaciju. Ovim se eliminiše opasnost da se neka posebno kvalitetna jedinka izgubi tokom evolucionog procesa. Elitizam može da se koristi i u generacijskim politikama i u politikama stabilnog stanja.

6.2.8 Zaustavljanje

Genetski algoritam se izvršava, tj. evolucioni proces stvaranja novih generacija se ponavlja, sve dok nije zadovoljen neki uslov zaustavljanja. Najčešće se koriste sledeći uslovi zaustavljanja:

- pronađeno je rešenje koje zadovoljava unapred zadati kriterijum;
- dostignut je zadati broj generacija;
- funkcija prilagođenosti izračunata je zadati broj puta;
- vrednost prilagođenosti najbolje jedinke se tokom određenog broja generacija nije popravila;
- kombinacija nekoliko uslova.

6.3 Svojstva genetskih algoritama

Genetski algoritmi imaju širok domen i uspešno se primenjuju na velikom broju optimizacionih problema, često onih koji su NP-kompletni ili još teži. S druge strane, još uvek nema mnogo teorijskih rezultata koji govore o svojstvima genetskih algoritama, o kvalitetu rešenja koja daju, pa čak ni o tome zašto su genetski algoritmi uspešni. U daljem tekstu, biće reči o nekim dobrim i lošim stranama genetskih algoritama.

Ciljna funkcija. Ciljna funkcija može biti potpuno proizvoljna i ne mora da zadovoljava nikakve uslove (na primer, da bude neprekidna ili diferencijabilna). U praktičnim primenama, ciljna funkcija često nije zadata eksplicitno već implicitno. Na primer, cilj može biti da robot pređe što duži put, ali se ta dužina za konkretna podešavanja robota, zbog komplikovanog okruženja, može dobiti samo simulacijom, ne i primenom neke formule.

Reprezentacija jedinki, funkcija prilagođenosti i operatori. Pogodan izbor reprezentacije jedinki, funkcije prilagođenosti i operatora ukrštanja obično je ključan za performanse algoritma (za brzinu dolaženja do rešenja i za kvalitet rešenja). Ipak, za mnoge optimizacione probleme nije lako konstruisati pogodnu funkciju prilagođenosti jer se obično ne može unapred oceniti da li je nešto rešenje ili nije. U prvoj fazi rešavanja, reprezentacija jedinki, funkcija prilagođenosti i operatori se prilagođavaju problemu, a onda se vrši i prilagođavanje parametara algoritma, kao i dodatno fino podešavanje procesa rešavanja.

Parametri algoritma. Adekvatan izbor operatora ukrštanja i parametara genetskog algoritma (veličina populacije, verovatnoća ukrštanja, verovatnoća mutacije, itd) veoma je važan za njegove performanse. S druge strane, upravo velika sloboda u izboru parametara istovremeno je i pretnja da mogu da budu korišćeni parametri koji daju loše performanse. Optimizovanje parametara genetskog algoritma je kompleksan problem koji se najčešće rešava izvođenjem eksperimenata – probnih rešavanja. Za izbor adekvatnih parametara nekad se koriste sâmi genetski algoritmi. Parametri genetskog algoritma ne moraju biti fiksirani, već mogu da se menjaju i prilagođavaju tokom rada. Na primer, ukoliko su tekuće jedinke raznolike, onda se može povećati verovatnoća ukrštanja, a smanjiti verovatnoća mutacije. Nasuprot tome, ukoliko su tekuće jedinke međusobno slične, onda će im i potomci biti slični, te se može smanjiti verovatnoća ukrštanja, a povećati verovatnoća mutacije (kako bi se povećale šanse za bekstvo iz lokalnog optimuma).

Domen genetskih algoritama. Genetski algoritmi primenljivi su na veoma širok skup problema. Ipak, za uspešno rešavanje konkretnih problema potrebno je napraviti mnogo dobrih izbora (na primer, za funkciju prilagođenosti i za parametre).

Kvalitet rešenja. Genetski algoritam ne daje garanciju da je pronađeno rešenje globalno optimalno, pa čak ni da je lokalno optimalno. Međutim, i ako nađeno rešenje nije ni globalno ni lokalno optimalno, često je ono dovoljno dobro rešenje. Dodatno, kao rezultat algoritma može se ponuditi neki skup najboljih pronađenih jedinki, što je nekad pogodno. Takvo ponašanje je zadovoljavajuće, posebno u problemima za koje ne postoje tehnike koje garantuju pronalaženje optimalnog rešenja.

Zahtevani resursi. Genetski algoritmi se jednostavno implementiraju. Ipak, za najbolje rezultate često je potrebno implementaciju prilagoditi konkretnom problemu. Iako su algoritmi i implementacije obično jednostavni, izvršavanje genetskih algoritama često je veoma vremenski i memorijski zahtevno. Genetski algoritmi mogu se pogodno i efikasno paralelizovati.

6.4 Rešavanje problema primenom genetskih algoritama

U ovom poglavlju biće data tri konkretna, jednostavna, ali ilustrativna primera primene genetskih algoritama.

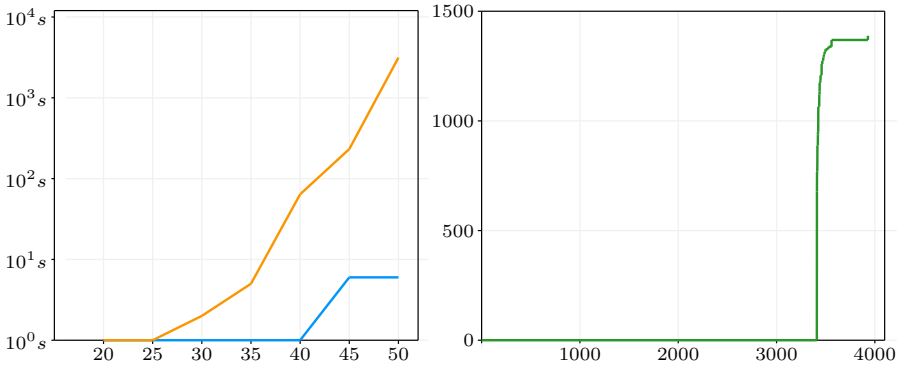
6.4.1 Rešavanje problema ranca

Genetski algoritmi često se koriste za rešavanje teških kombinatornih i optimizacionih problema. Jedan takav problem je dobro poznati problem ranca: dat je skup predmeta, svaki sa nekom poznatom vrednošću i poznatom masom; treba pronaći podskup predmeta koje treba staviti u dati ranac, tako da njihova masa ne pređe nosivost ranca, a da ukupna vrednost predmeta u rancu bude maksimalna moguća. Ovde ćemo razmatrati varijantu 0-1 problema ranca u kojoj nema više primeraka istog predmeta, te za svaki predmet postoje samo mogućnosti da je jedan takav u rancu ili da nijedan takav nije u rancu. Ovaj problem veoma je težak i trenutno nije poznato rešenje koje je polinomske složenosti u odnosu na dužinu ulaza (mada postoji rešenje polinomske složenosti u odnosu na nosivost ranca). Sa povećanjem dimenzije problema postojeći algoritmi brzo postaju praktično neupotrebljivi, te je ovaj problem zanimljiv za primenu genetskih algoritama.

Pretpostavlja se postojanje n predmeta, vrednosti $v_1, \ldots, v_n$ i masa $m_1, \ldots, m_n$. Na raspolaganju je ranac nosivosti M. Ako su $x_1, \ldots, x_n$ binarne promenljive koje označavaju da li je odgovarajući predmet stavljen u ranac ili ne i $\mathbf{x}$ binarni vektor čije su one koordinate, problem ranca može se definisati kao sledeći problem maksimizacije:

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{i=1}^{n} v_i x_i \quad \text{, pri čemu važi } \sum_{i=1}^{n} m_i x_i \leq M$$

Za rešavanje ovog problema primenom genetskog algoritma, potrebno je definisati njegove osnovne elemente. Za reprezentaciju rešenja, odnosno hromozom, prirodno se nameće sâm binarni vektor $\mathbf{x}$. Za funkciju prilagođenosti izabrali smo ukupnu vrednost izabranih predmeta ukoliko njihova ukupna masa ne premašuje nosivost ranca, a 0 u suprotnom. Za selekciju smo izabrali turnirsku, iako su mogući i drugačiji izbori. Za ukrštanje smo izabrali jednopoziciono ukrštanje. Mutacija se takođe vrši na uobičajen način. Veličina populacije jednaka je 1000, verovatnoća ukrštanja 0.8, a verovatnoća mutacije 0.01. Jedinke su inicijalizovane tako što su im svi bitovi



Slika 6.4: Na levoj slici prikazano je vreme izvršavanja egzaktnog algoritma (narandžasto) i genetskog algoritma (plavo) u odnosu na broj predmeta u problemu ranca (broj predmeta prikazan je na x osi, a vreme je prikazano na y osi, vremenska skala je logaritamska). Na desnoj slici prikazana je vrednost najboljeg rešenja kroz generacije do dostizanja optimalnog rešenja za dimenziju 50 (redni broj generacije prikazan je na x osi, vrednost najboljeg rešenja prikazana je na y osi).

nasumično generisani sa verovatnoćom 0.5 za oba ishoda. Politika zamene populacije je generacijska. Postupak se zaustavlja kada je dostignut zadati broj generacija. Navedeni izbor parametara i podešavanja algoritma načinjen je na osnovu isprobavanja (ne veoma sistematičnog u ovom slučaju) raznih mogućih vrednosti.

Kako bismo demonstrirali upotrebljivost genetskog algoritma, izvršili smo sledeći eksperiment. Nosivost ranca fiksirana je na 500. Razmatrano je nekoliko instanci problema ranca, koje su generisane nasumice: za svaku dimenziju problema n, vrednosti predmeta i njihove mase izabrane su nasumice iz intervala $[0, 100]$. Optimalno rešenje za svaku dimenziju pronađeno je veoma jednostavnim egzaktnim algoritmom (DFS sa bektrekingom), pri čemu je mereno i njegovo vreme izvršavanja. Za genetski algoritam mereno je vreme izvršavanja do pronalaženja optimalnog rešenja, u čemu je genetski uspeo u svakom od eksperimenata. Primetimo da za razliku od egzaktnog algoritma, genetski algoritam ne pruža garancije da će pronaći optimalno rešenje, niti može da prepozna da je dostigao optimalno rešenje (ako se to desi), te nema načina da se zaustavi tačno u tom trenutku kao u ovom eksperimentu (kada je optimalno rešenje bilo poznato unapred). Ipak, u ovom eksperimentu mereno je vreme do pronalaženja optimalnog rešenja, jer pruža neku informaciju o kvalitetu algoritma. Na slici 6.4 (levo), prikazana su pomenuta vremena u zavisnosti od dimenzije problema. Očigledno, u slučaju instanci problema veće dimenzije, genetski algoritam izvršava se drastično brže nego pomenuti egzaktni algoritam. Konkretno, za dimenziju 50, egzaktni algoritam izvršava se 3141 sekundu,[3] a genetski algoritam nalazi optimalno rešenje za svega 6 sekundi. Na istoj slici (desno) prikazan je i rast vrednosti najbolje pronađene jedinke kroz generacije za dimenziju 50. Grafik pokazuje da je većina vremena protekla pre pojavljivanja prve jedinke koja zadovoljava uslov nosivosti. Od njene pojave, optimizacija napreduje vrlo brzo. Ovo ponašanje lako je razumljivo. Za problem dimenzije 50, očekivana masa izbora koji definiše jedan nasumično generisani hromozom je 1250 (u proseku, polovina svih predmeta biće stavljena u ranac, a prosek njihovih masa je 50), što je značajno iznad ograničenja od 500, tako da nije verovatno da će u polaznoj populaciji biti jedinki koje zadovoljavaju ograničenje ukupne mase. S druge strane, u slučaju problema dimenzije 20, očekivana masa je 500, što odgovara ograničenju, pa je očekivano da će biti podjednako jedinki koje imaju i manju i veću masu. Promena inicijalizacije smanjenjem verovatnoće generisanja jedinica čini da optimizacija krene mnogo ranije i u slučaju problema dimenzije 50.

6.4.2 Vizuelizacija životnih navika i zdravstvenih stanja

Čak i uz duboku pažnju, razumevanje obimnih podataka često je teško i zamorno. Zbog toga se traga za pogodnim, sažetim načinima prezentovanja podataka. Na primer, matrica korelacije prikazuje zavisnost između različitih vrednosti: svako polje matrice sadrži vrednost *koeficijenta korelacije* između dve promenljive. Nećemo ulaziti u detalje ovog pojma i zadržaćemo se samo na njegovim osnovnim osobinama: koeficijent korelacije ima vrednosti između -1 i 1. Vrednost 1 znači da postoji savršena pozitivna zavisnost, vrednost -1 da postoji savršena negativna zavisnost, a vrednost 0 da nema zavisnosti.[4] Na primer, prisustvo gripa i povišena temperatura su visoko pozitivno korelisani. Pobede u međusobnim susretima dva košarkaška tima savršeno su negativno korelisane (ako je jedan tim pobedio, onda drugi tim nije pobedio). Među svim osobama rođenim 1980, visina i

[3] Naravno, ova razlika bila bi znatno manja u odnosu na neki napredni egzaktni algoritam za problem ranca.
[4] Preciznije bi bilo govoriti o linearnoj zavisnosti.

Navika ili stanje	1	2	3	4	5	6	7
1 Fizička aktivnost	1.00	−0.59	0.52	−0.68	−0.83	−0.80	−0.28
2 Pušenje		1.00	−0.57	0.68	0.60	0.65	0.53
3 Zdrava ishrana			1.00	−0.65	−0.28	−0.35	−0.23
4 Gojaznost				1.00	0.67	0.71	0.46
5 Dijabetes					1.00	0.87	0.48
6 Visok krvni pritisak						1.00	0.57
7 Povišen holesterol							1.00

Tabela 6.1: Tabela koreliranosti životnih navika i zdravstvenih stanja.

Nepoznata	c_1	c_2	c_3	c_4	c_5	c_6	c_7
Vrednost iz intervala $[0, 255]$	187	61	217	78	46	38	21
Ugao	262°	85°	305°	109°	64°	53°	29°

Tabela 6.2: Jedinka koja predstavlja najbolju vizuelizaciju dobijenu genetskim algoritmom. Za svaku promenljivu c_i, data je njena vrednost u hromozomu u rasponu od 0 do 255 i odgovarajući ugao na krugu.

mesec rođenja verovatno nisu korelisani.

Tabela 6.1 prikazuje korelisanost nekih životnih navika i zdravstvenih stanja (podaci su stvarni i odnose se na veliki uzorak stanovnika Sjedinjenih Američkih Država). Na primer, navika vežbanja pozitivno je korelisana sa navikom zdrave ishrane – koeficijent korelacije jednak je 0.52.

Navedena tabela sažeto prikazuje mnoštvo podataka koji se odnose na veliki broj osoba. Ali može se otići i korak dalje i, uz izvesna pojednostavljivanja, vizuelizovati njen sadržaj. Tako će suština tabele i njene glavne poruke biti još lakši za razumevanje. Sadržaj matrice korelacije može se vizuelizovati na sledeći način: svakoj promenljivoj biće dodeljena jedna tačka na krugu, tako da promenljive koje su jače pozitivno korelisane budu međusobno što bliže, a promenljive koje su jače negativno korelisane budu međusobno što dalje. U konkretnom slučaju, time se ilustruje koje životne navike obično idu jedna sa drugom, a koje ne.

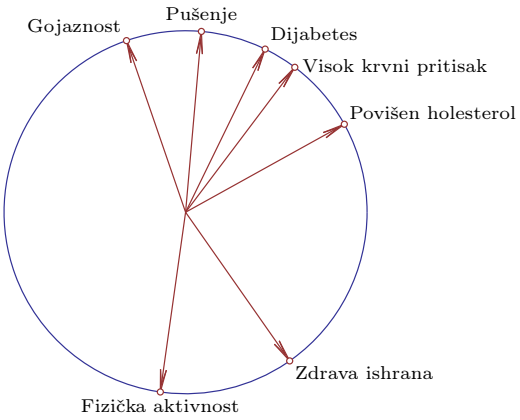
Precizirajmo sada način na koji to treba uraditi. Neka se razmatra n životnih navika i zdravstvenih stanja i neka su koeficijenti korelacije jednaki x_{ij}, za $1 \leq i, j \leq n$. Neka je c_i, za $1 \leq i \leq n$ pozicija pridružena promenljivoj i na krugu (to su nepoznate veličine koje tražimo) i neka su njene moguće vrednosti iz intervala $[0°, 360°)$. Neka je $\mathbf{c}$ vektor tih veličina. Smatraćemo da je bliskost na krugu dve promenljive jednaka kosinusu ugla između njih. Ukoliko su dve tačke na krugu jednake, onda je njihova bliskost jednaka $\cos 0° = 1$, a ukoliko su dijametralno suprotne onda je njihova bliskost jednaka $\cos 180° = -1$. Raspon vrednosti te mere jednak je rasponu vrednosti koeficijenta korelacije, pa je naš cilj da promenljive budu raspoređene na krugu tako da bliskost dve promenljive što bolje odgovara njihovom koeficijentu korelacije. Preciznije, problem se može definisati kao sledeći problem minimizacije (umesto kvadrata navedenih razlika mogle bi se koristiti njihove apsolutne vrednosti):

$$\min_{\mathbf{c}\in[0,360]^n} \sum_{i,j=1}^{n} (\cos(c_i - c_j) - x_{ij})^2$$

Ovaj problem može se aproksimativno rešiti primenom genetskih algoritama. U konkretnom primeru, postoji 7 promenljivih. Svaka može imati vrednosti iz intervala $[0°, 360°)$, ali taj interval možemo diskretizovati i zameniti intervalom celih brojeva $[0, 255]$ (pri čemu se 0° preslikava u 0, levi kraj intervala, a 360° u 255, desni kraj intervala).[5] Dakle, svaku od traženih vrednosti možemo predstaviti pomoću 8 bitova, pa će hromozom imati ukupno $7 \times 8 = 56$ bitova.

Genetski algoritam koristi populaciju veličine 100, verovatnoća ukrštanja je 0.6, a verovatnoća mutacije je 0.01. Korišćena je ruletska selekcija i generacijska politika zamene populacije. Algoritam se zaustavlja kada je dostignuto 10000 generacija. Jedinke su inicijalizovane generisanjem po 7 slučajnih brojeva u intervalu $[0, 255]$. Izbor parametara i podešavanja algoritma načinjen je nakon isprobavanja više mogućih vrednosti. Najbolje rešenje, sa vrednošću ciljne funkcije 1.23, dobijeno je u 5558. iteraciji. Najbolja jedinka data je u tabeli 6.2 i ilustrovana je slikom 6.5. Kako bismo se intuitivno uverili da ovakva ilustracija odražava informacije iz tabele korelacija, uočimo da su sve pozitivno korelisane promenljive raspoređene pod uglom manjim od 90°, a negativno korelisane pod uglom većim od 90°, osim zdrave ishrane i povišenog holesterola koji na slici deluju nekorelisano. Ipak, zanimljivo je da to i jeste najslabija korelacija u tabeli. Naravno, ne tvrdi se da je svaki zadati koeficijent

[5] Zbog preslikavanja poluotvorenog u zatvoreni interval, moguća vrednost polazne promenljive je i 360°. Ovakvo preslikavanje izabrano je zbog jednostavnosti, a moglo je da bude definisano i drugačije.



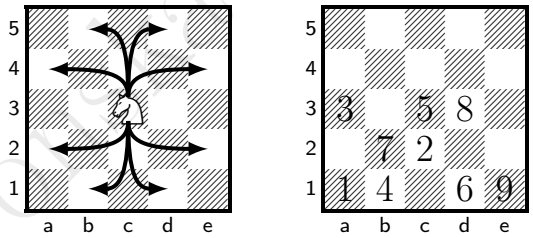
Slika 6.5: Vizuelizacija međusobnih zavisnosti životnih navika i zdravstvenih stanja.

korelacije između nekog para navika ili stanja jednak kosinusu odgovarajućeg dobijenog ugla, već samo da su svi koeficijenti korelacije zajedno aproksimirani izabranom reprezentacijom.

Primetimo da je u ovom problemu diskretizacijom smanjena šansa da algoritam nađe optimalno rešenje. To u ovom slučaju nije kritično, jer je glavni cilj bio intuitivno razumljiva vizuelizacija datih podataka.

6.4.3 Obilazak table skakačem

Problem obilaska table skakačem je problem pronalaženja kretanja skakača na šahovskoj tabli dimenzija $n \times n$, takvog da skakač poseti što veći broj različitih polja ali nijedno polje više puta.[6] U nastavku, pretpostavićemo da je tabla dimenzija 5×5 i da je skakač na početku u donjem levom uglu. Na slici 6.6, prikazana je jedna putanja od osam poteza koja se ne može nastaviti.



Slika 6.6: Kretanje skakača (levo) i jedna putanja skakača od osam poteza koja se ne može nastaviti (desno).

Za svako od 25 polja treba odrediti ono polje na koje je najbolje da skakač pređe. U zavisnosti od polja na kojem je, skakač može preći na dva do osam drugih polja, pa se izbor narednog polja uvek može kodirati pomoću tri binarne cifre, tj. pomoću tri bita. Skakač može napraviti najviše 24 poteza. Stoga, za potrebe primene genetskog algoritma, hromozom se može sastojati iz $24 \times 3 = 72$ bita, pri čemu svaka trojka odgovara jednom polju table i označava jedan od osam mogućih poteza sa tog polja. Očigledno, za neka polja neki od poteza koje hromozom može predstaviti neće biti legalna, ali ih svejedno dopuštamo.

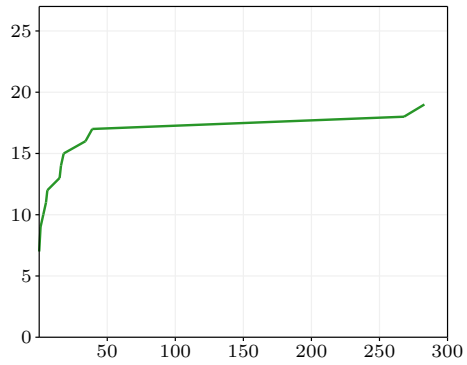
Za funkciju cilja i funkciju prilagođenosti prirodno se nameće broj skokova koje skakač može da izvede u skladu sa evaluiranim hromozomom, do skoka van table ili do skoka na već posećeno polje.

Koristi se generacijski genetski algoritam – u svakom koraku bira se dovoljno jedinki da se generiše cela nova populacija, a potom se vrše ukrštanja i mutacije. Selekcija je jednostavna ruletska, a ukrštanje sa jednom tačkom ukrštanja. Koristi se populacija od 3000 jedinki, verovatnoća ukrštanja 0.8 i verovatnoća mutacije po hromozomu 0.05. Algoritam se zaustavlja kada je dostignuto 1000 generacija. Izbor parametara i podešavanja algoritma načinjen je nakon isprobavanja više mogućih vrednosti.

Kako ponašanje genetskog algoritma može značajno zavisiti od polazne populacije koja se slučajno generiše, može se desiti da se u različitim pokretanjima dobije različit kvalitet rešenja. Zbog toga je, u ovom primeru,

[6] Strožije postavljen problem je problem pronalaženja kretanja skakača na šahovskoj tabli kojim skakač po jednom posećuje svako polje table.

rešavanje bilo vršeno 10 puta. Potom su izračunati prosečan kvalitet najboljeg rešenja i prosečna generacija u kojoj je ono nađeno. Prosečna dužina pronađene putanje je 19.1. Prosečan broj iteracija koje su bile potrebne za dostizanje najboljeg pronađenog rešenja je 326.7. Na slici 6.7, prikazana je zavisnost dužine pređene putanje za najbolju jedinku u odnosu na broj iteracija u jednom izvršavanju genetskog algoritma. Nije pronađena putanja koja obilazi sva polja. Razlog za to je što je taj problem previše težak za pravolinijski pristup koji je upotrebljen. Za njegovo puno rešavanje potrebne su dodatne, napredne tehnike. Ovaj primer ilustruje i kako genetski algoritmi u nekim problemima ne dovode nužno do optimalnog rešenja.



Slika 6.7: Zavisnost kvaliteta najbolje jedinke u populaciji od broja generacija genetskog algoritma (redni broj generacije prikazan je na x osi, vrednost najboljeg rešenja prikazana je na y osi).

Deo II

Automatsko rasuđivanje

Glava 7

Rešavanje problema korišćenjem automatskog rasuđivanja

Automatsko rasuđivanje (ili automatsko rezonovanje), bavi se razvojem algoritama i programa koji mogu da rasuđuju automatski, koristeći metode matematičko-deduktivnog zaključivanja u nekim konkretnim logičkim okvirima. Mnoge podoblasti veštačke inteligencije oslanjaju se na matematičku logiku i na automatsko rasuđivanje. Ove tehnike koristimo kada je potrebno netrivijalno rigorozno zaključivanje. Tu spadaju i problemi u okviru kojih je potrebno dokazati da neko tvrđenje važi za sve moguće objekte koji zadovoljavaju neke preduslove, kao i problemi u okviru kojih je potrebno dokazati da postoje objekti za koje neko tvrđenje važi.

U algoritmima za logičko zaključivanje često je neki korak zaključivanja moguće sprovesti na više različitih načina, ali nije precizirano koji od tih načina treba da se izabere. Naime, bez obzira na načinjeni izbor, izvedeni zaključci su uvek ispravni. Međutim, neki putevi do istog zaključka mogu da budu znatno kraći od drugih i tada je proces automatskog rasuđivanja znatno efikasniji. Ovo pokazuje da je i u logičkom rasuđivanju jedan od centralnih problema problem usmeravanja pretrage.

Logičkih okvira ima mnogo i pojedinačno su pogodni za opisivanje raznovrsnih teorija i praktičnih problema: iskazna logika, logika prvog reda, logika višeg reda, fazi logika, modalna logika, temporalna logika, deontičke logike, itd. Zbog ograničenosti prostora, mi ćemo se u nastavku baviti samo iskaznom logikom i logikom prvog reda.

Primer 7.1. *Popularna igra „sudoku“ (za jednog igrača) igra se na sledeći način. Data je tabela koja se sastoji od devet kolona i devet vrsta. Polja su dodatno grupisana u 9 kvadrata od po 9 polja. Na početku neka polja sadrže brojeve i zadatak je popuniti čitavu tabelu tako da važe sledeći uslovi:*

- *Svako polje sadrži neki broj između 1 i 9;*
- *Svaka vrsta sadrži sve brojeve između 1 i 9;*
- *Svaka kolona sadrži sve brojeve između 1 i 9;*
- *Svaki mali kvadrat sadrži sve brojeve između 1 i 9.*

Iz navedenih uslova jednostavno sledi da ni u jednoj koloni, vrsti niti malom kvadratu ne postoji broj koji se pojavljuje dva puta. Naredna slika prikazuje jedan rešeni sudoku (krupnijim ciframa prikazani su unapred zadati brojevi, sitnijim brojevi koji čine rešenje).

8	2	7	1	5	4	3	9	6
9	6	5	3	2	7	1	4	8
3	4	1	6	8	9	7	5	2
5	9	3	4	6	8	2	7	1
4	7	2	5	1	3	6	8	9
6	1	8	9	7	2	4	3	5
7	8	6	2	3	5	9	1	4
1	5	4	7	9	6	8	2	3
2	3	9	8	4	1	5	6	7

Kako svako polje ima jednu od vrednosti iz konačnog skupa, ovaj zadatak u principu može biti rešen sistematičnim ispitivanjem svih mogućnosti. Ipak, broj svih mogućnosti toliko je veliki (maksimalno 9^{81}), da to rešenje nije praktično primenljivo ni za računar, a kamoli za čoveka. Zato rešavanje zahteva neko netrivijalno rasuđivanje. U toku procesa rešavanja, za neka polja može se jednoznačno odrediti vrednost, tj. može se zaključiti, dokazati *da nema drugih mogućnosti. Slično, za neka polja broj mogućnosti može da se svede samo na dve ili samo na tri, itd. Takvim zaključivanjem prostor pretrage značajno se smanjuje i uspešno rešavanje postaje moguće. Sredstva automatskog rasuđivanja mogu da se koriste za donošenje pojedinačnih odluka u fazi rešavanja ili za rešavanje celokupnog problema.*

Primer 7.2. *Zna se da je otac neke osobe predak te osobe. Zna se i da je predak pretka neke osobe i sâm predak te osobe. Potrebno je dokazati da je otac Aninog oca Anin predak.*

Primetimo da se u ovom zadatku govori o porodičnim odnosima nad skupom ljudi koji nije konačan, te je potrebno rasuđivanje koje suštinski prevazilazi ispitivanje nekog konačnog skupa mogućnosti (iako se nekad može svesti na to).

Primer 7.3. *Pretpostavimo da je pitanje da li može doći do greške u izvršavanju narednog koda:*

```
int f(int y)
{
    int i, x = 1;
    for (int i = 0; i < y; i++)
        x += 2*y;
    return 1000 / x;
}
```

Jedina naredba u kojoj može doći do greške u fazi izvršavanja je naredba `return`*, u okviru koje se računa vrednost izraza primenom operacije deljenja. Do greške može doći ako je* `x` *jednako 0. Može se dokazati da* `x` *ni u kom slučaju ne može biti jednako 0 i to zahteva neko netrivijalno rasuđivanje. Postoje algoritmi koji omogućavaju automatsko dokazivanje ovakvih tvrđenja.*

Primeri primena automatskog rasuđivanja. Tehnike automatskog rasuđivanja koriste se u razvoju matematičkih teorija, u problemima raspoređivanja, u verifikaciji softvera i hardvera i drugde.

Raspored utakmica za špansku fudbalsku ligu, više godina pravljen je korišćenjem namenskih programa za rasuđivanje u iskaznoj logici. I mnogi drugi problemi raspoređivanja rešavani su na sličan način.

Godine 1996. program Otter/EQP zasnovan na metodu rezolucije, nakon osam dana rada, automatski je dokazao hipotezu otvorenu oko pedeset godina ranije i do tada nedokazanu: svaka Robinsova algebra je Bulova. Tu vest preneli su tada gotovo svi svetski mediji kao vest o značajnoj prekretnici – polje matematičkih otkrića, rezervisano do tada samo za ljude, počeli su da osvajaju i računari.

Formalne metode su oblast računarstva koja se bavi specifikovanjem, razvojem i verifikacijom softverskih i hardverskih sistema. U okviru ovih metoda, analize zasnovane na matematičkoj logici imaju zadatak da obezbede pouzdanost sistema koji se razvijaju. Ključnu ulogu u formalnim metodama imaju logika i automatsko

rasuđivanje. Sistem kontrola leta u Ujedinjenom Kraljevstvu i linije pariskog metroa bez vozača, na primer, razvijeni su i verifikovani formalnim metodama.

Namenski programi za rasuđivanje u specijalizovanim teorijama kao što je, na primer, linearna aritmetika, zovu se SMT rešavači i intenzivno se koriste u verifikaciji sofvera i hardvera – testiranju i dokazivanju njegove ispravnosti, ali i u sintezi programa. Jedan od najznačajnijih rešavača je Z3, kompanije Microsoft.

Faze rešavanja problema korišćenjem automatskog rasuđivanja. Tôk rešavanja problema korišćenjem automatskog rasuđivanja često obuhvata sledeće osnovne faze:

- modelovanje problema;
- rešavanje problema opisanog u matematičkim terminima;
- interpretiranje i analiza rešenja.

U navedeni opšti okvir ulaze problemi koji se formulišu tako da ih neki pogodan rešavač/dokazivač može rešiti. Navedene faze treba shvatiti uslovno, jer postoje i problemi čije rešavanje može da se razlikuje. Na primer, zadatak može biti i razvoj samog rešavača/dokazivača za neku klasu logičkih problema i tada proces rešavanja može da bude drugačiji.

7.1 Modelovanje problema

Modelovanje problema predstavlja formulisanje problema precizno, u matematičkim terminima, korišćenjem pogodnog logičkog okvira.

Potrebno je najpre utvrditi vrstu osnovnih objekata i nepoznatih veličina u problemu. Na primer, to mogu biti jednostavni iskazi (poznati i nepoznati), celi brojevi i slično. Onda treba opisati uslove koji moraju da važe za objekte opisane u problemu.

U nekom logičkom okviru ne mogu da se izraze neki uslovi, pa u fazi modelovanja biramo i pogodni logički okvir, na primer – iskaznu logiku (glava 8) ili logiku prvog reda (glava 9). Dalje, u okviru kao što je logika prvog reda, mogu se opisati teorije pogodne za opisivanje raznovrsnih struktura i njihovih svojstava. Postoje, na primer, teorije koje opisuju realne brojeve, teorije koje opisuju cele brojeve fiksne širine i tako dalje.

Primer 7.4. *Šef protokola na jednom dvoru treba da organizuje bal za predstavnike ambasada. Kralj traži da na bal bude pozvan Peru ili da ne bude pozvan Katar (Qatar). Kraljica zahteva da budu pozvani Katar ili Rumunija (ili i Katar i Rumunija). Princ zahteva da ne bude pozvan Peru ili da ne bude pozvana Rumunija (ili da ne budu pozvani ni Peru ni Rumunija). Da li je moguće organizovati bal tako da su zadovoljeni zahtevi svih članova kraljevske porodice?*

Navedeni problem potrebno je najpre modelovati na neki precizan način. Iskaz, tvrdnju „na bal će doći ambasador Perua“ označićemo sa p, iskaz „na bal će doći ambasador Katara“ označićemo sa q, a iskaz „na bal će doći ambasador Rumunije“ sa r. Uslov koji postavlja kralj, onda glasi „važi p ili ne važi q“ ili kraće zapisano „p ili ne q“. Uslov koji postavlja kraljica glasi „q ili r“. Uslov koji postavlja princ glasi „ne p ili ne r“.

Sva navedena ograničenja, svi ovi iskazi, zajedno čine novi, komplikovaniji iskaz koji možemo da zapišemo na sledeći način:

„(p ili ne q) i (q ili r) i (ne p ili ne r)“

Ovaj složeni iskaz predstavlja precizan zapis problema. Potrebno je proveriti da li polazni iskazi p, q i r mogu da imaju konkretne vrednosti tačno *ili* netačno *takve da složeni iskaz ima vrednost* tačno. *Da bi se taj problem rešio potrebno je precizno definisati i na koji način se složenim iskazima pridružuje vrednost* tačno *ili* netačno *ukoliko je poznato koje vrednosti su pridružene polaznim iskazima.*

Primer 7.5. *U primeru 7.1, osnovnim objektima mogu se smatrati celi brojevi koji čine vrednosti polja tabele. Na primer, promenljiva x_{12} može da označava vrednost polja $(1, 2)$.*

Osnovni iskazi mogu da budu iskazi koji opisuju sadržaj pojedinačnih polja. Jedan takav iskaz je „polje $(1, 2)$ sadrži vrednost 5“ ili, kraće, „$x_{12} = 5$“. Osnovni iskazi mogu da budu i „$x_{12} \neq 3$“, „$1 \leq x_{12}$“, „$x_{12} \leq 9$“ i slično. Na ovaj način, možemo uvesti veliki broj osnovnih iskaza od kojih će u konačnom rešenju neki biti tačni a neki netačni.

Problem opisuju uslovi (tj. ograničenja) koji moraju da važe nad osnovnim iskazima, poput „ako važi

$x_{23} = 4$, *onda važi* $x_{24} \neq 4$*“. Kao polazne pretpostavke treba uvesti iskaze o zadatim poljima, na primer* „$x_{19} = 2$*“.*

Primer 7.6. *Date su sledeće tvrdnje: „svaki čovek je smrtan“, „Sokrat je čovek“. Pitanje je da li se iz ovih tvrdnji može utvrditi da je tačno i „Sokrat je smrtan“.*

Tvrdnju da je x čovek zapišimo kao „x je čovek“ a tvrdnju da je x smrtan zapišimo kao „x je smrtan“. Tvrdnju „svaki čovek je smrtan“ zapišimo „za svako x važi: ako (x je čovek) onda (x je smrtan)“. Zadato tvrđenje onda (pomalo rogobatno u odnosu na svakodnevni jezik) glasi: „ako (za svako x važi: ako (x je čovek) onda (x je smrtan)) i (Sokrat je čovek) onda (Sokrat je smrtan)“.

Ključna razlika u odnosu na primer 7.4 je to što navedeni iskazi (pa i složene tvrdnje) zavise od nekog parametra x. To sugeriše da će logički okvir potreban za rešavanje ovog problema morati da bude izražajniji od logičkog okvira za primer 7.4.

Prethodne tvrdnje još uvek nisu zapisana u preciznim logičkim terminima ali to i neće biti urađeno u ovom primeru.

Primer 7.7. *U primeru 7.2, osobe (koje nisu konkretne) možemo označiti latiničnim slovima x, y, z, ... Ana je konkretna osoba, pa ćemo je označiti baš tako. Zadata znanja o porodičnim odnosima možemo onda da formulišemo na sledeći način:*

- *„za proizvoljne tri osobe x, y, z važi: ako je x predak od y i y je predak od z, onda je x predak od z“*
- *„za proizvoljnu osobu x važi: otac osobe x je predak osobe x“*

dok je tvrđenje koje treba dokazati:

- *„otac oca osobe Ana je predak osobe Ana.“*

Prethodna formulacija još nije zapisana u preciznim logičkim terminima i stoga modelovanje još nije završeno. Jedan formalni jezik u kojem se prethodne tvrdnje mogu zapisati je jezik logike prvog reda (koji će tek kasnije biti precizno uveden). Neka $predak(x, y)$ označava tvrdnju da je osoba x predak osobe y (koja može biti tačna ili netačna za konkretne osobe) i neka $otac(x)$ označava oca osobe x. Ukoliko $\Rightarrow$ označava implikaciju, $\wedge$ konjunkciju, a univerzalni kvantifikator $\forall x$ označava da prateća tvrdnja važi za svako x, onda se prethodna pravila mogu na jeziku logike prvog reda zapisati na sledeći način:

- $\forall x \forall y \forall z(predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z))$
- $\forall u(predak(otac(u), u))$

a tvrđenje koje treba dokazati:

- $predak(otac(otac(Ana)), Ana)$

Primer 7.8. *U primeru 7.3, programske promenljive možemo modelovati celim brojevima ili brojevima fiksne širine, takozvanim bitvektorima.*

Neka programska promenljiva `x` *bude modelovana promenljivom x koja ima celobrojne vrednosti. Preciznije, kako se vrednosti programske promenljive* `x` *menjaju tokom izvršavanja programa, njoj neće odgovarati jedna promenljiva x, nego niz promenljivih $x_0, x_1, x_2, x_3, \ldots$ Slično, neka programskoj promenljivoj* `y` *odgovara promenljiva y koja ima celobrojne vrednosti (*`y` *se ne menja tokom izvršavanja programa, pa možemo da je modelujemo jednom promenljivom). Važe sledeći uslovi: „$x_0 = 1$”, „$x_{i+1} = x_i + 2y$”, za svako $i \geq 0$. Ukoliko za modelovanje koristimo cele brojeve, a želimo da rad datog programa modelujemo precizno, u navedenim uslovima trebalo bi dodati računanje po modulu (koje odgovara izračunavanju na računaru).*

Odsustvo greške u navedenom programu modelujemo uslovom „izraz `1000 / x` *je u datom programu uvek definisan“, tj. uslovom „promenljiva* `x` *nikad nema vrednost 0“. U terminima uvedenih promenljivih ovaj uslov glasi: „$x_i = 0$“ ne važi ni za jednu vrednost $i \geq 0$.*

7.2 Rešavanje problema

U fazi rešavanja, traži se rešenje matematički formulisanog problema, korišćenjem pogodnih metoda zaključivanja. Metode za automatsko rasuđivanje (za logički formulisane probleme) razvijaju se uspešno već više od šezdeset godina.

Primer 7.9. *Ispitivanje da li, pod nekim uslovima, složeni iskaz iz primera 7.4 može biti tačan, može se sprovesti tako što bi bile ispitane vrednosti složenog iskaza za sve moguće vrednosti pridružene iskazima p, q i r. Tih iskaza ima tri, za svaki postoje dve mogućnosti – tačno i netačno, pa ukupno ima 2^3 mogućnosti koje treba ispitati.*

Primer 7.10. *Ako se za rešavanje sudokua koriste iskazi oblika „$x_{ij} = k$“ (primer 7.10), onda postoji ukupno $81 \cdot 9$ takvih iskaza i $2^{81 \cdot 9}$ mogućnosti koje treba ispitati. Međutim, umesto takvog naivnog pristupa, u praksi se koriste pristupi koji slične analize sprovode daleko efikasnije. Na primer, ako smo pretpostavili da je p tačno, onda je čitav uslov „p ili ne q“ tačan i nema potrebe da razmatramo mogućnosti za q. Slično, ako smo pretpostavili da je p netačno, onda je i čitav uslov „p i q“ netačan, nezavisno od vrednosti q.*

Primer 7.11. *Ispitivanje da li je tačno tvrđenje iz primera 7.6 može se sprovesti na sledeći način: pošto za svako x važi „ako (x je čovek) onda (x je smrtan)“, važi i kada je x upravo Sokrat, tj. važi „ako (Sokrat je čovek) onda (Sokrat je smrtan)“. Odatle i iz „Sokrat je čovek“, primenom pravila* modus ponens *(iz A i $A \Rightarrow B$ sledi B) sledi „Sokrat je smrtan“, pa važi dato tvrđenje.*

Primer 7.12. *U primeru 7.7 određena znanja o porodičnim odnosima zapisana su u vidu tvrđenje logike prvog reda, a pored njih zapisano je i tvrđenje koje je potrebno dokazati. Postoji mnoštvo tehnika kojima se može dokazati da je neko tvrđenje B posledica skupa tvrđenja $\{A_1, \dots A_n\}$. U ovom primeru ilustrovaćemo površno tehniku koja će kasnije biti detaljno objašnjena a u kojoj se koristi svođenje na kontradikciju: dokazuje se da je skup tvrđenja $\{A_1, \dots A_n, \neg B\}$ kontradiktoran tj. protivrečan (odakle sledi polazno tvrđenje). U konkretnom slučaju, potrebno je dokazati da je kontradiktoran sledeći skup tvrđenja:*

$$\{\forall x \forall y \forall z (predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z)) \,, \\ \forall x (predak(otac(u), u)) \,,\ \neg predak(otac(otac(Ana)), Ana)\}$$

Smatrajući da se univerzalna kvantifikacija podrazumeva, ovaj skup može da se zapiše kraće:

$$\{predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z), \\ predak(otac(u), u), \neg predak(otac(otac(Ana)), Ana)\}$$

Potrebno je otkriti protivrečnost među navedenim formulama ili nekim njihovim posledicama (i to na način koji se može automatizovati). Kako u samom skupu ne uočavamo protivrečnost, potrebno je iz njega izvoditi posledice dok protivrečnost ne bude očigledna. Razmotrimo šta se može zaključiti iz datih tvrđenja, na primer, iz $predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z)$ i $predak(otac(u), u)$. U prvom tvrđenju, podtvrđenja sa leve implikacije imaju sličnu formu kao drugo tvrđenje. Kako prvo tvrđenje govori o svim mogućim x i y, onda ono važi i za specifične vrednosti $otac(u)$ i u, tim redom. Ako primenimo zamene $x \mapsto otac(u)$ i $y \mapsto u$, prvo tvrđenje postaje $predak(otac(u), u) \wedge predak(u, z) \Rightarrow predak(otac(u), z)$ te je njegova prva „pretpostavka“ (tj. prvo podtvrđenje) ujednačena sa drugim tvrđenjem $predak(otac(u), u)$. Zbog toga se može smatrati da je prva pretpostavka prvog tvrđenja ispunjena i da se dalje može razmatrati i jednostavniji ostatak $predak(u, z) \Rightarrow predak(otac(u), z)$. Ovo rasuđivanje može se zapisati u vidu izvođenja:

$$\frac{predak(otac(u), u) \quad predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z)}{predak(u, z) \Rightarrow predak(otac(u), z)} \; x \mapsto otac(u), y \mapsto u$$

Dodatno, promenljive su i u novoizvedenom tvrđenju implicitno univerzalno kvantifikovane, pa se vrši njihovo preimenovanje kako ne bi došlo do slučajnih podudaranja sa starim promenljivim, a što bi moglo da promeni smisao skupa tvrđenja. Dakle, novoizvedeno tvrđenje je:

$$predak(u', z') \Rightarrow predak(otac(u'), z')$$

Novoizvedena tvrđenja mogu se koristiti u daljem zaključivanju ravnopravno sa starim, pa se može primeniti i sledeće izvođenje:

$$\frac{predak(otac(u),u) \quad predak(u',z') \Rightarrow predak(otac(u'),z')}{predak(otac(otac(u)),u)}\ u' \mapsto otac(u), z' \mapsto u$$

što, nakon preimenovanja, daje:

$$predak(otac(otac(u'')),u'')$$

Konačno, izvođenje

$$\frac{predak(otac(otac(u'')),u'') \quad \neg predak(otac(otac(Ana)),Ana)}{\bot}\ u'' \mapsto Ana$$

daje kontradikciju, čime je pokazano da je poćetni skup tvrđenja protivrečan. U kasnijem tekstu, pokazaće se da su svi navedeni koraci zaključivanja specijalni slučajevi jednog opšteg pravila – pravila rezolucije, koje čini osnovu značajnog metoda rezolucije. Kako je iz skupa formula

$$\{predak(x,y) \wedge predak(y,z) \Rightarrow predak(x,z), predak(otac(u),u),$$

$$\neg predak(otac(otac(Ana)),Ana)\}$$

izvedena kontradikcija, to znači da je pretpostavka $\neg predak(otac(otac(Ana)),Ana)$ bila pogrešna, te je dokazano da tvrđenja

- $\forall x \forall y \forall z(predak(x,y) \wedge predak(y,z) \Rightarrow predak(x,z))$
- $\forall u(predak(otac(u),u))$

povlače tvrđenje:

- $predak(otac(otac(Ana)),Ana)$

Primer 7.13. *U primeru 7.3, potrebno je dokazati da iz uslova „$x_0 = 1$" i „$x_{i+1} = x_i + 2y$", za svako $i \geq 0$ sledi da „$x_i = 0$" ne važi ni za jednu vrednost $i \geq 0$. Navedeno tvrđenje možemo dokazati za bilo koju konkretnu vrednost i, višestrukom primenom tvrđenja „ako je x_i neparan broj, onda je i $x_{i+1} = x_i + 2y$ neparan broj". To se može dokazati u teoriji koja opisuje cele brojeve ili cele brojeve fiksne širine (i sabiranje nad njima). Međutim, ukoliko želimo da dokažemo da „$x_i = 0$" ne važi ni za jednu vrednost $i \geq 0$, onda naš logički okvir treba da uključuje i princip matematičke indukcije. Postoje algoritmi za automatsko rasuđivanje i za takve logičke okvire.*

Navedeni procesi zaključivanja opisani su neformalno i grubo, ali služe kao motivacija za stroga pravila zaključivanja u iskaznoj logici i logici prvog reda i njihovo automatizovanje.

7.3 Interpretiranje i analiza rešenja

Dobijeno rešenje matematički formulisanog problema potrebno je formulisati u terminima početnog problema i potrebno je razumeti svojstva dobijenog rešenja. Na primer, ako se rešavanjem primera 7.4 dobije da p mora biti tačno, to znači da na bal treba da dođe ambasador Perua.

Zaključci koji se dobijaju procesom rešavanja su nesumnjivi, oni ne mogu biti pogrešni (sem ako je neispravan program koji sprovodi rasuđivanje). U tom smislu nikakva analiza, niti evaluacija tačnosti dobijenih zaključaka nije potrebna. Međutim, često su potrebni dodatna analiza i objašnjenje dobijenog rešenja u zavisnosti od izabranog modelovanja. Na primer, dva različita načina modelovanja opisana u primeru 7.8 nisu ekvivalentna: zaista, kod celih brojeva nema prekoračenja, a kod celih brojeva fiksne širine – ima. Rasuđivanje u domenu celih brojeva može da bude efikasnije, ali je rasuđivanje u domenu brojeva fiksne širine preciznije i vernije opisuje programske promenljive. Dakle, ukoliko smo se odlučili za modelovanje koje omogućava efikasno automatsko zaključivanje, ali ne modeluje zadati problem verno, treba znati za koje instance problema se rezultat faze rešavanja odnosi neposredno na polazni zadatak.

Pored navedenog, nekad je potrebno tumačiti i neuspeh faze rešavanja. Ako neki sistem ne uspe da dokaže tvrđenje S, u nekim slučajevima to može da znači da S zaista nije tačno, a u nekim slučajevima može da bude posledica činjenice da korišćeni sistem nije u stanju da dokaže sva moguća tačna tvrđenja (dakle, moguće je da

je S tačno, ali naš sistem to ne može da dokaže). Nekada se ne može razlučiti (čak i ako poznajemo algoritam rasuđivanja) između ta dva: nekada ako sistem prijavi da nije dokazao S, to još ne znači da S nije tačno.

Najvažnija opšta svojstva koje algoritmi za rasuđivanje mogu da imaju su sledeća:

Potpunost je svojstvo koje kaže da je algoritam u stanju da dokaže svako tvrđenje iz svog domena koje je dokazivo (pre ili kasnije, nekada je za to potrebno veoma dugo vreme).

Saglasnost je svojstvo koje kaže da ako algoritam tvrdi da je neko tvrđenje (iz njegovog domena) tačno, onda ono zaiste jeste tačno.

Potpunost je svakako poželjna osobina, a saglasnost je neophodna (kako bi zaključci koje donose algoritmi bili nesumnjivi).

Moć nekih algoritama logičkog rasuđivanja ograničena je sâmom prirodom problema koji se rešavaju. Za *problem odlučivanja*, tj. za problem koji zahteva odgovor „da“ ili „ne“, kažemo da je *odlučiv* ako postoji algoritam koji može da reši svaku njegovu instancu, a algoritam sa takvim svojstvom zovemo *procedura odlučivanja*. Na primer, neki problem raspoređivanja je *odlučiv* ako postoji algoritam koji za svaku instancu tog problema može da utvrdi da li traženi raspored postoji ili ne. I mnogi problemi rasuđivanja su problemi odlučivanja, pa se ove definicije odnose i na njih. Na primer, odlučiv je problem SAT (eng. *satisfiability problem*) – problem ispitivanja iskazne zadovoljivosti. Mnogi važni problemi logičkog rasuđivanja nisu odlučivi, tj. oni su *neodlučivi*. Za takve probleme moramo biti spremni da nećemo moći da dobijemo rešenje za proizvoljnu ulaznu instancu. Moguće je razviti samo algoritam koji primenom nekakvih heuristika može da reši neke instance problema. Postoje i *poluodlučivi problemi*: oni nisu odlučivi ali za njih mogu da postoje algoritmi koji uspešno mogu da reše problem za sve instance za koje je odgovor „da“. Takav algoritam zovemo *procedura poluodlučivanja*. Za instance za koje je ispravan odgovor „ne“, procedura poluodlučivanja vraća odgovor „ne“ ili se ne zaustavlja. Poluodlučiv je, na primer, halting problem – problem ispitivanja da li se dati program zaustavlja za datu ulaznu vrednost. I mnogi važni problemi logičkog rasuđivanja su poluodlučivi. Na primer, poluodlučiv je problem ispitivanja valjanosti u logici prvog reda.

Pored navedenih pitanja, za algoritme rasuđivanja, kao i za skoro sve druge vrste algoritama važna su pitanja vremenske i prostorne složenosti (najgoreg i prosečnog slučaja).

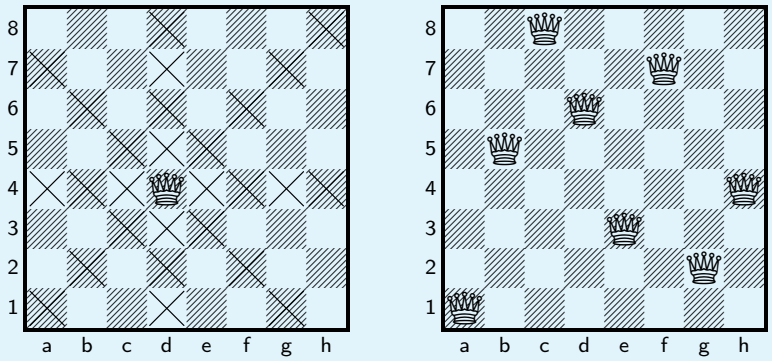
Glava 8

Automatsko rasuđivanje u iskaznoj logici

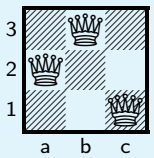
U iskaznoj logici (eng. *propositional logic*) kao osnovni objekti razmatraju se jednostavni, elementarni iskazi, tvrdnje, tvrđenja (na primer „Ana je zubar“). Iskazi mogu biti kombinovani u složenije iskaze logičkim veznicima (na primer „Ana je zubar i Ana živi u Beogradu“). Iskazna logika ima svoju sintaksu (koja opisuje njen jezik) i svoju semantiku (koja definiše istinitosnu vrednost iskaza). Centralni problemi u iskaznoj logici su ispitivanje da li je data iskazna formula *valjana* (*tautologija*) tj. da li je tačna za sve moguće istinitosne vrednosti elementarnih iskaza od kojih je sačinjena (na primer tvrđenje „Ana je zubar ili Ana nije zubar“ je uvek tačno – i ako je tvrđenje „Ana je zubar“ tačno i ako je netačno), kao i ispitivanje da li je data iskazna formula *zadovoljiva*, tj. da li je tačna za neke istinitosne vrednosti elementarnih iskaza od kojih je sačinjena (na primer tvrđenje „Ana je zubar i Ana živi u Beogradu“ može, mada ne mora uvek biti tačno). Problem ispitivanja zadovoljivosti formule u KNF obliku poznat je kao problem SAT i on je centralni predstavnik klase NP-kompletnih problema. Postoji više metoda za ispitivanje valjanosti i zadovoljivosti.

Iskazna logika dovoljno je izražajna za opisivanje raznovrsnih problema, uključujući mnoge praktične probleme, kao što su, na primer, problemi raspoređivanja ili dizajniranje kombinatornih kola. Iskazna logika posebno je pogodna za opisivanje problema nad konačnim domenima. Naime, svaki objekat koji može imati konačan broj stanja može se opisati konačnim brojem iskaznih promenljivih: ako je broj mogućih stanja 2^n, onda je dovoljno koristiti n iskaznih promenljivih. Svi brojevi zapisani u računaru zapisani su bitovima, pa se i svi ti brojevi mogu modelovati iskaznim promenljivim: koliko bitova, toliko iskaznih promenljivih. Sabiranje celih brojeva (kao i mnoge druge operacije) onda može da se opiše u terminima iskazne logike. Slično važi i za mnoge druge vrste podataka i mnoge vrste problema. Sa tako velikom izražajnom snagom i velikim brojem raznolikih primena, iskazna logika i rešavači za iskaznu logiku često se smatraju „švajcarskim nožićem“ savremenog računarstva, a posebno – veštačke inteligencije.

Primer 8.1. *Razmotrimo, za ilustraciju rešavanja primenom logike, problem „n dama” (opisan ukratko u primeru 3.4 i rešavan u poglavlju 8.4.3). Cilj je rasporediti n dama na šahovskoj tabli dimenzija $n \times n$ tako da se nikoje dve dame ne napadaju. Na narednoj slici prikazano je kretanje dame i jedno rešenje problema za $n = 8$.*



Jednostavnosti radi, u nastavku ćemo razmatrati problem dimenzije 3*, za koji je jedno neispravno raspoređivanje tri dame (raspoređivanje koje ne čini rešenje) prikazano na narednoj slici.*



Uslovi koje ispravno raspoređivanje treba da zadovolji su:

- *na nekom od polja* $a1$, $a2$, $a3$ *nalazi se dama.*
- *na nekom od polja* $b1$, $b2$, $b3$ *nalazi se dama.*
- *na nekom od polja* $c1$, $c2$, $c3$ *nalazi se dama.*
- *ako je neka dama na polju* $a1$, *onda na polju* $a2$ *ne može da bude dama.*
- *ako je neka dama na polju* $a1$, *onda na polju* $a3$ *ne može da bude dama.*
- *ako je neka dama na polju* $a2$, *onda na polju* $a1$ *ne može da bude dama.*
- *ako je neka dama na polju* $a2$, *onda na polju* $a3$ *ne može da bude dama.*
- *ako je neka dama na polju* $a3$, *onda na polju* $a1$ *ne može da bude dama.*
- *ako je neka dama na polju* $a3$, *onda na polju* $a2$ *ne može da bude dama.*
- ...
- *ako je neka dama na polju* $a2$, *onda na polju* $b3$ *ne može da bude dama.*
- *ako je neka dama na polju* $b3$, *onda na polju* $a2$ *ne može da bude dama.*
- *ako je neka dama na polju* $b1$, *onda na polju* $c2$ *ne može da bude dama.*
- *ako je neka dama na polju* $c2$, *onda na polju* $b1$ *ne može da bude dama.*

Navedeni uslovi zavise od iskaza oblika „na polju ?? nalazi se dama“. Označimo sa p_{a1} *iskaz „na polju* $a1$ *nalazi se dama“, sa* p_{a2} *iskaz „na polju* $a2$ *nalazi se dama“, ..., sa* p_{c3} *iskaz „na polju* $c3$ *nalazi se dama“. Onda navedeni uslovi mogu da se zapišu kraće:*

- p_{a1} *ili* p_{a2} *ili* p_{a3}.
- p_{b1} *ili* p_{b2} *ili* p_{b3}.
- p_{c1} *ili* p_{c2} *ili* p_{c3}.
- *ako je* p_{a1}, *onda nije* p_{a2}.
- *ako je* p_{a1}, *onda nije* p_{a3}.
- *ako je* p_{a2}, *onda nije* p_{a1}.
- *ako je* p_{a2}, *onda nije* p_{a3}.
- *ako je* p_{a3}, *onda nije* p_{a1}.
- *ako je* p_{a3}, *onda nije* p_{a2}.
- ...
- *ako je* p_{a2}, *onda nije* p_{b3}.
- *ako je* p_{b3}, *onda nije* p_{a2}.
- *ako je* p_{b1}, *onda nije* p_{c2}.
- *ako je* p_{c2}, *onda nije* p_{b1}.

Ovim su, od jednostavnih iskaza, konstruisani složeniji. Svi navedeni složeni iskazi zajedno čine još složeniji iskaz – iskaz koji sadrži sve uslove zadatka. Sintaksa iskazne logike govori o pravilima po kojim se od elementarnih iskaza mogu konstruisati složeniji, to jest, o pravilima za konstruisanje ispravnih iskaznih formula.

Svaki od jednostavnih iskaza kao što je p_{a1} može biti tačan ili netačan. U zavisnosti od toga, može se odrediti istinitosna vrednost složenijih iskaza. Na primer, ako je p_{a1} tačno, a p_{a2} netačno, onda je tačno i „ako je p_{a1}, onda nije p_{a2}". Semantika iskazne logike govori o tome kako se složenim iskazima (to jest, iskaznim formulama) određuje istinitosna vrednost na osnovu istinitosnih vrednosti elementarnih iskaza. Pošto je u fazi rešavanja problema vrednost iskaza kao što je p_{a1} nepoznata i pošto on može biti tačan ili netačan, p_{a1} ćemo zvati i iskazna promenljiva.

Sâmo rešavanje ovako modelovanog problema „n dama" može se svesti na rešavanje sledećeg problema: odrediti istinitosne vrednosti elementarnih iskaza $p_{a1}, p_{a2}, \ldots, p_{c3}$, takve da svi navedeni uslovi budu ispunjeni (to jest, da svi odgovarajući iskazi imaju istinitosnu vrednost tačno). Može se postaviti pitanje da li takve istinitosne vrednosti uopšte postoje, tj. da li početni problem uopšte ima rešenja (a ako ima – mogu se tražiti sva rešenja). Pitanje da li uopšte postoji rešenje može se rešiti razmatranjem svih mogućih varijacija vrednosti za $p_{a1}, p_{a2}, \ldots, p_{c3}$. Takvih varijacija ima $2^9 = 512$ i razmatranje svih je naporno i nepraktično. Postoje i metode koje ne razmatraju sve mogućnosti i postojanje rešenja mogu obično da ispitaju znatno efikasnije.

U kontekstu navedenih ograničenja, mogu se izvesti i neki zaključci. Na primer, ako važi p_{a1} onda važi p_{b2} ili p_{b3}. Ovakvi zaključci mogu se dobiti različitim pristupima, a mogu se koristiti za ubrzavanje traganja za rešenjem.

8.1 Sintaksa i semantika iskazne logike

Sintaksa iskazne logike govori o njenom *jeziku* — o skupu njenih (ispravno formiranih) formula i ne razmatra njihovo značenje i njihove (moguće) istinitosne vrednosti. Semantika iskazne logike definiše značenje, tj. tačnost iskaznih formula. Tačnost jedne iskazne formule definiše se na osnovu tačnosti elementarnih iskaza od kojih je ona sačinjena.

8.1.1 Sintaksa iskazne logike

Skup iskaznih formula obično se definiše za fiksiran, prebrojiv skup *iskaznih promenljivih* (*iskaznih varijabli*) ili *iskaznih slova* P, dve logičke konstante — $\top$ (*te*) i $\bot$ (*nete*), kao i konačan skup osnovnih logičkih (tj. bulovskih) veznika: unarnog — *negacija* i binarnih — *konjunkcija*, *disjunkcija*, *implikacija*, *ekvivalencija*.

Definicija 8.1 (Skup iskaznih formula)**.**

- *Iskazne promenljive (elementi skupa P) i logičke konstante su iskazne formule;*
- *Ako su A i B iskazne formule, onda su iskazne formule i objekti dobijeni kombinovanjem ovih formula logičkim veznicima;*
- *Iskazne formule mogu biti dobijene samo na navedene načine.*

Uobičajeno je da se formule zapisuju u vidu nizova simbola i da se negacija zapisuje kao $\neg$, konjunkcija kao $\wedge$, disjunkcija kao $\vee$, implikacija kao $\Rightarrow$ i ekvivalencija kao $\Leftrightarrow$. U takvom zapisu, ako su A i B iskazne formule, onda su iskazne formule i $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ i $(A \Leftrightarrow B)$. Na primer, zapis $(A \wedge \top)$ čitamo „A i te". U ovakvom zapisu, neophodno je koristiti zagrade kako bi se izbegla višesmislenost. Da bi se izbeglo korišćenje velikog broja zagrada, obično se izostavljaju spoljne zagrade i podrazumeva se sledeći prioritet veznika (od višeg ka nižem): $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$.

Elemente skupa P i logičke konstante zovemo *atomičkim iskaznim formulama*. *Literal* je iskazna formula koja je ili atomička iskazna formula ili negacija atomičke iskazne formule.

Ako su dve iskazne formule A i B identične, onda to zapisujemo $A = B$, a inače pišemo $A \neq B$.

Elementi skupa P obično se označavaju malim latiničnim slovima (eventualno sa indeksima). Iskazne formule obično se označavaju velikim latiničnim slovima (eventualno sa indeksima). Skupovi iskaznih formula obično se označavaju velikim slovima grčkog alfabeta (eventualno sa indeksima).

Primer 8.2. *Uslovi iz primera 8.1 mogli bi da se zapišu kao iskazne formule nad skupom iskaznih promen-*

ljivih $\{p_{a1}, p_{a2}, \ldots, p_{c3}\}$. *Na primer, uslov „na nekom od polja* $a1$, $a2$, $a3$ *nalazi se dama", tj. uslov „*p_{a1} *ili* p_{a2} *ili* p_{a3}*" može da se zapiše kao* $p_{a1} \vee p_{a2} \vee p_{a3}$. *Analogno zapisujemo i preostale uslove:*

- $p_{b1} \vee p_{b2} \vee p_{b3}$.
- $p_{c1} \vee p_{c2} \vee p_{c3}$.
- $p_{a1} \Rightarrow \neg p_{a2}$.
- $p_{a1} \Rightarrow \neg p_{a3}$.
- $p_{a2} \Rightarrow \neg p_{a1}$.
- ...
- $p_{c2} \Rightarrow \neg p_{b1}$.

Ako iskazne formule zamišljamo u vidu stabla (ili ih predstavljamo u računarskom programu u vidu stabla), onda svakom podstablu stabla koje odgovara nekoj iskaznoj formuli takođe odgovara iskazna formula. Sve te formule zovemo *potformulama* (eng. *subformulae*) formule koja odgovara korenu stabla. Skup potformula jedne formule može se, preciznije, definisati na sledeći način.

Definicija 8.2 (Skup potformula)**.** Skup potformula *formule* A *definisan je na sledeći način:*

- *svaka iskazna formula* A *je potformula sâma sebi;*
- *ako je* A *jednako* $\neg B$*, onda je svaka potformula formule* B *istovremeno i potformula formule* A*. Ako je* A *jednako* $B \wedge C$, $B \vee C$, $B \Rightarrow C$ *ili* $B \Leftrightarrow C$*, onda je svaka potformula formule* B *i svaka potformula formule* C *istovremeno i potformula formule* A*;*
- *Potformule formule* A *su samo formule opisane prethodnim stavkama.*

Primer 8.3. *Skup potformula formule* $(p \Rightarrow q) \vee r$ *je* $\{p, q, r, p \Rightarrow q, (p \Rightarrow q) \vee r\}$.

Na različite načine može se definisati *složenost* iskaznih formula. Na primer, složenost može biti definisana kao dubina stabla koje odgovara formuli ili kao broj logičkih vezika koje formula sadrži.

Često je potrebno jedan deo iskazne formule zameniti nekom drugom iskaznom formulom. Pojam zamene precizno uvodi naredna definicija.

Definicija 8.3 (Zamena)**.** *Rezultat* zamene (supstitucije) *svih pojavljivanja iskazne formule* C *u iskaznoj formuli* A *iskaznom formulom* D *označavamo sa* $A[C \mapsto D]$*. Ta zamena definiše se na sledeći način:*

- *ako važi* $A = C$*, onda je* $A[C \mapsto D] = D$*;*
- *ako važi* $A \neq C$*, onda:*
  - *ako je* A *atomička iskazna formula, onda je* $A[C \mapsto D] = A$*;*
  - *ako za neku formulu* B *važi da je formula* A *jednaka* $(\neg B)$ *, onda je* $A[C \mapsto D] = \neg(B[C \mapsto D])$*;*
  - *ako za neke formule* B_1 *i* B_2 *važi da je formula* A *jednaka* $B_1 \wedge B_2$, $B_1 \vee B_2$, $B_1 \Rightarrow B_2$ *ili* $B_1 \Leftrightarrow B_2$*, onda je formula* $A[C \mapsto D]$ *jednaka (redom)* $B_1[C \mapsto D] \wedge B_2[C \mapsto D]$, $B_1[C \mapsto D] \vee B_2[C \mapsto D]$, $B_1[C \mapsto D] \Rightarrow B_2[C \mapsto D]$ *ili* $B_1[C \mapsto D] \Leftrightarrow B_2[C \mapsto D]$.

Primer 8.4.
$((p \Rightarrow q) \vee r)[r \mapsto p] = (p \Rightarrow q) \vee p$
$((p \Rightarrow q) \vee r)[p \Rightarrow q \mapsto \neg p \vee q] = (\neg p \vee q) \vee r$
$(p \wedge (q \Rightarrow r))\,[q \mapsto r] = p \wedge (r \Rightarrow r)$
$((p \Rightarrow q) \vee r)[s \mapsto p] = ((p \Rightarrow q) \vee r)$
$((p \vee q) \wedge (p \vee r))[p \mapsto s] = ((s \vee q) \wedge (s \vee r))$

Ako iskazne formule predstavimo u računarskom programu u vidu stabla, onda opisanu zamenu možemo opisati rekurzivnom funkcijom (čiji su bazni slučajevi opisani prvim dvema stavkama definicije). Ta funkcija

zamenjuje jedno podstablo formule (zapravo — sva podstabla koja odgovaraju istoj formuli) nekim drugim zadatim stablom.

8.1.2 Semantika iskazne logike

Semantika iskazne logike govori o *istinitosnim vrednostima* iskaznih formula. Istinitosna vrednost iskazne formule može biti 0 ili 1 (intuitivno – *netačno* ili *tačno*).[1] Istinitosne vrednosti formula definišu se u skladu sa uobičajenim, svakodnevnim rasuđivanjem. Zbog toga, definicija semantike iskazne logike može da deluje i suvišno, ali ona je potrebna da bi baratanje formulama moglo da se opiše na precizan način (pogodan, na primer, za računarsku obradu).

Formula $\bot$ ima istinitosnu vrednost 0, a formula $\top$ ima istinitosnu vrednost 1. Istinitosna vrednost složenih (neatomičkih) formula zavisi samo od istinitosne vrednosti njenih potformula. Dakle, u krajnjoj instanci, istinitosna vrednost formule zavisi (samo) od istinitosnih vrednosti iskaznih promenljivih koje se u njoj pojavljuju. Funkcije koje pridružuju istinitosnu vrednost promenljivim (tj. funkcije v iz P u $\{0,1\}$) zovemo *valuacijama* (eng. *valuation, assignment*). *Interpretacija* (eng. *interpretation*) je proširenje valuacije: svaka valuacija v određuje jednu funkciju I_v koju zovemo *interpretacijom za valuaciju* v i koja pridružuje (jedinstvene) istinitosne vrednosti formulama (tj. preslikava skup iskaznih formula u skup $\{0,1\}$).

Definicija 8.4 (Interpretacija)**.** Interpretacija I_v *za valuaciju* v *definiše se na sledeći način:*

- $I_v(\top) = 1$ *i* $I_v(\bot) = 0$*;*
- $I_v(p) = v(p)$*, za svaki element* p *skupa* P*;*
- $I_v(\neg A) = \begin{cases} 1, & \text{ako je } I_v(A) = 0 \\ 0, & \text{inače} \end{cases}$
- $I_v(A \wedge B) = \begin{cases} 1, & \text{ako je } I_v(A) = 1 \text{ i } I_v(B) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v(A \vee B) = \begin{cases} 0, & \text{ako je } I_v(A) = 0 \text{ i } I_v(B) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(A \Rightarrow B) = \begin{cases} 0, & \text{ako je } I_v(A) = 1 \text{ i } I_v(B) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(A \Leftrightarrow B) = \begin{cases} 1, & \text{ako je } I_v(A) = I_v(B) \\ 0, & \text{inače} \end{cases}$

Vrednost $I_v(A)$ zovemo *istinitosnom vrednošću iskazne formule* A *u interpretaciji* I_v (ili *u valuaciji* v). Ako za valuaciju v važi $I_v(A) = 1$, onda se za formulu A kaže da je *tačna u interpretaciji* I_v, a inače da je *netačna u interpretaciji* I_v.

Primer 8.5. *Tvrđenje „Ana je zubar i Ana živi u Beogradu“ može da se opiše kao konjunkcija* $p \wedge q$*, gde iskazna promenljiva* p *odgovara elementarnom iskazu „Ana je zubar“, a iskazna promenljiva* q *odgovara elementarnom iskazu „Ana živi u Beogradu“. Može biti da je Ana zubar i može biti da Ana nije zubar, tj. istinitosna vrednost promenljive* p *može biti* 0 *ili* 1*. Ukoliko je* $v(p) = 0$*, tj. ukoliko Ana nije zubar, na osnovu definicije semantike, za valuaciju* v *važi* $I_v(p) = 0$ *i, dalje,* $I_v(p \wedge q) = 0$*, tj. tvrđenje „Ana je zubar i Ana živi u Beogradu“ nije tačno (bez obzira na istinitosnu vrednost iskaza „Ana živi u Beogradu“).*

Primer 8.6. *U primeru 8.2 (koji je nastavak primera 8.1), ako je* $v(p_{a1}) = 1$ *i* $v(p_{a2}) = 1$*, onda važi* $I_v(p_{a1} \Rightarrow \neg p_{a2}) = 0$*. Ovo govori da ni u jednom rešenju ne mogu dame da budu i na polju* $a1$ *i na polju* $a2$*.*

Definicija 8.5 (Zadovoljivost, valjanost, kontradiktornost, porecivost)**.** *Iskazna formula* A *je:*

[1] Za istinitosne vrednosti namerno se ne uzimaju $\bot$ i $\top$, već 0 i 1, da bi se jasno razlikovali svet sintakse i svet semantike iskaznih formula.

- zadovoljiva *(eng.* satisfiable*), ako postoji valuacija v u kojoj je A tačna (i tada se kaže da je v* model *za A);*
- valjana *(eng.* valid*) ili* tautologija *(eng.* tautology*), ako je svaka valuacija v model za A i to zapisujemo $\models A$;*
- nezadovoljiva *(eng.* unsatisfiable*) ili* kontradikcija *(eng.* contradictory*), ako ne postoji valuacija u kojoj je tačna;*
- poreciva *(eng.* falsifiable*), ako postoji valuacija u kojoj nije tačna.*

Primer 8.7. *Tvrđenje „Ana je zubar ili Ana nije zubar“ može da se opiše kao disjunkcija $p \vee \neg p$, gde iskazna promenljiva p odgovara elementarnom iskazu „Ana je zubar“. Može biti da je Ana zubar i može biti da Ana nije zubar, tj. istinitosna vrednost promenljive p može biti 0 ili 1. To su dve moguće relevantne valuacije za dato tvrđenje. Ukoliko je $v(p) = 1$, tj. ukoliko Ana jeste zubar, na osnovu definicije semantike važi $I_v(p \vee \neg p) = 1$, a ukoliko je $v(p) = 0$, tj. ukoliko Ana nije zubar, na osnovu definicije semantike važi $I_v(\neg p) = 1$ i, dalje, $I_v(p \vee \neg p) = 1$. Dakle, formula $p \vee \neg p$, tj. tvrđenje „Ana je zubar ili Ana nije zubar“ je tautologija.*

Primer 8.8. *Iskazne formule $p \Rightarrow p$ i $p \vee \neg p$ su tautologije, iskazna formula $p \Rightarrow q$ je zadovoljiva i poreciva, a iskazna formula $p \wedge \neg p$ je kontradikcija.*

Primer 8.9. *Ako su iskazne formule A i $A \Rightarrow B$ tautologije, onda je i B tautologija. Zaista, pretpostavimo da su A i $A \Rightarrow B$ tautologije i da postoji valuacija v takva da u interpretaciji I_v formula B nije tačna. Formula A je tautologija, pa je tačna i u interpretaciji I_v. Kako je u interpretaciji I_v formula A tačna, a formula B netačna, formula $A \Rightarrow B$ u njoj nije tačna, što protivreči pretpostavci da je $A \Rightarrow B$ tautologija. Dakle, formula B je tačna za svaku valuaciju, pa je ona tautologija.*

Definicija 8.6 (Zadovoljivost i kontradiktornost skupa formula)**.** *Skup iskaznih formula Γ je*

- zadovoljiv*, ako postoji valuacija v u kojoj je svaka formula iz Γ tačna. Za takvu valuaciju v kaže se da je* model *za Γ.*
- nezadovoljiv *ili* kontradiktoran*, ako ne postoji valuacija u kojoj je svaka formula iz Γ tačna.*

Primer 8.10. *Skup iskaznih formula $\{p \Rightarrow q, p, \neg q\}$ je kontradiktoran (ali nijedan njegov pravi podskup nije kontradiktoran).*

Naredna teorema govori da uslov koji rešenje nekog problema mora da zadovolji može da se razmatra ne samo kao skup svih pojedinačnih poduslova, već i kao konjunkcija formula koje odgovaraju tim poduslovima.

Teorema 8.1. *Valuacija v je model skupa formula $\{A_1, \ldots, A_n\}$ ako i samo ako je v model formule $A_1 \wedge \ldots \wedge A_n$.*

8.2 Logičke posledice i logički ekvivalentne formule

Često je veoma važno pitanje da li je neki iskaz posledica nekih drugih iskaza. Ovo pitanje može se opisati u terminima pojma *logičke posledice.*

Definicija 8.7 (Logička posledica i logička ekvivalencija)**.** *Ako je svaki model za skup iskaznih formula Γ istovremeno i model za iskaznu formulu A, onda se kaže da je A* logička posledica *(eng.* logical consequence*) skupa Γ i piše se $\Gamma \models A$.*

Ako je svaki model iskazne formule A model i iskazne formule B i obratno (tj. ako važi $\{A\} \models B$ i $\{B\} \models A$), onda se kaže se da su formule A i B logički ekvivalentne *(eng.* logically equivalent*) i piše se*

$A \equiv B$.

Primer 8.11. *Tvrđenje „Boban živi u Kruševcu“ je logička posledica skupa tvrđenja { „Boban živi u Beogradu ili Boban živi u Kruševcu“, „Boban ne živi u Beogradu” }. Zaista, ako je valuacija v proizvoljan model formula „Boban živi u Beogradu ili Boban živi u Kruševcu“ i „Boban ne živi u Beogradu”, na osnovu definicije semantike, u interpretaciji I_v mora da je tačan iskaz „Boban živi u Beogradu“ ili iskaz “Boban živi u Kruševcu“. Prva mogućnost otpada zbog uslova „Boban ne živi u Beogradu”, pa mora da je tačan iskaz „Boban živi u Kruševcu“.*

Primer 8.12. *Važi $\{A, A \Rightarrow B\} \models B$. Zaista, ako je valuacija v proizvoljan model formula A i $A \Rightarrow B$, onda kako je $I_v(A) = 1$, iz $I_v(A \Rightarrow B) = 1$ sledi da ne može da važi $I_v(B) = 0$, pa mora da važi $I_v(B) = 1$, tj. v je model i za B, što je i trebalo dokazati.*

Ako ne važi $\Gamma \models A$, onda to zapisujemo $\Gamma \not\models A$.

Ako važi $\{\} \models A$, onda je svaka valuacija model za formulu A, tj. A je valjana formula. Važi i obratno — ako je A valjana formula, onda važi $\{\} \models A$. Zato umesto $\{\} \models A$, pišemo kraće $\models A$, kao što zapisujemo valjane formule.

Ako je skup Γ kontradiktoran, onda je proizvoljna formula njegova logička posledica (naime, ako skup Γ nema modela, onda za proizvoljnu formulu A važi da je svaki model za Γ – a takvih nema – istovremeno i model za A). Specijalno, svaka formula je logička posledica skupa $\{\bot\}$.

Ako je skup Γ konačan, onda umesto $\{A_1, \ldots, A_n\} \models B$ pišemo kraće $A_1, \ldots, A_n \models B$. Koristeći teoremu 8.1, lako se može dokazati naredna teorema.

Teorema 8.2. *Važi $A_1, \ldots, A_n \models B$ ako i samo ako važi $A_1 \wedge \ldots \wedge A_n \models B$.*

Ako važi $A \equiv B$, onda u bilo kojoj valuaciji formule A i B imaju jednake vrednosti. Tvrđenja oblika $A \equiv B$ zovemo *logičkim ekvivalencijama*. Relacija $\equiv$ je, očigledno, relacija ekvivalencije nad skupom iskaznih formula. U razmatranju skupova uslova, bilo koji uslov očigledno može biti zamenjen nekim njemu logički ekvivalentnim uslovom jer time neće biti promenjen skup modela celokupnog uslova.

Primer 8.13. *Za formule $p_{a1} \Rightarrow \neg p_{a3}$ i $p_{a3} \Rightarrow \neg p_{a1}$ iz primera 8.1, može se pokazati da važi: $p_{a1} \Rightarrow \neg p_{a3} \equiv p_{a3} \Rightarrow \neg p_{a1}$. To govori da nije potrebno da u skupu uslova postoje obe formule, dovoljno je zadržati jednu od njih. Isto važi i za druge analogne parove formula, te je dovoljno razmatrati sledeći skup formula:*

$p_{a1} \vee p_{a2} \vee p_{a3}$, $p_{b1} \vee p_{b2} \vee p_{b3}$, $p_{c1} \vee p_{c2} \vee p_{c3}$,

$p_{a1} \Rightarrow \neg p_{a2}$, $p_{a1} \Rightarrow \neg p_{a3}$, $p_{a2} \Rightarrow \neg p_{a3}$,

$p_{b1} \Rightarrow \neg p_{b2}$, $p_{b1} \Rightarrow \neg p_{b3}$, $p_{b2} \Rightarrow \neg p_{b3}$,

$p_{c1} \Rightarrow \neg p_{c2}$, $p_{c1} \Rightarrow \neg p_{c3}$, $p_{c2} \Rightarrow \neg p_{c3}$,

$p_{a1} \Rightarrow \neg p_{b1}$, $p_{a1} \Rightarrow \neg p_{c1}$, $p_{b1} \Rightarrow \neg p_{c1}$,

$p_{a2} \Rightarrow \neg p_{b2}$, $p_{a2} \Rightarrow \neg p_{c2}$, $p_{b2} \Rightarrow \neg p_{c2}$,

$p_{a3} \Rightarrow \neg p_{b3}$, $p_{a3} \Rightarrow \neg p_{c3}$, $p_{b3} \Rightarrow \neg p_{c3}$,

$p_{a3} \Rightarrow \neg p_{b2}$, $p_{a3} \Rightarrow \neg p_{c1}$, $p_{b2} \Rightarrow \neg p_{c1}$,

$p_{a2} \Rightarrow \neg p_{b1}$, $p_{b3} \Rightarrow \neg p_{c2}$,

$p_{a1} \Rightarrow \neg p_{b2}$, $p_{a1} \Rightarrow \neg p_{c3}$, $p_{b2} \Rightarrow \neg p_{c3}$,

$p_{a2} \Rightarrow \neg p_{b3}$, $p_{b1} \Rightarrow \neg p_{c2}$.

Primer 8.14. *Neke od logičkih ekvivalencija (ili, preciznije, neke od shema logičkih ekvivalencija) su:*

$\neg\neg A$	$\equiv$	A	*zakon dvojne negacije*
$A \vee \neg A$	$\equiv$	$\top$	*zakon isključenja trećeg*
$A \wedge A$	$\equiv$	A	*zakon idempotencije za* $\wedge$
$A \vee A$	$\equiv$	A	*zakon idempotencije za* $\vee$
$A \wedge B$	$\equiv$	$B \wedge A$	*zakon komutativnosti za* $\wedge$
$A \vee B$	$\equiv$	$B \vee A$	*zakon komutativnosti za* $\vee$
$A \Leftrightarrow B$	$\equiv$	$B \Leftrightarrow A$	*zakon komutativnosti za* $\Leftrightarrow$
$A \wedge (B \wedge C)$	$\equiv$	$(A \wedge B) \wedge C$	*zakon asocijativnosti za* $\wedge$
$A \vee (B \vee C)$	$\equiv$	$(A \vee B) \vee C$	*zakon asocijativnosti za* $\vee$
$A \Leftrightarrow (B \Leftrightarrow C)$	$\equiv$	$(A \Leftrightarrow B) \Leftrightarrow C$	*zakon asocijativnosti za* $\Leftrightarrow$
$A \wedge (A \vee B)$	$\equiv$	A	*zakon apsorpcije*
$A \vee (A \wedge B)$	$\equiv$	A	*zakon apsorpcije*
$A \wedge (B \vee C)$	$\equiv$	$(A \wedge B) \vee (A \wedge C)$	*zakon distributivnosti* $\wedge$ *u odnosu na* $\vee$
$(B \vee C) \wedge A$	$\equiv$	$(B \wedge A) \vee (C \wedge A)$	*zakon distributivnosti* $\wedge$ *u odnosu na* $\vee$
$A \vee (B \wedge C)$	$\equiv$	$(A \vee B) \wedge (A \vee C)$	*zakon distributivnosti* $\vee$ *u odnosu na* $\wedge$
$(B \wedge C) \vee A$	$\equiv$	$(B \vee A) \wedge (C \vee A)$	*zakon distributivnosti* $\vee$ *u odnosu na* $\wedge$
$\neg(A \wedge B)$	$\equiv$	$\neg A \vee \neg B$	*De Morganov zakon*
$\neg(A \vee B)$	$\equiv$	$\neg A \wedge \neg B$	*De Morganov zakon*
$A \wedge \top$	$\equiv$	A	*zakon konjunkcije sa tautologijom*
$A \vee \top$	$\equiv$	$\top$	*zakon disjunkcije sa tautologijom*
$A \wedge \bot$	$\equiv$	$\bot$	*zakon konjunkcije sa kontradikcijom*
$A \vee \bot$	$\equiv$	A	*zakon disjunkcije sa kontradikcijom*

Logičke ekvivalencije navedene u primeru 8.14, pokazuju, između ostalog, da su konjunkcija i disjunkcija komutativni i asocijativni veznici. Zato možemo smatrati da konjunkcija (i disjunkcija) mogu da povezuju više od dve formule, pri čemu ne moramo da vodimo računa o njihovom poretku. Svaki član uopštene konjunkcije zovemo *konjunkt*, a svaki član uopštene disjunkcije zovemo *disjunkt*. Disjunkciju više literala (pri čemu njihov poredak nije bitan) zovemo *klauza* (eng. *clause*). Klauza je *jedinična* ako sadrži samo jedan literal.

Naglasimo da, na primer, $B \models A$ i $B \equiv A$ nisu formule iskazne logike, nego su to *formule koje govore o iskaznim formulama*, pa ih zato zovemo *meta formule* (dok iskazne formule čine *objektne formule*). Naredna teorema povezuje meta i objektni nivo i govori o tome kako ispitivanje da li su neke dve formule logički ekvivalentne i da li je jedna logička posledica druge može da se svede na objektni nivo i na problem ispitivanja da li je neka formula tautologija.

Teorema 8.3.
Važi $A \models B$ ako i samo ako je iskazna formula $A \Rightarrow B$ tautologija.
Važi $A \equiv B$ ako i samo ako je iskazna formula $A \Leftrightarrow B$ tautologija.

Na osnovu prethodne dve teoreme sledi da važi $A_1, \ldots, A_n \models B$ ako i samo ako je formula $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ tautologija.

Primer 8.15. *Ako važi „kiša pada“ i ”ako kiša pada, onda će meč biti otkazan“, onda važi i „meč će biti otkazan“. Naime, iskaz „meč će biti otkazan“ je logička posledica skupa iskaza { „kiša pada“, „ako kiša pada, onda će meč biti otkazan“ } ako i samo ako je formula*
„kiša pada“ $\wedge$ „ako kiša pada, onda će meč biti otkazan” $\Rightarrow$ „meč će biti otkazan“
tautologija (što jeste ispunjeno).

Već je rečeno da u razmatranju skupova uslova, svaki uslov može biti zamenjen nekim njemu logički ekvivalentnim uslovom (i time neće biti promenjen skup modela celokupnog uslova). Naredna teorema tvrdi i sledeće: ako se u formuli A neka njena potformula zameni njoj logički ekvivalentnom formulom, biće dobijena formula koja je logički ekvivalentna formuli A. Ova teorema opravdava *transformisanje* iskazne formule u neki oblik pogodan za, na primer, ispitivanje zadovoljivosti.

Teorema 8.4 (Teorema o zameni)**.** *Ako je $C \equiv D$, onda je $A[C \mapsto D] \equiv A$.*

Primer 8.16. *Kako je* $q \Rightarrow r \equiv \neg q \vee r$*, važi* $(p \wedge (q \Rightarrow r))[q \Rightarrow r \mapsto \neg q \vee r] = p \wedge (\neg q \vee r) \equiv p \wedge (q \Rightarrow r)$*.*

Primer 8.17. *Kako je* $\neg(\neg p \wedge \neg q) \equiv p \vee q$*, važi da su formule* $\neg(\neg p \wedge \neg q) \vee r$ *i* $p \vee q \vee r$ *logički ekvivalentne. Odatle dalje sledi da se u C programu red*

```
if (!(!a && !b) || c)
```

može zameniti sledećim redom:

```
if (a || b || c)
```

Primetimo da zagrade oko izraza `a || b` *nisu neophodne jer je disjunkcija asocijativni veznik.*

8.3 Ispitivanje zadovoljivosti i valjanosti u iskaznoj logici

U praktičnim primenama, onda kada je neki cilj formulisan na jeziku iskazne logike, centralni problem postaje ispitivanje da li je neka iskazna formula tautologija, da li je zadovoljiva i slično. Početno pitanje je da li su ti problemi uopšte odlučivi, tj. da li postoje algoritmi koji *uvek* mogu da daju odgovor na njih. Na primer, pitanje je da li postoji algoritam koji za svaku formulu koja je tautologija može da utvrdi da jeste tautologija, a za svaku formulu koja nije tautologija može da utvrdi da nije tautologija. Jednostavno se, kao što je pokazano u narednom poglavlju, pokazuje da takav algoritam postoji, pa je problem ispitivanja tautologičnosti odlučiv. Dodatno, odlučivi su i problemi ispitivanja zadovoljivosti, porecivosti i kontradiktornosti. Štaviše, ovi problemi su blisko povezani i bilo koja tri mogu se lako svesti na četvrti. To znači da je dovoljno da imamo efikasan algoritam za jedan od ova četiri problema. U nastavku ćemo se fokusirati na rešavanje problema zadovoljivosti. Algoritam za ispitivanje zadovoljivosti onda možemo iskoristiti i za preostala tri problema: formula A je tautologija akko $\neg A$ nije zadovoljiva, A je poreciva akko je $\neg A$ zadovoljiva, a A je kontradikcija akko A nije zadovoljiva.

8.3.1 Istinitosne tablice i odlučivost problema valjanosti i zadovoljivosti

Na osnovu definicije interpretacije, može se konstruisati istinitosna tablica za proizvoljnu iskaznu formulu. Istinitosne tablice pogodne su za ispitivanje i valjanosti i zadovoljivosti i nezadovoljivosti i porecivosti. U istinitosnoj tablici za neku formulu, svakoj vrsti odgovara jedna valuacija iskaznih slova koja se pojavljuju u toj formuli. Svakoj koloni odgovara jedna potformula te formule. Ukoliko iskazna formula A sadrži iskazne promenljive $p_1, p_2, \ldots, p_n$, onda istinitosna tablica treba da sadrži sve moguće valuacije za ovaj skup promenljivih (valuacije za druge promenljive nisu relevantne). Takvih valuacija ima 2^n. U zavisnosti od vrednosti iskaznih promenljivih, izračunavaju se vrednosti potformula razmatrane iskazne formule, sve do sâme te formule. Ako su u koloni koja odgovara sâmoj iskaznoj formuli sve vrednosti jednake 1, onda je formula tautologija. Ako je bar jedna vrednost jednaka 1, formula je zadovoljiva. Ako je bar jedna vrednost jednaka 0, formula je poreciva. Ako su sve vrednosti jednake 0, formula je kontradikcija. Ovo pokazuje da su problemi ispitivanja valjanosti, zadovoljivosti, nezadovoljivosti i porecivosti odlučivi problemi, tj. postoje algoritmi koji ih mogu rešiti. Međutim, opisani algoritam, zasnovan na istinitosnim tablicama, naivan je i potpuno neupotrebljiv za realne probleme. Za rešavanje teških realnih problema potrebno je razviti znatno efikasnije algoritme.

Primer 8.18. *Iskaznoj formuli* $(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$ *odgovara sledeća istinitosna tablica:*

p	q	$\neg q$	$\neg p$	$\neg q \Rightarrow \neg p$	$p \Rightarrow q$	$(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$
0	0	1	1	1	1	1
0	1	0	1	1	1	1
1	0	1	0	0	0	1
1	1	0	0	1	1	1

Dakle, data formula je zadovoljiva i valjana. Ona nije poreciva i nije kontradikcija.

Primer 8.19. *U primeru 8.1, za tablu dimenzije* 3×3*, razmatra se skup formula nad 9 iskaznih promenljivih, te bi odgovarajuća istinitosna tablica imala* $2^9 = 512$ *vrsta.*

Primer 8.20. *Date su tri kutije, zna se da je tačno u jednoj od njih zlato. Na njima piše:*
na kutiji 1: „zlato nije ovde“

na kutiji 2: „zlato nije ovde“
na kutiji 3: „zlato je u kutiji 2“
Ako se zna da je tačno jedna tvrdnja tačna, treba pogoditi gde je zlato.
Neka iskazne promenljive b_1, b_2 i b_3 odgovaraju tvrdnjama „zlato je u kutiji 1“, „zlato je u kutiji 2“ i „zlato je u kutiji 3“.
Uslov da je tačno u jednoj od njih zlato može se opisati sledećom formulom:
$A = (b_1 \wedge \neg b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge b_3)$
Uslov da je tačno jedna ispisana tvrdnja tačna, može se opisati sledećom formulom:
$B = (\neg b_1 \wedge \neg\neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg b_2 \wedge \neg b_2) \vee (\neg\neg b_1 \wedge \neg\neg b_2 \wedge b_2)$
Potrebno je pronaći valuaciju u kojoj su obe formule A i B tačne:

b_1	b_2	b_3	A	B
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Postoji samo jedna valuacija v u kojoj su obe formule tačne. U njoj važi $I_v(b_1) = 1$, te je zlato u prvoj kutiji.

8.3.2 Normalne forme

Napredne procedure za ispitivanje valjanosti ili zadovoljivosti se, zbog jednostavnosti i veće efikasnosti, obično definišu samo za neke specifične vrste iskaznih formula, za formule koje su u nekoj specifičnoj formi.

Definicija 8.8 (Konjunktivna normalna forma)**.** *Iskazna formula je u* konjunktivnoj normalnoj formi *(KNF) ako je oblika*

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

pri čemu je svaka od formula A_i $(1 \leq i \leq n)$ klauza (tj. disjunkcija literala).

Definicija 8.9 (Disjunktivna normalna forma)**.** *Iskazna formula je u* disjunktivnoj normalnoj formi *(DNF) ako je oblika*

$$A_1 \vee A_2 \vee \ldots \vee A_n$$

pri čemu je svaka od formula A_i $(1 \leq i \leq n)$ konjunkcija literala.

Ako je iskazna formula A logički ekvivalentna iskaznoj formuli B i iskazna formula B je u konjunktivnoj (disjunktivnoj) normalnoj formi, onda se kaže da je formula B konjunktivna (disjunktivna) normalna forma formule A. Jedna iskazna formula može da ima više različitih konjunktivnih (disjunktivnih) normalnih formi (na primer, i formula $(p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (q \vee s)$ i formula $(s \vee q) \wedge (p \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge (p \vee \neg p)$ su konjunktivne normalne forme formule $(p \wedge q) \vee (r \wedge s)$). Slično, jedna formula koja je u konjunktivnoj normalnoj formi može biti konjunktivna normalna forma za više iskaznih formula.

Korišćenjem pogodnih ekvivalencija, svaka iskazna formula može biti transformisana u svoju konjunktivnu (disjunktivnu) normalnu formu. Transformisanje iskazne formule u konjunktivnu normalnu formu može biti opisano algoritmom prikazanim na slici 8.1. Kada se govori o „primeni neke logičke ekvivalencije“ misli se na korišćenje logičke ekvivalencije na osnovu teoreme o zameni (teorema 8.4).

Zaustavljanje algoritma KNF može se dokazati korišćenjem pogodno odabrane *mere zaustavljanja*.[2] Za neke pojedinačne korake, može se dokazati da se zaustavljaju korišćenjem jednostavnih mera — na primer, za prvi korak algoritma, kao mera se može koristiti broj veznika $\Leftrightarrow$ u formuli. Izlazna formula logički je ekvivalentna ulaznoj formuli na osnovu teoreme o zameni (teorema 8.4) i činjenice da se u algoritmu koriste samo logičke ekvivalencije.

[2] U cilju dokazivanja zaustavljanja postupka transformisanja formule u konjunktivnu normalnu formu definiše se preslikavanje τ (*mera zaustavljanja*) iz skupa iskaznih formula u skup prirodnih brojeva:

Algoritam: KNF

Ulaz: Iskazna formula F

Izlaz: Konjunktivna normalna forma formule F

1: **dok god** je to moguće **radi**
2: primeni logičku ekvivalenciju (eliminiši veznik $\Leftrightarrow$):
$A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$;
3: **dok god** je to moguće **radi**
4: primeni logičku ekvivalenciju (eliminiši veznik $\Rightarrow$):
$A \Rightarrow B \equiv \neg A \vee B$;
5: **dok god** je to moguće **radi**
6: primeni neku od logičkih ekvivalencija:
$\neg(A \wedge B) \equiv \neg A \vee \neg B$,
$\neg(A \vee B) \equiv \neg A \wedge \neg B$;
7: **dok god** je to moguće **radi**
8: primeni logičku ekvivalenciju (eliminiši višestruke veznike $\neg$):
$\neg\neg A \equiv A$;
9: **dok god** je to moguće **radi**
10: primeni neku od logičkih ekvivalencija:
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$,
$(B \wedge C) \vee A \equiv (B \vee A) \wedge (C \vee A)$.

Slika 8.1: Algoritam KNF.

Teorema 8.5 (Korektnost algoritma KNF). *Algoritam KNF se zaustavlja i ima sledeće svojstvo: ako je F ulazna formula, onda je izlazna formula F' u konjunktivnoj normalnoj formi i logički je ekvivalentna sa F.*

Transformisanje formule u disjunktivnu normalnu formu opisuje se algoritmom analognim algoritmu KNF.

Transformisanje formule u njenu konjunktivnu normalnu formu može da dâ formulu čija je složenost eksponencijalna u odnosu na složenost polazne formule. Na primer, transformisanjem formule

$$(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_n \wedge B_n)$$

(koja ima n disjunkata) u njenu konjunktivnu normalnu formu, dobija se formula koja ima 2^n konjunkta.

Zbog potencijalno ogromne izlazne formule, umesto algoritma KNF, u praksi se najčešće koristi Cejtinovo kodiranje – koje je linearno i u smislu vremena i u smislu prostora, ali uvodi dodatne promenljive, te zato rezultujuća formula nije logički ekvivalentna polaznoj već samo *slabo ekvivalentna*: početna formula je zadovoljiva ako i samo ako je zadovoljiva rezultujuća formula. To je za primene obično dovoljno dobro i, štaviše, iz modela za rezultujuću formulu (ukoliko oni postoje) mogu se rekonstruisati modeli za polaznu formulu. Smatraćemo da su iz polazne formule, korišćenjem adekvatnih logičkih ekvivalencija (poput $A \vee \top \equiv \top$), eliminisane sve konstante $\top$ i $\bot$. Cejtinova transformacija može se opisati na sledeći način. Neka $Sub(F)$ označava skup svih potformula formule F. Za svaku formulu A iz $Sub(F)$ uvodi se nova iskazna promenljiva p_A. Ako je A iskazna promenljiva, onda se p_A naziva *osnovna promenljiva*, a inače se p_A naziva *definiciona promenljiva*. Formula F se najpre transformiše u sledeću formulu (gde $\star$ označava binarni iskazni veznik iz skupa binarnih veznika koji se pojavljuju u F):

$$\begin{aligned} \tau(A) &= 2 \text{ (gde je } A \text{ atomička formula)} \\ \tau(\neg A) &= 2^{\tau(A)} \\ \tau(A \vee B) &= \tau(A) \cdot \tau(B) \\ \tau(A \wedge B) &= \tau(A) + \tau(B) + 1 \end{aligned}$$

Može se jednostavno dokazati da je vrednost $\tau(F_2)$ uvek manja od $\tau(F_1)$ ako je formula F_2 dobijena primenom nekog koraka algoritma na formulu F_1. Na primer, važi da je $\tau(\neg(A \vee B)) = 2^{\tau(A \vee B)} = 2^{\tau(A) \cdot \tau(B)}$ veće od $\tau(\neg A \wedge \neg B) = \tau(\neg A) + \tau(\neg B) + 1 = 2^{\tau(A)} + 2^{\tau(B)} + 1$. Odatle sledi da se postupak transformisanja proizvoljne formule u konjunktivnu normalnu formu zaustavlja za proizvoljnu ulaznu formulu F (jer ne postoji beskonačan strogo opadajući niz prirodnih brojeva čiji je prvi element $\tau(F)$).

vrsta formule	rezultujuće klauze
$p_A \Leftrightarrow (\neg p_B)$	$(p_A \vee p_B) \wedge (\neg p_A \vee \neg p_B)$
$p_A \Leftrightarrow (p_B \wedge p_C)$	$(p_A \vee \neg p_B \vee \neg p_C) \wedge (\neg p_A \vee p_B) \wedge (\neg p_A \vee p_C)$
$p_A \Leftrightarrow (p_B \vee p_C)$	$(\neg p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B) \wedge (p_A \vee \neg p_C)$
$p_A \Leftrightarrow (p_B \Rightarrow p_C)$	$(\neg p_A \vee \neg p_B \vee p_C) \wedge (p_A \vee p_B) \wedge (p_A \vee \neg p_C)$
$p_A \Leftrightarrow (p_B \Leftrightarrow p_C)$	$(\neg p_A \vee \neg p_B \vee p_C) \wedge (\neg p_A \vee p_B \vee \neg p_C) \wedge$ $(p_A \vee p_B \vee p_C) \wedge (p_A \vee \neg p_B \vee \neg p_C)$

Tabela 8.1: Pravila za Cejtinovu transformaciju.

$$p_F \wedge \bigwedge_{\substack{A \in Sub(F) \\ A = B \star C}} (p_A \Leftrightarrow (p_B \star p_C)) \wedge \bigwedge_{\substack{A \in Sub(F) \\ A = \neg B}} (p_A \Leftrightarrow \neg p_B)$$

Lako se može dokazati da je navedena formula slabo ekvivalentna sa formulom F. Na kraju, navedena formula se trivijalno transformiše u KNF oblik primenom pravila iz tabele 8.1. Svaki konjunkt se transformiše u KNF formulu sa najviše četiri klauze, od kojih svaka ima najviše tri literala.

Cejtinova transformacija daje formulu čija veličina je linearna u odnosu na veličinu polazne formule. Preciznije, ako polazna formula sadrži n logičkih veznika, onda će izlazna formula sadržati najviše $4n$ klauza, od kojih svaka ima najviše tri literala. To se može pokazati jednostavnim induktivnim argumentom.

Primer 8.21. *Data je iskazna formula* $(p \wedge (q \wedge r)) \vee ((q \wedge r) \wedge \neg p)$. *Definicione promenljive* p_4, p_5, p_6, p_7, p_8 *uvode se na sledeći način:*

$$\overbrace{(p \wedge \underbrace{(q \wedge r)}_{p_4})}^{p_6} \vee \overbrace{(\underbrace{(q \wedge r)}_{p_4} \wedge \underbrace{\neg p}_{p_5})}^{p_7}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{p_8}$$

Međuoblik za Cejtinovu formu je onda:

$$p_8 \wedge (p_8 \Leftrightarrow (p_6 \vee p_7)) \wedge (p_6 \Leftrightarrow (p \wedge p_4)) \wedge$$
$$(p_7 \Leftrightarrow (p_4 \wedge p_5)) \wedge (p_4 \Leftrightarrow (q \wedge r)) \wedge (p_5 \Leftrightarrow \neg p))$$

Konačno, izlazna KNF *formula je:*

$p_8 \wedge$
$(\neg p_8 \vee p_6 \vee p_7) \wedge (p_8 \vee \neg p_6) \wedge (p_8 \vee \neg p_7) \wedge$
$(p_6 \vee \neg p \vee \neg p_4) \wedge (\neg p_6 \vee p) \wedge (\neg p_6 \vee p_4) \wedge$
$(p_7 \vee \neg p_4 \vee \neg p_5) \wedge (\neg p_7 \vee p_4) \wedge (\neg p_7 \vee p_5) \wedge$
$(p_4 \vee \neg q \vee \neg r) \wedge (\neg p_4 \vee q) \wedge (\neg p_4 \vee r) \wedge$
$(p_5 \vee p) \wedge (\neg p_5 \vee \neg p)$

Problem sa Cejtinovom transformacijom je u tome što ona uvodi mnogo novih promenljivih. Postoje raznovrsne tehnike za smanjivanje broja promenljivih i broja klauza.

8.3.3 Problem SAT i DPLL procedura

Za svaku iskaznu formulu postoji njena konjunktivna normalna forma i većina primena iskazne logike svodi se na ispitivanje zadovoljivosti neke formule koja je u tom, KNF obliku.

Problem ispitivanja zadovoljivosti date iskazne formule u KNF obliku označava se sa SAT (od engleskog *satisfiability problem* — problem zadovoljivosti). Programi koji rešavaju instance SAT problema zovu se SAT rešavači. SAT problem je NP-kompletan i on ima ogroman i teorijski i praktični značaj.

Kako se još uvek ne zna da li su klase P i NP problema jednake, još uvek se ne zna ni da li postoji algoritam za ispitivanje zadovoljivosti iskaznih formula koji je polinomske složenosti.[3] Kako je opšte uverenje da su klase problema P i NP različite, veruje se i da ne postoji algoritam polinomske složenosti za rešavanje SAT problema.

Problem ispitivanja zadovoljivosti formula u DNF obliku suštinski je drugačiji od ispitivanja zadovoljivosti formula u KNF obliku. Drugi je NP-kompletan, a prvi je trivijalan (dovoljno je razmatrati zadovoljivost disjunkata

[3] Kada se govori o klasama složenosti, obično se podrazumeva da se složenost algoritma izražava u terminima broja bitova potrebnih za zapisivanje ulaza.

pojedinačno, a to se svodi na provere da li u disjunktu postoji logička konstanta $\bot$ ili literal i njegova negacija) i pripada klasi P. Ipak, svođenje problema SAT na problem ispitivanja zadovoljivosti DNF formule nije, u opštem slučaju, razuman put za rešavanje problema SAT, zbog kompleksnosti same transformacije, na primer, formula koje su već u KNF obliku u DNF oblik. Dodajmo da i problem ispitivanja tautologičnosti formule u KNF obliku pripada klasi P (dovoljno je razmatrati tautologičnost konjunkata pojedinačno, a to se svodi na provere da li u konjunktu postoji logička konstanta $\top$ ili literal i njegova negacija).

Dejvis–Patnam–Logman–Lavlendova ili DPLL procedura[4] je procedura za ispitivanje zadovoljivosti iskaznih formula u KNF obliku, to jest, procedura za rešavanje instanci SAT problema. Ulazna formula je konjunkcija klauza. Pri tome (kako su konjunkcija i disjunkcija komutativne i asocijativne) nije bitan poredak tih klauza niti je u bilo kojoj od tih klauza bitan poredak literala, te se ulazna formula može smatrati skupom (ili, preciznije, multiskupom[5]) klauza, od kojih se svaka može smatrati skupom (ili, preciznije, multiskupom) literala. Ipak, radi određenosti rada algoritma, smatraćemo da je skup (odnosno multiskup) klauza uređen.

U proceduri se podrazumevaju sledeće konvencije:

- prazan skup klauza je zadovoljiv;
- klauza koja ne sadrži nijedan literal (zvaćemo je *prazna klauza*) je nezadovoljiva i formula koja sadrži praznu klauzu je nezadovoljiva.

DPLL procedura prikazana je na slici 8.2, a njena svojstva daje teorema 8.6.

Teorema 8.6 (Korektnost DPLL procedure)**.** *Za svaku iskaznu formulu DPLL procedura se zaustavlja i vraća odgovor* DA *ako i samo ako je polazna formula zadovoljiva.*

Primer 8.22. *Formula iz primera 8.13 trivijalno se može transformisati u* KNF *oblik i na nju se može primeniti DPLL procedura. Prvo pravilo koje je primenljivo je* split *i može da se primeni, na primer, na promenljivu p_{a1}. U prvoj grani koja se razmatra, p_{a1} se zamenjuje sa $\top$ (što odgovara pridruživanju vrednosti* tačno*) i u narednim koracima se, primenom pravila* unit propagation *promenljive p_{a2}, p_{a3}, p_{b1}, p_{c1}, p_{b2}, p_{c3} zamenjuju sa $\bot$. Zatim se promenljive p_{b3} i p_{c2} zamenjuju sa $\top$, nakon čega klauza $\neg p_{b3} \vee \neg p_{c2}$ postaje prazna. Slično se dešava i u grani u kojoj se p_{a1} zamenjuje sa $\bot$, te procedura vraća odgovor NE, što znači da ne postoji rešenje problema n dama za $n = 3$.*

Procedura DPLL je u najgorem slučaju eksponencijalne vremenske složenosti po broju iskaznih promenljivih u formuli, usled rekurzivne primene pravila *split*. Eksponencijalne složenosti su i svi drugi do sada poznati algoritmi za ispitivanje zadovoljivosti. Ipak, svi ti algoritmi znatno su efikasniji od metode istinitosnih tablica.

Procedura DPLL može se smatrati algoritmom pretrage potpunog stabla valuacija promenljivih koje učestvuju u formuli. Radi se o pretraživanju bektrekingom, pošto detekcija prazne klauze i „koraci zaključivanja“ (*tautology*, *unit propagation* i *pure literal*) omogućavaju da se ne pretražuje nužno čitavo stablo. Na efikasnost pretrage utiču i heuristike koje je usmeravaju biranjem načina na koji se primenjuje pravilo *split*: treba izabrati jednu promenljivu i treba odlučiti da li se prvo izvršava grana DPLL($D[p \mapsto \top]$) ili grana DPLL($D[p \mapsto \bot]$) (u algoritmu prikazanom na slici 8.2 poredak je fiksiran zbog jednostavnosti). Pošto se ispituje da li postoji valuacija u kojoj su sve klauze formule tačne, pohlepni algoritam bi mogao da za *split* promenljivu bira onu čijim se pogodnim instanciranjem dobija najveći broj tačnih klauza u tekućoj parcijalnoj valuaciji. Druge varijante ovog pravila su da se bira iskazna promenljiva sa najviše pojavljivanja u tekućoj formuli, da se bira neko od iskaznih slova iz najkraće klauze itd. Ove heuristike, kao ni druge slične, ne garantuju optimalnost procesa pretrage.

Primer 8.23. *Neka je potrebno ispitati zadovoljivost formule date klauzama:*

C_1 : $\neg a, \neg b, c$
C_2 : $a, \neg b$
C_3 : b, c
C_4 : $\neg b, \neg c$

Formula ima dve zadovoljavajuće valuacije. Proverom zadovoljivosti procedurom DPLL, pronalazi se jedna od te dve valuacije. Prvo stablo na slici 8.3 prikazuje proces pretrage u slučaju datog skupa klauza. Kako obe zadovoljavajuće valuacije pridružuju promenljivoj b vrednost 0, a promenljivoj c vrednost 1, nakon dodavanja klauze

[4] Prva verzija procedure čiji su autori Dejvis (Martin Davis) i Patnam (Hilary Putnam), unapređena je dve godine kasnije u radu Dejvisa, Logmana (Georg Logemann) i Lavlenda (Donald Loveland), pa otuda naziv DPLL.

[5] Neformalno, multiskup je skup u kojem se elementi mogu pojavljivati više puta.

Algoritam: DPLL

Ulaz: Multiskup klauza D ($D = \{C_1, C_2, \ldots, C_n\}$)

Izlaz: *DA*, ako je multiskup D zadovoljiv, *NE*, inače;

ako D je prazan **onda**
    vrati *DA*;
zameni sve literale $\neg\bot$ sa $\top$ i zameni sve literale $\neg\top$ sa $\bot$;
obriši sve literale jednake $\bot$;
ako D sadrži praznu klauzu **onda**
    vrati *NE*;

{Korak *tautology*:}
ako neka klauza C_i sadrži $\top$ ili sadrži neki literal i njegovu negaciju **onda**
    vrati vrednost koju vraća DPLL($D \setminus C_i$);

{Korak *unit propagation*:}
ako neka klauza je jedinična i jednaka nekom iskaznom slovu p **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \top]$);
ako neka klauza je jedinična i jednaka $\neg p$, za neku iskaznu promenljivu p **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$);

{Korak *pure literal*:}
ako D sadrži literal p (gde je p neka iskazna promenljiva), ali ne i $\neg p$ **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \top]$) ;
ako D sadrži literal $\neg p$ (gde je p neka iskazna promenljiva), ali ne i p **onda**
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$);

{Korak *split*:}
ako DPLL($D[p \mapsto \top]$) (gde je p jedno od iskaznih slova koja se javljaju u D) vraća *DA* **onda**
    vrati *DA*;
inače
    vrati vrednost koju vraća DPLL($D[p \mapsto \bot]$).

Slika 8.2: DPLL procedura.

$C_5 : b, \neg c$

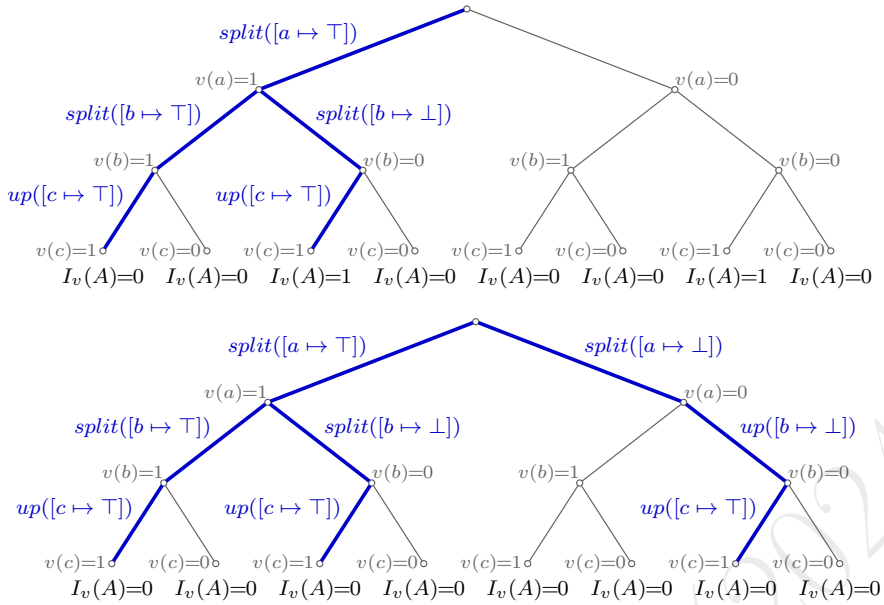
prethodni skup klauza postaje nezadovoljiv. Proces pretrage procedurom DPLL u ovom slučaju, prikazan je na drugom stablu na istoj slici. U ovom primeru upečatljivo je da DPLL procedura ispituje svega tri od osam listova zahvaljujući tome što osim koraka pretrage oličenih u pravilu split*, postoje i koraci zaključivanja koje se vrši primenom pravila* unit propagation *(kada se ne ispituju alternative zamena učinjenih tim pravilom).*

DPLL procedura proverava da li je formula zadovoljiva, ali ona se, kao što je već rečeno, može koristiti i za ispitivanje da li je neka formula valjana, poreciva ili kontradikcija. Na primer, formula A je valjana ako i samo ako je formula $\neg A$ nezadovoljiva, što se može proveriti DPLL procedurom (pri čemu je, naravno, formulu $\neg A$ potrebno najpre transformisati u konjunktivnu normalnu formu).

8.4 Rešavanje problema svođenjem na SAT

Mnogi praktični problemi mogu se rešiti korišćenjem iskazne logike. Obično je postupak rešavanja ovakav:

- elementarni iskazi (tvrdnje) koji figurišu u opisu problema, predstavljaju se iskaznim promenljivim (u duhu nekog *kodiranja*);
- uslovi problema se predstavljaju iskaznim formulama nad tim iskaznim promenljivim;



Slika 8.3: Proces provere zadovoljivosti procedurom DPLL prikazan u vidu pretrage u potpunom stablu valuacija za dva skupa klauza. Pretraga se vrši obilaskom stabla u dubinu sleva nadesno. U prvom slučaju, formula A data je skupom klauza $\{\neg a \vee \neg b \vee c, a \vee \neg b, b \vee c, \neg b \vee \neg c\}$ koji je zadovoljiv. U drugom slučaju, skupu klauza dodata je i klauza $b \vee \neg c$ i dobijeni skup je nezadovoljiv.

- konjunkcija tih iskaznih formula transformiše se u konjunktivnu normalnu formu;
- zadovoljivost formule u konjunktivnoj normalnoj formi ispituje se SAT rešavačem;
- ukoliko je formula zadovoljiva, svaki njen model daje jedno rešenje polaznog problema.

Svođenjem na SAT mogu se pogodno opisati mnogi problemi nad konačnim domenima. Naredni primer pokazuje kako sabiranje prirodnih brojeva, a i rešavanje jednačina koje uključuju takvo sabiranje mogu biti svedeni na SAT.

Primer 8.24. *Neka su u i v prirodni brojevi manji od 4. Onda, ako je broj u predstavljen parom iskaznih promenljvih (p, q) (koje odgovaraju njegovim ciframa u binarnom zapisu) a broj v predstavljen parom (r, s), onda je broj $u + v$ (po modulu 2^2) predstavljen parom $((p \veebar r) \veebar (q \wedge s), q \veebar s)$ (gde $\veebar$ označava ekskluzivnu disjunkciju).*

Primer 8.25. *Neka je zadat problem određivanja vrednosti u, ako je poznato da je $v = 2$ i $v = u + 1$ (po modulu 4). Broj 1 može se predstaviti parom $(\bot, \top)$ i, kako je poznato da važi $v = 2$ i $v = u + 1$, onda se v može predstaviti i kao $(\top, \bot)$ i kao $((p \veebar \bot) \veebar (q \wedge \top), q \veebar \top)$, i, nakon pojednostavljivanja, sa $(p \veebar q, \neg q)$. Da bi se dobila formula koja odgovara zadatim uslovima i iz koje se može dobiti vrednost broja u, formule na obe pozicije moraju da budu ekvivalentne i sledeća formula mora biti zadovoljiva: $((p \veebar q) \Leftrightarrow \top) \wedge (\neg q \Leftrightarrow \bot)$. Ona je zadovoljiva i ima samo jedan model. U tom modelu promenljiva p ima vrednost 0 a promenljiva q ima vrednost 1. Dakle, nepoznata vrednost u ima binarni zapis 01, pa je ona jednaka 1.*

Navedeni primer je trivijalan, ali ilustruje kako se rešavanje jednačina nad prirodnim brojevima može svesti na SAT. Analogno se mogu rešavati i mnogo kompleksniji problemi.

Rešavanje problema svođenjem na SAT biće ilustrovano kroz nekoliko različitih vrsta svođenja na SAT i nekoliko konkretnih primera.

8.4.1 Primeri kodiranja ograničenja

Retko kodiranje. U praktičnim problemima koji se rešavaju svođenjem na SAT, ne figurišu samo iskazne promenljive, već često i celobrojne promenljive koje mogu imati vrednosti iz nekog ograničenog skupa. U takvim situacijima često se koristi *retko kodiranje* (eng. *sparse encoding*) u kojem se uvode iskazne promenljive poput

$p_{a,i}$, koja je tačna ako i samo ako promenljiva a ima vrednost i. Time se uslov da promenljiva a ima jednu vrednost iz zadatog domena I zadaje uslovom (uslov „barem jedna“):

$$\bigvee_{i \in I} p_{a,i}$$

Promenljiva a ne može imati dve vrednosti istovremeno, što se opisuje formulom (uslov „najviše jedna“):

$$\bigwedge_{i,j \in I, i \neq j} \neg p_{a,i} \vee \neg p_{a,j}$$

Pored uslova koji su potrebni kako bi se iskazalo da promenljiva ima (tačno jednu) vrednost iz nekog konačnog skupa, često je potrebno kodirati i razna druga ograničenja. Za neke česte tipove ograničenja postoje ustaljeni načini kodiranja u okviru retkog kodiranja.

U nekim problemima potrebno je onemogućiti neku kombinaciju vrednosti promenljivih iz nekog skupa S. Kombinacija vrednosti može biti zadata parovima $(a, f(a))$ za sve elemente a iz S, tj. uslovom $\wedge_{a \in S} p_{a,f(a)}$. Tada se za svaku nedozvoljenu kombinaciju vrednosti uvodi takozvana „klauza konflikta”:

$$\neg \bigwedge_{a \in S} p_{a,f(a)}$$

to jest

$$\bigvee_{a \in S} \neg p_{a,f(a)}.$$

Ovakve klauze mogu se koristiti i kada je potrebno pronaći više rešenja ili sva rešenja nekog problema. Naime, kada se pronađe jedno rešenje, onda polaznoj formuli treba dodati klauzu koja zabranjuje tu kombinaciju vrednosti (zovemo je i „blokirajuća klauza“) i onda rešiti novodobijenu SAT instancu. Postupak se analogno nastavlja i kada je pronađeno nekoliko rešenja.

Ograničenja oblika „ako a ima vrednost i, onda b mora imati neku od vrednosti iz skupa I“ obično se opisuju formulama sledećeg oblika:

$$\neg p_{a,i} \vee \bigvee_{j \in I} p_{b,j}.$$

Logaritamsko kodiranje. U *logaritamskom kodiranju* (eng. *log encoding*) svakom bitu vrednosti numeričkih promenljivih (zapisanih u binarnoj reprezentaciji) pridružuje se jedna iskazna promenljiva. U ovoj reprezentaciji ne postoji potreba za uslovima „barem jedna“ i „najviše jedna“, jer svaka valuacija uvedenih iskaznih promenljivih daje tačno jednu vrednost odgovarajuće promenljive. Naravno, kada je broj mogućih vrednosti numeričke promenljive manji od broja mogućih vrednosti iskaznih promenljivih koji se koriste za njeno kodiranje, neke valuacije iskaznih promenljivih potrebno je zabraniti dodatnim klauzama (na primer, ako promenljiva n može da ima vrednosti od 0 do 6, za njeno kodiranje se koriste tri iskazne promenljive, ali se zabranjuje valuacija koja daje vrednost 7).

I u logaritamskom kodiranju moguće je izraziti uslove koje u slučaju retkih kodiranja izražavaju direktno i potporno kodiranje, ali zbog prirode logaritamskog kodiranja, te uslove potrebno je zadati nad binarnim kombinacijama koje predstavljaju vrednosti numeričkih promenljivih. Na primer, neka promenljive a i b uzimaju celobrojne vrednosti od 0 do 7 i neka su kodirane iskaznim promenljivim $p_{a,0}$, $p_{a,1}$, $p_{a,2}$, $p_{b,0}$, $p_{b,1}$ i $p_{b,2}$, pri čemu viši indeksi označavaju bitove veće težine. Ukoliko se vrednost 3 promenljive a uzajamno isključuje sa vrednošću 6 promenljive b, taj uslov može se kodirati u terminima bitova, klauzom

$$p_{a,2} \vee \neg p_{a,1} \vee \neg p_{a,0} \vee \neg p_{b,2} \vee \neg p_{b,1} \vee p_{b,0}.$$

Primer 8.26. *Zadatak je obojiti dve kuće (neka su označene sa a i b) po jednom od tri raspoložive boje (neka su označene brojevima 1, 2, 3), ali tako da budu obojene različito. Neka iskazna promenljiva* $p_{a,1}$ *predstavlja tvrdnju da je kuća a obojena bojom* 1*, neka* $p_{a,2}$ *predstavlja tvrdnju da je kuća a obojena bojom* 2*, ... i neka* $p_{b,3}$ *predstavlja tvrdnju da je kuća b obojena bojom* 3*.*

U retkim kodiranjima problema biće potrebni uslovi „barem jedna“:

$p_{a,1} \vee p_{a,2} \vee p_{a,3}$

$p_{b,1} \vee p_{b,2} \vee p_{b,3}$

i uslovi „najviše jedna“:

$\neg p_{a,1} \vee \neg p_{a,2}$

$\neg p_{a,1} \vee \neg p_{a,3}$

$\neg p_{a,2} \vee \neg p_{a,3}$

$\neg p_{b,1} \vee \neg p_{b,2}$
$\neg p_{b,1} \vee \neg p_{b,3}$
$\neg p_{b,2} \vee \neg p_{b,3}$

Dodatno, u direktnom kodiranju biće opisan i uslov da nisu obe kuće obojene istom bojom (klauze konflikta):

$\neg p_{a,1} \vee \neg p_{b,1}$
$\neg p_{a,2} \vee \neg p_{b,2}$
$\neg p_{a,3} \vee \neg p_{b,3}$

Alternativno, navedene klauze konflikta bi mogle, korišćenjem potpornog kodiranja, da se zadaju na sledeći način:

$\neg p_{a,1} \vee (p_{b,2} \vee p_{b,3})$
$\neg p_{a,2} \vee (p_{b,1} \vee p_{b,3})$
$\neg p_{a,3} \vee (p_{b,1} \vee p_{b,2})$
$\neg p_{b,1} \vee (p_{a,2} \vee p_{a,3})$
$\neg p_{b,2} \vee (p_{a,1} \vee p_{a,3})$
$\neg p_{b,3} \vee (p_{a,1} \vee p_{a,2})$

Pažljivom analizom može se pokazati da poslednje tri navedene klauze nisu potrebne.

Problem može biti opisan i korišćenjem logaritamskog kodiranja: neka iskazne promenljive $p_{a,0}$, $p_{a,1}$ odgovaraju ciframa binarnog zapisa boje koja odgovara kući a, a $p_{b,0}$, $p_{b,1}$ odgovaraju ciframa binarnog zapisa boje koja odgovara kući b, pri čemu viši indeksi označavaju bitove veće težine. Ako su bojama pridružene vrednosti 0, 1 i 2, treba zabraniti vrednost 3, pa zato postoje uslovi:

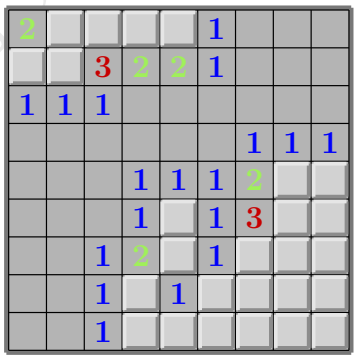
$\neg p_{a,1} \vee \neg p_{a,0}$
$\neg p_{b,1} \vee \neg p_{b,0}$

Klauze konflikta su sledeće klauze:

$p_{a,1} \vee p_{a,0} \vee p_{b,1} \vee p_{b,0}$ *(nisu obe boje 0)*
$p_{a,1} \vee \neg p_{a,0} \vee p_{b,1} \vee \neg p_{b,0}$ *(nisu obe boje 1)*
$\neg p_{a,1} \vee p_{a,0} \vee \neg p_{b,1} \vee p_{b,0}$ *(nisu obe boje 2)*

8.4.2 Igra „čistač mina“

Razmotrimo kako iskazna logika može pomoći u igranju računarske igre „čistač mina“ (eng. *minesweeper*, slika 8.4).



Slika 8.4: Igra „čistač mina“.

Data je tabla dimenzija $n \times m$. Kada igrač izabere jedno zatvoreno polje na tabli, igra je završena ako se pokaže da je na njoj mina. Inače, igrač saznaje ukupan broj mina na susednim poljima. Ukoliko je taj broj jednak 0, onda se on ne ispisuje a rekurzivno se otvaraju sva susedna polja (jer je jasno da na njima nema mina). Cilj igre je otvoriti sva polja na kojima nema mina. Pokazaćemo da korišćenjem iskazne logike možemo da dobijamo zaključke potrebne za uspešno igranje igre. Pridružimo iskazne promenljive p, q, r, s, t, u poljima u levom gornjem uglu table (u nastavku ćemo imenima ovih promenljivih nazivati i polja kojima su pridružena):

p	q	r	
s	t	u	

Pretpostavimo da smo otvorili polje p i da na njemu nema mine. Pretpostavimo i da smo za polje p dobili broj 1. Opišimo ovu situaciju sredstvima iskazne logike. Neka stanje table (stvarno stanje, ne samo otkriveni deo) određuje valuaciju uvedenih promenljivih: na nekom polju postoji mina ako i samo ako odgovarajuća promenljiva ima vrednost 1 u toj valuaciji. Kako znamo da na polju p nema mina, u valuaciji v koja odgovara stanju table, mora da je tačna formula $\neg p$. Kako je na polju p broj 1, to znači da je neka mina ili na polju q ili na polju s ili na polju t. Ove uslove opisuju formule:

$q \vee s \vee t$,
$q \Rightarrow \neg s \wedge \neg t$,
$s \Rightarrow \neg q \wedge \neg t$,
$t \Rightarrow \neg q \wedge \neg s$

Iz raspoloživog znanja nije moguće odrediti na kojem od tri polja q, s, t je mina, te moramo da rizikujemo. Pretpostavimo da smo otvorili polje q, da na njemu nema mine i da smo dobili opet broj 1.

1	1	r	
s	t	u	

Pokažimo da sada bezbedno možemo da otvorimo polja r i u. Kako je na polju q broj 1, to znači da je neka mina ili na polju p ili na polju s ili na polju t ili na polju r ili na polju u. Ove uslove opisuju formule:

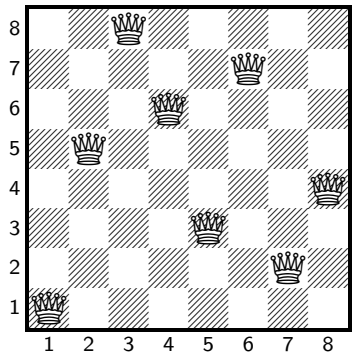
$p \vee s \vee t \vee r \vee u$,
$p \Rightarrow \neg s \wedge \neg t \wedge \neg r \wedge \neg u$,
$s \Rightarrow \neg p \wedge \neg t \wedge \neg r \wedge \neg u$,
$t \Rightarrow \neg p \wedge \neg s \wedge \neg r \wedge \neg u$,
$r \Rightarrow \neg p \wedge \neg s \wedge \neg t \wedge \neg u$,
$u \Rightarrow \neg p \wedge \neg s \wedge \neg t \wedge \neg r$.

Označimo sa Φ konjunkciju svih navedenih formula, zajedno sa formulama $\neg p$ i $\neg q$ (koje govore da nema mina na poljima p i q). Pitanje da li možemo bezbedno da otvorimo polje r svodi se onda na pitanje da li formula Φ ima za logičku posledicu formulu $\neg r$ (koja govori da nema mine na polju r). To pitanje se dalje svodi na pitanje da li je formula $\Phi \Rightarrow \neg r$ tautologija i, konačno, na pitanje da li je formula $\Phi \wedge r$ nezadovoljiva. SAT rešavač može lako da utvrdi da ta formula jeste nezadovoljiva, te mina ne može biti na polju r. Analogno se može zaključiti da mina ne može biti na polju u.

U ovom konkretnom slučaju, zaključak se izvodi lako, ali se opisani mehanizam može uspešno koristiti i u mnogo komplikovanijim situacijama.

8.4.3 Problem n dama

Za svako konkretno n, analogno kao u slučaju $n = 3$ (primer 8.1), problem n dama (slika 8.5) može se rešiti korišćenjem iskazne logike. Primenimo kodiranje u stilu retkog kodiranja: neka svakom polju (i, j) šahovske table odgovara jedna iskazna promenljiva p_{ij} (neka prvi indeks predstavlja redni broj kolone, analogno šahovskoj notaciji). Neka promenljiva p_{ij} ima vrednost 1 ako je na polju (i, j) neka dama, a 0 inače. Zadata ograničenja o nenapadanju dama mogu se onda, korišćenjem direktnog kodiranja, opisati na sledeći način:



Slika 8.5: Jedno rešenje za problem 8 dama.

1. u svakoj koloni mora da bude barem jedna dama:

$\bigvee_{i=1,\ldots,n} p_{ki}$, za $1 \leq k \leq n$;

2. u svakoj koloni može da bude najviše jedna dama:

   $\bigwedge_{i=1,\ldots,n-1;j=i+1,\ldots,n} \neg p_{ki} \vee \neg p_{kj}$, za $1 \leq k \leq n$;

3. u svakoj vrsti može da bude najviše jedna dama:

   $\bigwedge_{i=1,\ldots,n-1;j=i+1,\ldots,n} \neg p_{ik} \vee \neg p_{jk}$, za $1 \leq k \leq n$;

4. nema dama koje se napadaju dijagonalno:

   $\bigwedge_{i=1,\ldots,n;j=1,\ldots,n;k=1,\ldots,n;l=1,\ldots,n} \neg p_{ij} \vee \neg p_{kl}$, za i, j, k, l za koje važi $|k - i| = |l - j|$.

Naglasimo sledeće: kako prva dva skupa uslova obezbeđuju da ima ukupno n dama, a treći da u svakoj vrsti ima najviše jedna dama, nije potrebno zadavati i uslov da u svakoj vrsti mora da bude barem jedna dama. Moguće je, kao redundantne, eliminisati i neke od uslova iz četvrtog skupa (na primer, tako da ne postoje pojedinačni međusobno ekvivalentni uslovi).

Konjunkcija navedenih uslova daje formulu koja opisuje zadati problem. Ona je već u konjunktivnoj normalnoj formi i njena zadovoljivost može biti ispitana nekim SAT rešavačem. Na primer, za $n = 8$, formula ima 92 modela i svaki od njih daje po jedno raspoređivanje dama koje ispunjava date uslove.

8.4.4 Raspoređivanje sportskih utakmica

Iskazna logika često se koristi u problemima raspoređivanja. Jedan od takvih problema je raspoređivanje sportskih utakmica. Pretpostavićemo da se koristi kružni sistem takmičenja po principu „igra svako sa svakim“ koji se karakteriše sledećim uslovima:

1. Postoji n timova (n je paran broj) i svaka dva tima jednom igraju jedan protiv drugog.
2. Sezona traje $n - 1$ nedelja.
3. Svaki tim svake nedelje igra jednu utakmicu.
4. Postoji $n/2$ terena i svake nedelje se na svakom terenu igra jedna utakmica.
5. Nijedan tim ne igra više od dva puta na istom terenu.

Neka su timovi označeni brojevima od 1 do 10. Primer ispravnog rasporeda dat je u tabeli 8.2.

nedelja / teren	1	2	3	4	5	6	7	8	9
1	6-9	4-6	1-8	4-10	2-8	7-9	5-7	1-2	3-5
2	2-3	1-5	2-4	1-7	9-10	8-10	3-6	4-9	6-8
3	5-10	2-7	3-9	5-9	1-3	1-6	4-8	6-10	4-7
4	1-4	8-9	5-6	3-8	6-7	2-5	1-10	3-7	2-10
5	7-8	3-10	7-10	2-6	4-5	3-4	2-9	5-8	1-9

Tabela 8.2: Primer ispravnog rasporeda za 10 timova.

Osnovni iskaz relevatan za sastavljanje rasporeda je da „tim k_1 igra protiv tima k_2 na terenu i u nedelji j“. Kako je u nekim ograničenjima potrebno govoriti o pojedinačnim timovima, a ne samo o parovima, ovo tvrđenje neće biti predstavljeno jednom promenljivom, nego dvema. Promenljiva p_{ij}^{1l} označava da tim l igra (protiv nekog tima) na terenu i u nedelji j, kao prvi tim u paru. Promenljiva p_{ij}^{2l} označava da tim l igra (protiv nekog tima) na terenu i u nedelji j, kao drugi tim u paru. Stoga, skup promenljivih je:

$$\{p_{ij}^{1l} \mid 1 \leq i \leq n/2, 1 \leq j \leq n-1, 1 \leq l \leq n\} \cup \{p_{ij}^{2l} \mid 1 \leq i \leq n/2, 1 \leq j \leq n-1, 1 \leq l \leq n\}$$

Dakle, ukupno se koristi $2 \cdot (n/2) \cdot (n-1) \cdot n = n^2(n-1)$ promenljivih. Raspored čini skup parova $(p_{ij}^{1k_1}, p_{ij}^{2k_2})$ koji izražavaju iskaze „tim k_1 igra protiv tima k_2 na terenu i u nedelji j“. Ograničenja se izražavaju formulom koja predstavlja konjunkciju sledećih klauza:

1. na svakoj utakmici je redni broj prvog tima manji od rednog broja drugog tima:

   $$\neg p_{ij}^{1k_1} \vee \neg p_{ij}^{2k_2}$$

   za sve $1 \leq i \leq n/2$, $1 \leq j \leq n-1$, $1 \leq k_1, k_2 \leq n$, takve da važi $k_1 \geq k_2$;

2. na svakom terenu svake nedelje održava se neka utakmica:

$$p_{ij}^{11} \vee \ldots \vee p_{ij}^{1n}$$

za sve $1 \leq i \leq n/2$, $1 \leq j \leq n-1$ (dovoljno je obezbediti da svake nedelje na svakom terenu igra barem jedan tim);

3. nijedan tim nijedne nedelje ne igra više od jedne utakmice:

$$\neg p_{i_1 j}^{r_1 l} \vee \neg p_{i_2 j}^{r_2 l}$$

za sve $1 \leq i_1, i_2 \leq n/2$, $1 \leq j \leq n-1$, $1 \leq r_1, r_2 \leq 2$, $1 \leq l \leq n$, za koje važi $i_1 \neq i_2$ ili $r_1 \neq r_2$;

4. dva različita tima sastaju se najviše jednom:

$$\neg p_{i_1 j_1}^{1 k_1} \vee \neg p_{i_1 j_1}^{2 k_2} \vee \neg p_{i_2 j_2}^{1 k_1} \vee \neg p_{i_2 j_2}^{2 k_2}$$

za sve $1 \leq i_1, i_2 \leq n/2$, $1 \leq j_1, j_2 \leq n-1$, $1 \leq k_1, k_2 \leq n$, takve da važi $j_1 \neq j_2$ i $k_1 < k_2$;

5. nijedan tim ne igra više od dva puta na istom terenu:

$$\neg p_{i j_1}^{r_1 k} \vee \neg p_{i j_2}^{r_2 k} \vee \neg p_{i j_3}^{r_3 k}$$

za sve $1 \leq i \leq n/2$, $1 \leq j_1, j_2, j_3 \leq n-1$, $1 \leq k \leq n$, $1 \leq r_1, r_2, r_3 \leq 2$, takve da važi $j_1 \neq j_2$, $j_1 \neq j_3$ i $j_2 \neq j_3$.

Ukupan broj klauza u formuli je reda $O(n^6)$ (tog reda je broj klauza koje opisuje 4. korak). Iako je, i za relativno male vrednosti n, taj broj klauza veoma veliki, savremeni SAT rešavači uspešno mogu ispitati zadovoljivost odgovarajućih formula.

Pažljivom analizom može se pokazati da su neke od navedenih klauza redundantne. Ipak, ukoliko sâm proces rešavanja nije previše zahtevan, racionalno je modelovanje održavati jednostavnim i lakim za razumevanje.

8.4.5 Verifikacija softvera

Zamislimo da se u nekoj softverskoj kompaniji radi na reviziji postojećeg koda i, kako je on veoma važan, potrebno ga je unaprediti – učiniti ga efikasnijim i čitljivijim. Nekome se čini da je narednu funkciju `f` napisanu na programskom jeziku C:

```
int f(int y)
{
   int x;
   x = x ^ y;
   y = x ^ y;
   x = x ^ y;
   return x * x;
}
```

moguće zameniti narednom funkcijom `g`:

```
int g(int n)
{
  return n * n;
}
```

Pitanje je, dakle, da li ove dve funkcije za iste argumente uvek vraćaju iste rezultate. Pogodno je u razmatranje uvesti i promenljive za međurezultate (jer programska promenljiva u toku izvršavanja programa može da menja vrednost, a iskazna promenljiva u okviru jedne formule uvek ima istu vrednost):

```
x1 = x ^ y;
y1 = x1 ^ y;
x2 = x1 ^ y1;
```

Kako funkcija `g` vraća vrednost `n * n`, gde je `n` nepromenjena vrednost parametra, dovoljno je dokazati da za funkciju `f` važi da pre naredbe `return x * x;` promenljiva `x` ima istu vrednost kao parametar `y`, tj. da je vrednost `x2` jednaka vrednosti `y`, tj. da važi `x2 == y`. Ako se iskoristi način na koji su zadate vrednosti međurezultata, ova jednakost postaje:

```
(x ^ y) ^ ((x ^ y) ^ y) == y
```

Ako pretpostavimo da tip `int` ima širinu četiri bajta, programske promenljive možemo da predstavimo nizovima od po $4 \cdot 8 = 32$ iskazne promenljive. Onda izrazu `(x ^ y) ^ ((x ^ y) ^ y)` odgovara niz od 32 iskazne formule i za dokazivanje navedene jednakosti potrebno je dokazati da je svaka od 32 formule koja odgovara izrazu `(x ^ y) ^ ((x ^ y) ^ y)` logički ekvivalentna odgovarajućoj iskaznoj promenljivoj iz reprezentacije programske promenljive `y`. U ovom konkretnom slučaju (a nije tako uvek), može se primetiti da bitovski operator `^` (operator ekskluzivne disjunkcije) primenjen na dve vrednosti dejstvuje samo bit-po-bit, tj. rezultat operacije primenjen na neka dva bita operanada ne utiče na druge bitove rezultata. Zato se razmatranje navedene jednakosti može svesti na razmatranje pojedinačnih bitova, te je dovoljno dokazati logičku ekvivalenciju ($\underline{\vee}$ označava veznik ekskluzivne disjunkcije):

$$(p \underline{\vee} q) \underline{\vee} ((p \underline{\vee} q) \underline{\vee} q) \equiv q$$

Kako je formula $(p \underline{\vee} q) \underline{\vee} ((p \underline{\vee} q) \underline{\vee} q) \Leftrightarrow q$ zaista tautologija, navedena jednakost uvek je tačna, pa su funkcije `f` i `g` ekvivalentne i jedna može biti zamenjena drugom. (Primetimo da programska promenljiva `x` nije inicijalizovana u okviru funkcije `f`, ali to ipak ne utiče na rezultat rada funkcije.)

8.4.6 Provera ekvivalentnosti kombinatornih kola

Iskazna logika ima primene u automatizaciji dizajna elektronskih kola, koje uključuju simulaciju, minimizaciju i verifikaciju dizajna kola. Vrsta elektronskih kola koja je najpogodnija za primenu metoda iskazne logike su kombinatorna kola — mreže povezanih logičkih elemenata kod kojih vrednosti izlaza zavise isključivo od vrednosti ulaza. Logički elementi predstavljaju elektronske implementacije logičkih veznika. Takođe, navedeno svojstvo važi i za istinitosne vrednosti iskaznih formula — za datu formulu, one zavise isključivo od vrednosti koje valuacija dodeljuje iskaznim promenljivim u toj formuli.

Jedan problem verifikacije hardvera koji je u vezi sa prethodno uočenom analogijom je provera ekvivalentnosti kombinatornih kola. Dva logička kola su ekvivalentna ukoliko za sve kombinacije vrednosti na svojim ulazima, daju iste izlaze.

Ova vrsta verifikacije je korisna u sledećem kontekstu. Kombinatorna kola mogu biti vrlo složena i dizajniraju se u alatima koji podržavaju neki od jezika za opis hardvera kao što su Verilog ili VHDL. Pre nego što se na osnovu kreiranog dizajna pristupi fizičkoj implementaciji logičkog kola, taj dizajn prolazi kroz niz transformacija kojima se vrše optimizacije kola kako bi se uštedelo na njegovoj površini, brzini i slično. Svaki od koraka ovog postupka može biti vrlo složen. Algoritmi na kojima pomenute transformacije počivaju garantuju održanje korektnosti ali, zbog složenosti softvera u kojem su ti algoritmi implementirani, postoji mogućnost da je u nekom koraku napravljena greška i da finalni, optimizovani, dizajn kola više nije ekvivalentan polaznom. Zbog toga je pre fizičke izrade logičkog kola potrebno proveriti ekvivalentnost polaznog i finalnog dizajna kola. Treba primetiti da ustanovljena ekvivalentnost ne garantuje funkcionalnu korektnost kola — to da ono zaista radi ono što bi trebalo. Međutim, i to je moguće ustanoviti proverom ekvivalentosti sa kolom za koje je poznato da je funkcionalno korektno, ukoliko takvo kolo postoji.

Provera ekvivalentnosti kombinatornih kola vrši se tako što se za svaki izlaz kreiraju iskazne formule koje odgovaraju njegovom dizajnu u dva kola. Neka su to formule F i F'. Ukoliko su kola ekvivalentna, za sve kombinacije vrednosti ulaza vrednosti ovih formula su jednake. U terminima iskaznih formula, za svaku valuaciju v mora da važi $I_v(F) = I_v(F')$. Dakle, formula $F \Leftrightarrow F'$ mora biti tautologija, a formula $\neg(F \Leftrightarrow F')$ nezadovoljiva. Zadovoljivost iskazne formule može se proveriti pomoću SAT rešavača. Ukoliko za svaki izlaz rešavač ustanovi da je ovakva formula nezadovoljiva, kola su ekvivalentna, a ukoliko ustanovi da postoji zadovoljavajuća valuacija, ta valuacija predstavlja vrednosti ulaza za koje se izlazi kola razlikuju, što može poslužiti kao polazna tačka u otklanjanju greške.

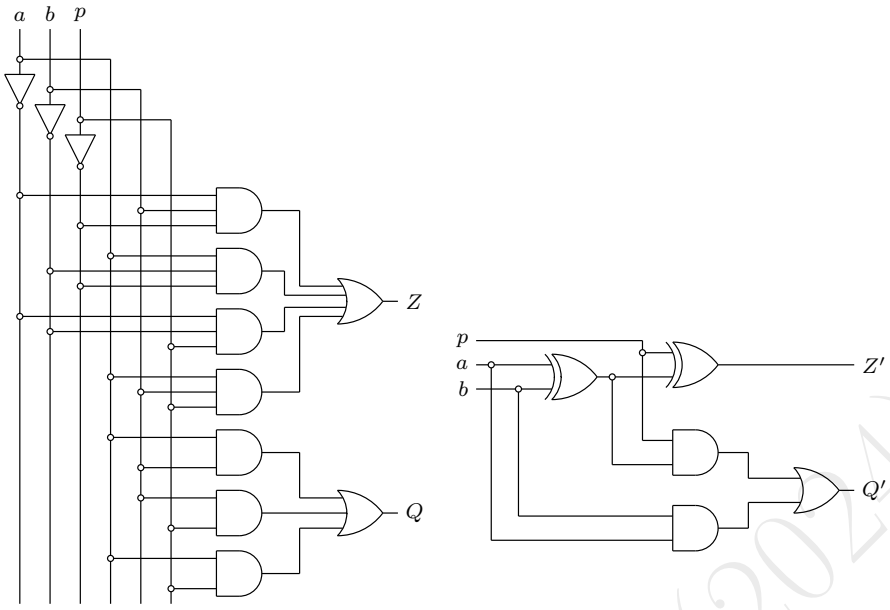
Postupak provere ekvivalentnosti ilustrovaćemo na primeru sabirača (slika 8.6, levo): ulazne vrednosti a i b su cifre brojeva koji se sabiraju, p je prethodni prenos, a na izlazu je formula koja opisuje cifru zbira Z i formula koja opisuje novi prenos Q. Pretpostavimo da je optimizovanjem kola na slici dobijeno drugo kolo (slika 8.6, desno): ulazne vrednosti a i b su cifre brojeva koji se sabiraju, p je prethodni prenos, a na izlazu su formule koje opisuju cifru zbira Z' i novi prenos Q'. Na osnovu dizajna, za svaki izlaz može se kreirati iskazna formula koja mu odgovara:

$$Z = (\neg a \wedge b \wedge \neg p) \vee (a \wedge \neg b \wedge \neg p) \vee (\neg a \wedge \neg b \wedge p) \vee (a \wedge b \wedge p)$$

$$Q = (a \wedge b) \vee (b \wedge p) \vee (a \wedge p)$$

$$Z' = (a \underline{\vee} b) \underline{\vee} p$$

$$Q' = (a \wedge b) \vee (a \underline{\vee} b) \wedge p$$



Slika 8.6: Osnovni i optimizovani dizajn sabirača.

Kola su ekvivalentna ukoliko su formule $\neg(Z \Leftrightarrow Z')$ i $\neg(Q \Leftrightarrow Q')$ nezadovoljive. Treba imati u vidu da formule koje se dobijaju iz ovakvih primena mogu imati i desetine hiljada pa i stotine hiljada promenljivih, ali da SAT rešavači ipak uspevaju da provere njihovu zadovoljivost (pre svega zahvaljujući nekim pravilnostima u strukturi tih formula, a koje se u toku procesa rešavanja mogu naučiti i iskoristiti).

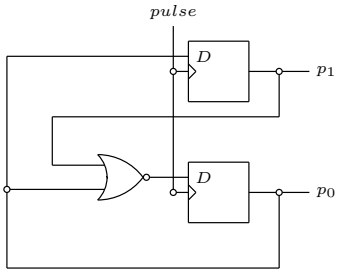
8.4.7 Ograničena provera modela

Jedna od najkorišćenijih tehnika u verifikaciji hardvera i softvera je *ograničena provera modela* (eng. *bounded model checking*). Funkcionisanje hardvera ili softvera se može apstraktno opisati konačnim automatima čija stanja opisuju stanja sistema koji se izučava, a grane moguće prelaske sistema iz stanja u stanje. Takav konačni automat predstavlja model sistema koji se analizira. Jedan od ciljeva verifikacije je dokazivanje da sistem zadovoljava određena svojstva, poput svojstva da nikad neće doći u stanje koje predstavlja grešku ili bezbednosni rizik. Za sistem koji pruža odgovore na zadate zahteve, primer takvog svojstva je da, nakon stanja u kojem je primljen zahtev, sistem sigurno dolazi u neko stanje u kojem će traženi odgovor biti dat. Ispitivanje ovakvih svojstava, koje se naziva *proverom modela*, može predstavljati neodlučiv problem. Stoga se u praksi obično koristi *ograničena provera modela* koja se svodi na dokazivanje da neko svojstvo važi u svim stanjima u koja se iz polaznog stanja može dospeti u najviše k prelaza.

Da bi se sprovela ograničena provera modela tehnikom svođenja na SAT, prvo je potrebno uočiti kako se stanja mogu modelovati iskaznim promenljivim. Potom je potrebno iskaznim formulama nad tim promenljivim zapisati željeno svojstvo, polazne pretpostavke i način na koji se promenljive menjaju prilikom prelaska iz stanja u stanje. Potom se, kao i obično, formule rešavaju SAT rešavačem.

Ograničenu proveru modela ćemo ilustrovati na primeru dvobitnog brojača prikazanog na slici 8.7. Ulaz *pulse* daje signal brojaču (tj. njegovim flip-flopovima) kad treba da izračuna nove vrednosti izlaza p_0 i p_1. Te izlazne vrednosti biće kasnije upotrebljene kao nove ulazne vrednosti flip-flopova. Neka su na izlazima p_0 i p_1, na primer, vrednosti 0 i 0. Kada stigne signal *pulse*, flip-flopovi se aktiviraju, p_1 dobija vrednost 0, a p_0 dobija vrednost izlaza NOR kola za ulaze 0 i 0, tj. vrednost 1. Dakle, ako su ulazne vrednosti (p_1, p_0) jednake $(0, 0)$, izlazne su $(0, 1)$. Slično, ako su ulazne vrednosti $(0, 1)$, izlazne su $(1, 0)$. Generalno, ako je stanje u trenutku i opisano bitovima p_0^i i p_1^i, iz dizajna brojača sledi da za svaka dva uzastopna stanja važi: $p_0^{i+1} \equiv \neg p_0^i \wedge \neg p_1^i$ i $p_1^{i+1} \equiv p_0^i$. Ponašanje brojača sažeto je dato tabelom prelaska prikazanoj na slici 8.3. Kao što se vidi iz tabele, ako se kreće od stanja $(0, 0)$, ide se u stanje $(0, 1)$, pa u stanje $(1, 0)$, pa u stanje $(0, 0)$, i to se ciklično ponavlja. Ako se stanja razmatraju kao brojevi u binarnom zapisu, od stanja 0 ide se u stanje 1, pa u stanje 2, pa u stanje 0 i to se ciklično ponavlja, te se zato opisano kolo naziva *brojač*.

Cilj je dokazati da ako brojač započne brojanje od bilo kog stanja (tj. broja) koje nije 3, onda nikada neće doći do broja 3. Željeno svojstvo opisuje se formulom $\neg p_0^i \vee \neg p_1^i$. Uslov koji važi u polaznom stanju je $\neg p_0^0 \vee \neg p_1^0$. Da bi se pokazalo da stanje k ne odgovara broju 3, treba ustanoviti da je sledeća formula nezadovoljiva:



Slika 8.7: Dizajn brojača koji broji ciklično od 0 do 2.

p_1^i	p_0^i	p_1^{i+1}	p_0^{i+1}
0	0	0	1
0	1	1	0
1	0	0	0
1	1	1	0

Tabela 8.3: Tablica prelaska brojača koji broji od 0 do 2.

$$(\neg p_0^0 \vee \neg p_1^0) \wedge (p_0^1 \Leftrightarrow \neg p_0^0 \wedge \neg p_1^0) \wedge (p_1^1 \Leftrightarrow p_0^0) \wedge \ldots$$
$$\wedge (p_0^k \Leftrightarrow \neg p_0^{k-1} \wedge \neg p_1^{k-1}) \wedge (p_1^k \Leftrightarrow p_0^{k-1}) \wedge (p_0^k \wedge p_1^k)$$

Prvi konjunkt u gornjoj konjunkciji predstavlja polazni uslov, potom slede po dva konjunkta koji opisuju prelaske između susednih stanja, a poslednji konjunkt izražava negaciju željenog svojstva u poslednjem stanju. Ukoliko je formula zadovoljiva za neko k, to znači da postoji put od polaznog stanja kojim se može doći do stanja koje ne zadovoljava traženo svojstvo. U tom slučaju, dobijena valuacija određivala bi stanja preko kojih se može doći do problematičnog stanja, što može pomoći u pronalaženju greške u analiziranom sistemu. U našem primeru, navedena formula nezadovoljiva je za svako k (ali to opšte svojstvo ne može se utvrditi SAT rešavačem).

8.4.8 Link gramatike

Link gramatike (eng. *link grammars*) su sintaksički model prirodnog jezika u kojem se parovi reči povezuju *vezama*. U rečniku konkretnog jezika, svakoj reči jezika pridružena je lista mogućih *konektora* različitih tipova, od kojih neki povezuju ulevo a neki udesno. Konektori koji povezuju udesno označavaju se znakom „+“, a konektori koji povezuju ulevo označavaju se znakom „−“. Dve reči u rečenici mogu biti povezane vezom ako postoji konektor koji levu reč povezuje udesno i konektor istog tipa koji desnu reč povezuje ulevo. Na primer, reč „big“ može da ima konektor `A+`, a reč „cat“ konektor `A-`, što omogućava kreiranje fraze „big cat“. Povezivanje se odnosi na konkretna pojavljivanja neke reči, pa ukoliko neka reč ima dva pojavljivanja, ta dva pojavljivanja razmatraju se potpuno nezavisno.

Reči mogu da budu povezane sa više reči i ulevo i udesno. Pojedinačnim rečima pridružena su i dodatna pravila koja konjunkcijama i disjunkcijama povezuju moguće konektore. Ovakvi opisi mogu da sadrže proizvoljno ugnježđene disjunkcije i konjunkcije. Na primer, reč „cat“ može da bude opisana na sledeći način:

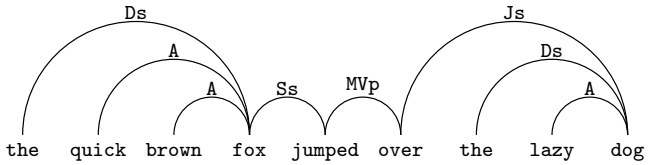
```
cat (Ds- and (Ss+ or SIs-)) or (A- and Ds- and (Ss+ or SIs-))
```

Ovaj opis znači da reč „cat“ mora imati konektor `Ds` ulevo i, istovremeno, konektor `Ss` udesno ili `SIs` ulevo ili, alternativno, konektore `A` i `Ds` ulevo i, istovremeno, konektor `Ss` udesno ili `SIs` ulevo.

Niz reči čini ispravnu rečenicu jezika definisanog gramatikom ako postoji način da se povežu njegove reči tako da važe sledeći uslovi:

- povezanost: vezama su povezane sve reči u rečenici;
- zadovoljenje: veze zadovoljavaju sve postojeće uslove za sve reči u rečenici;
- planarnost: veze se ne seku (kada su iscrtane iznad reči).

U nastavku je dato ispravno povezivanje za rečenicu na engleskom jeziku „the quick brown fox jumped over the lazy dog“:



Svaki konektor ima specifično značenje (a za konkretan jezik može postojati oko stotinu vrsta konektora). Za engleski jezik, na primer, konektor `Ds` povezuje član (*the* ili *a*) sa imenicom u jednini, konektor `A` povezuje atribut sa imenicom, konektor `Ss` povezuje imenicu-subjekat u jednini sa glagolom, konektor `O` povezuje glagole sa svojim imenicom-objektom. Imena konektora su mnemonička i asociraju na svoje značenje. Na primer, ime `Ds` je skovano od engleskih reči „determiner“ (što znači određivač) i „singular“ (što znači jednina).

Parser zasnovan na link gramatikama je algoritam koji zadatoj rečenici pridružuje sintaksičku strukturu sačinjenu od skupa veza između parova reči. Poznavanje sintaksičke strukture rečenice omogućava njeno „razumevanje“, dalju automatsku obradu, automatsko prevođenje na drugi jezik i slično.

Prilikom parsiranja rečenice potrebno je ispitati veliki broj mogućih povezivanja reči. Naime, za svaku reč u rečenici potrebno je razmatrati sve moguće konektore zadate opisom te reči u rečniku. Onda je potrebno ispitati da li je tako izabrane konektore moguće povezati na ispravan način. Pošto se svi uslovi mogu svesti na uslove da ima ili nema nekog konektora i ima ili nema neke veze, očigledno je da se ovakvo parsiranje prirodno može svesti na problem SAT. I, zaista, parseri za link gramatike često koriste taj pristup.

Ilustrujmo, pojednostavljeno, na primeru rečenice na engleskom jeziku „the big cat chased John“ parsiranje korišćenjem link gramatika. Pretpostavimo da je raspoloživ sledeći sadržaj rečnika:

```
the     Ds+ or Dp+
big     A+
cat     (Ds- and (Ss+ or SIs-)) or
             (A- and (Ds- and (Ss+ or SIs-)))
chased ((Ss- or T-) and O+) or V-
John    O- or (A- and O-)
```

Modelujmo iskaznim promenljivim i iskaznim formulama finalno ispravno povezivanje reči date rečenice. Neka iskazna promenljiva `K[the, Ds+]` označava da će u finalnom povezivanju reč „the“ (prva reč u rečenici, uvek se misli na jednu konkretnu) biti povezana udesno konektorom `Ds`, neka `K[the, Dp+]` označava da će u finalnom povezivanju reč „the“ biti povezana udesno konektorom `Dp`, neka `K[cat, (Ds- and (Ss+ or SIs-))]` označava da će u finalnom povezivanju reč „cat“ biti povezana tako da zadovoljava uslov `(Ds- and (Ss+ or SIs-))` i tako dalje. Zbog uslova povezanosti i uslova zadovoljenja moraju biti zadovoljeni uslovi:

$$\texttt{K[the, Ds+]} \vee \texttt{K[the, Dp+]}$$
$$\texttt{K[big, A+]}$$
$$\texttt{K[cat, Ds- and (Ss+ or SIs-)]} \vee \texttt{K[cat, (A- and (Ds- and (Ss+ or SIs-))]}$$
$$\texttt{K[chased, (Ss- or T-) and O+]} \vee \texttt{K[chased, V-]}$$
$$\texttt{K[John, O- or (A- and O-)]}$$

Zbog uslova zadovoljenja, za složene uslove potrebno je dodati i formule koje opisuju logičku strukturu pravila, na primer:

$$\texttt{K[cat, Ds- and (Ss+ or SIs-)]} \Leftrightarrow \texttt{K[cat, Ds-]} \wedge (\texttt{K[cat, Ss+]} \vee \texttt{K[cat, SIs-]})$$

Neka iskazna promenljiva `V[the-Ds-cat]` označava da će u finalnom povezivanju reči „the“ i „cat“ biti povezane konektorom `Ds`. Za svaki par reči koji ima isti konektor usmeren u dva smera, zbog uslova ispravnog povezivanja potrebno je dodati formule poput:

$$\texttt{V[the-Ds-cat]} \Rightarrow \texttt{K[the, Ds+]} \wedge \texttt{K[cat, Ds-]}$$

Ukoliko bi u rečenici postojala još neka reč sa konektorom `Ds-` (na primer, „dog“), moralo bi da se obezbedi da se (jedna konkretna) reč „the“ povezuje samo sa jednom:

$$\neg\,(\texttt{V[the-Ds-dog]} \wedge \texttt{V[the-Ds-cat]})$$

Potrebno je obezbediti i uslov planarnosti: treba zabraniti parove veza koje se seku. Na primer, ne mogu da budu povezane reči „the“ i „cat“ i istovremeno reči „big“ i „John“:

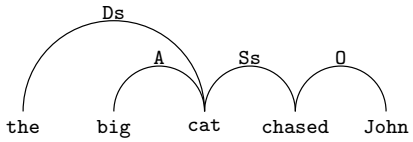
$$\neg\,(\texttt{V[the-Ds-cat]} \wedge \texttt{V[big-A-John]}).$$

Konjunkcija svih ovih uslova daje formulu čiju zadovoljivost treba ispitati. Ako je formula zadovoljiva, onda (svaki) njen model daje jedno ispravno povezivanje, tj. parsiranje zadate rečenice. U navedenom primeru, model

je valuacija v u kojoj važi:

$v(\texttt{V[the-Ds-cat]}) = 1$
$v(\texttt{V[big-A-cat]}) = 1$
$v(\texttt{V[cat-Ss-chased]}) = 1$
$v(\texttt{V[chased-O-John]}) = 1$

što daje ispravno povezivanje ilustrovano sledećom slikom:



Izražajna snaga link gramatika jednaka je izražajnoj snazi konteksno slobodnih gramatika, ali je opisivanje gramatike prirodnog jezika i parsiranje često znatno lakše za link gramatike.

Rečnici potrebni za parsiranje razlikuju se, naravno, od jezika do jezika. Njihovo kreiranje, posebno za prirodne jezike, zahtevan je zadatak.

8.4.9 SAT rešavači i dimacs-cnf format

Kao što je već rečeno, SAT *rešavači* (eng. SAT *solvers*) su programi koji rešavaju instance SAT problema. Većina savremenih SAT rešavača zasnovana je na DPLL proceduri, ali je obogaćena mnogim tehnikama i heuristikama. Neki od popularnih SAT rešavača su MiniSAT, PicoSAT i zChaff.

SAT rešavači obično očekuju ulaz u DIMACS-CNF formatu. U ovom formatu, prvi red sadrži informaciju o broju iskaznih promenljivih i broju klauza, a naredni redovi sadrže zapis po jedne klauze. Promenljive su označene rednim brojevima, negirane promenljive odgovarajućim negativnim brojevima i svaki red završava se brojem 0. Na primer, sadržaj

```
p cnf 3 2
1 -3 0
-1 2 3 0
```

odgovara formuli (sa tri promenljive i dve klauze): $(p_1 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$.

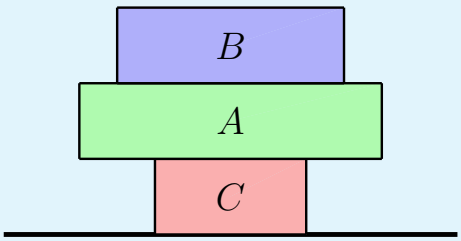
Glava 9

Automatsko rasuđivanje u logici prvog reda

Logika prvog reda, predikatska logika (eng. *first order logic, predicate logic*), znatno je izražajnija od iskazne logike. Ključna novina u odnosu na iskaznu logiku je uvođenje kvantifikovanja, univerzalnog i egzistencijalnog. Zahvaljujući kvantifikatorima, u logici prvog reda mogu se formulisati tvrđenja koja nije moguće formulisati na jeziku iskazne logike. U logici prvog reda dozvoljeno je samo kvantifikovanje promenljivih.[1] U okviru logike prvog reda mogu se opisati mnoge matematičke teorije i mnogi praktični problemi. Za neke probleme (nad konačnim domenima) pogodnije je rešavanje korišćenjem iskazne logike, ali za neke je opisivanje i rešavanje znatno lakše korišćenjem predikatske logike.

Kao i u iskaznoj logici, centralni problemi u predikatskoj logici su ispitivanje da li je data formula valjana i da li je data formula zadovoljiva. Za razliku od iskazne logike, ovi problemi nisu odlučivi, te ne postoje efektivni algoritmi za njihovo rešavanje. No, problem ispitivanja valjanosti i problem ispitivanja nezadovoljivosti za predikatsku logiku su poluodlučivi, pa postoje metode koje za svaku valjanu formulu mogu da dokažu da je ona valjana (ali ne mogu za bilo koju formulu koja nije valjana da utvrde da nije valjana) i metode koje za svaku nezadovoljivu formulu mogu da dokažu da je ona nezadovoljiva (ali ne mogu za bilo koju zadovoljivu formulu da utvrde da je ona zadovoljiva).

Primer 9.1. *Razmotrimo jednu jednostavnu varijantu problema slaganja kutija: kutije (označene slovima) poređane su jedna na drugu. Za neke se zna da li su ispod ili iznad neke druge kutije, ali nije zadata potpuna informacija o poretku svih kutija. U ovom jednostavnom primeru dovoljno je baratati informacijama o tome koja je kutija* iznad *neke kutije. Ipak, koriste se i informacije o odnosu* ispod*, čime se ilustruju pogodnosti upotrebe jezika koji nije minimalan. U mnogim realnim primenama, od minimalnog jezika znatno je pogodniji neki izražajniji jezik.*



Pretpostavimo da su nekako naslagane tri kutije A, B i C i da je poznato da je B iznad A, a da je C ispod A. Pitanje je da li je moguće zaključiti da je B iznad C ili je moguće zaključiti da je B ispod C (ili nije moguće zaključiti ni jedno ni drugo). Opisani problem možemo opisati u terminima iskazne logike: iskazna promenljiva a_{AB} (od engleskog above*) može da označava da je A iznad B, a_{AC} da je A iznad C, a_{BA} da je B iznad A, b_{BC} (od engleskog* below*) da je B ispod C, itd. Potrebno je za svake dve kutije obezbediti da važi da ako je prva iznad druge, onda druga nije iznad prve, na primer, za kutije A i B važi: $a_{AB} \Rightarrow \neg a_{BA}$. Potrebno je za svake dve kutije obezbediti da važi da ako je prva iznad druge, onda je druga ispod prve i obratno, na primer, za kutije A i B važi: $a_{AB} \Leftrightarrow b_{BA}$. Potrebno je za svake tri kutije obezbediti da važi: ako je prva iznad druge i druga iznad treće, onda je prva iznad treće, na primer, za*

[1] U logici višeg reda predikati i funkcije kao argumente mogu imati druge predikate i funkcije i dozvoljeno je njihovo kvantifikovanje. Na primer, u logici drugog reda predikati i funkcije mogu za argumente imati predikate i funkcije prvog reda i mogu biti kvantifikovani. Predikati i funkcije reda n mogu za argumente imati predikate i funkcije reda $n-1$ i mogu biti kvantifikovani.

kutije A, B i C važi: $a_{AB} \wedge a_{BC} \Rightarrow a_{AC}$. Ako postoje tri kutije, onda ovakvih uslova ima $3! = 6$, a ako ih ima n, onda tih uslova ima $6\binom{n}{3}$. Dakle, iako jeste moguće, kodiranje u terminima iskazne logike može da bude rogobatno i prostorno veoma zahtevno. Bilo bi dobro ako bismo umesto $6\binom{n}{3}$ uslova mogli da koristimo samo jedan: „za svake tri kutije X, Y, Z važi: ako je a_{XY} i a_{YZ} onda je a_{XZ}.“ Logika prvog reda daje takvu mogućnost i zadati problem mogao bi da se elegantno opiše sledećim uslovima, pri čemu se ne koriste iskazne promenljive poput a_{AB} nego atomičke formule sa argumentima poput $above(A, B)$ i $below(A, B)$, pri čemu $above(A, B)$ označava da je A iznad B, a $below(A, B)$ označava da je A ispod B:

- *„za svake dve kutije x, y važi: ako je $above(x, y)$ onda nije $above(y, x)$“;*
- *„za svake dve kutije x, y važi: $above(x, y)$ ako i samo ako $below(y, x)$“;*
- *„za svake tri kutije x, y, z važi: ako je $above(x, y)$ i $above(y, z)$ onda je $above(x, z)$“.*

Za sve navedene, a i druge slične formule, potrebno je definisati način na koji im se pridružuje vrednost tačno *ili* netačno.

Za rešavanje početnog problema, potrebno je iz navedenih uslova i iz pretpostavki da važi $above(B, A)$ i $below(C, A)$ izvesti zaključak $above(B, C)$ ili $below(B, C)$.

Primer 9.2. *U primeru 7.2, potrebno je dokazati (jedan način grubo je opisan u primeru 7.12) da iz uslova:*

- *„za proizvoljne tri osobe x, y, z važi: ako je x predak od y i y je predak od z, onda je x predak od z“*
- *„za proizvoljnu osobu x važi: otac osobe x je predak osobe x“*

sledi

- *„otac oca osobe Ana je predak osobe Ana.“*

9.1 Sintaksa i semantika logike prvog reda

Sintaksa logike prvog reda govori o njenom *jeziku* — o skupu njenih (ispravno formiranih) formula i ne razmatra njihovo značenje i njihove (moguće) istinitosne vrednosti. Značenje formula logike prvog reda uvodi semantika logike prvog reda.

9.1.1 Sintaksa logike prvog reda

Skup formula logike prvog reda obično se definiše za fiksiran, prebrojiv skup promenljivih V, dve logičke konstante — $\top$ (*te*) i $\bot$ (*nete*), konačan skup osnovnih logičkih veznika: unarnog — *negacija* i binarnih — *konjunkcija*, *disjunkcija*, *implikacija*, *ekvivalencija*, kao i dva kvantifikatora (kvantora) — *univerzalni* i *egzistencijalni*. Nabrojane komponente čine *logički (ili opšti) deo* jezika prvog reda. Pored njih, u izgradnji formula logike prvog reda koriste se elementi *signature*.

Definicija 9.1 (Signatura)**.** Signaturu $\mathcal{L}$ *čine:*

- *(najviše prebrojiv) skup funkcijskih simbola (sa svojim arnostima);*
- *(najviše prebrojiv) skup predikatskih (relacijskih) simbola (sa svojim arnostima).*

Funkcijske simbole arnosti 0 *zovemo* simbolima konstanti.

Funkcijski simbol f koji ima arnost, na primer, 2 zapisujemo nekad i kao $f_{/2}$. Takođe, predikatski simbol p koji ima arnost 3 zapisujemo nekad i kao $p_{/3}$.

Pojedinačni jezici prvog reda razlikuju se po signaturi. Za svaki zaseban problem (ili matematičku teoriju) bira se specifična, pogodna signatura. Na primer, za rešavanje problema kutija (iz primera 9.1), koristiće se predikatski simboli koji odgovaraju položajima ispod i iznad, u izgradnji geometrije koristiće se predikatski simbol za paralelnost, a u izgradnji aritmetike – funkcijski simbol $+$.

Naredna definicija uvodi pojmove *termova* i *formula logike prvog reda* (ili kraće – skup *formula*) nad nekom (fiksiranom) signaturom $\mathcal{L}$. Termove, atomičke formule i formule nad signaturom $\mathcal{L}$ zvaćemo, kraće, samo *termovi*

(eng. *terms*), *atomičke formule* (eng. *atomic formulae*) i *formule* (eng. *formulae*), ako je signatura jasno određena kontekstom ili ako nije relevantna. Intuitivno, termovi odgovaraju elementima nekog domena (na primer, zbiru može da odgovara broj, majka neke osobe je neka osoba), a formulama odgovaraju tvrđenja koja mogu biti tačna ili netačna (na primer, tvrđenje da je neka osoba roditelj neke druge).

Definicija 9.2 (Skup termova i formula logike prvog reda nad signaturom $\mathcal{L}$). *Za signaturu $\mathcal{L}$:*

- *promenljive i funkcijski simboli arnosti 0 su* termovi*; term je i objekat dobijen primenom funkcijskog simbola f arnosti n na termove $t_1, \ldots, t_n$; termovi mogu biti dobijeni samo na opisane načine;*
- atomička formula *je logička konstanta ili objekat dobijen primenom predikatskog simbola p arnosti n na termove $t_1, \ldots, t_n$;*
- *atomičke formule su formule;*
- *ako su $\mathcal{A}$ i $\mathcal{B}$ formule, onda su formule i objekti dobijeni kombinovanjem ovih formula logičkim veznicima i kvantifikatorima (sa promenljivim);*
- *formule mogu biti dobijene samo na načine opisane u prethodne dve stavke.*

Literal (eng. *literal*) je atomička formula ili negacija atomičke formule. *Klauza* (eng. *clause*) je disjunkcija više literala.

Analogno iskaznom slučaju, uobičajeno je da se formule zapisuju kao nizovi simbola i da se negacija zapisuje kao $\neg$, konjunkcija kao $\wedge$, disjunkcija kao $\vee$, implikacija kao $\Rightarrow$, ekvivalencija kao $\Leftrightarrow$, univerzalni kvantifikator kao $\forall$, egzistencijalni kao $\exists$. Primena funkcijskog simbola f na termove $t_1, \ldots, t_n$ zapisuje se prefiksno kao $f(t_1, \ldots, t_n)$ (i analogno za predikatske simbole). Binarni funkcijski i predikatski simboli zapisiju se ponekad i u infiksnom obliku. Ako su $\mathcal{A}$ i $\mathcal{B}$ formule i x element skupa V, onda su formule i $(\neg\mathcal{A})$, $(\mathcal{A} \wedge \mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \Rightarrow \mathcal{B})$, $(\mathcal{A} \Leftrightarrow \mathcal{B})$, $(\forall x)\mathcal{A}$, $(\exists x)\mathcal{A}$ i slično. Na primer, zapis $(\forall x)\mathcal{A}$ čitamo „za svako x $\mathcal{A}$“, a zapis $(\exists x)\mathcal{A}$ čitamo „postoji x takvo da je $\mathcal{A}$“ (naglasimo da su formule $(\forall x)\mathcal{A}$ i $(\exists x)\mathcal{A}$ ispravne i ako $\mathcal{A}$ ne sadrži promenljivu x). U opisanom zapisu neophodno je koristiti zagrade kako bi se izbegla višesmislenost. Da bi se izbeglo korišćenje velikog broja zagrada obično se izostavljaju spoljne zagrade a i podrazumeva se prioritet veznika kao u iskaznoj logici, uz dodatak da kvantifikatori imaju viši prioritet od svih veznika (pa je, na primer, $(\forall x)\mathcal{A} \wedge \mathcal{B}$ kraći zapis za $((\forall x)\mathcal{A}) \wedge \mathcal{B}$, a ne za $(\forall x)(\mathcal{A} \wedge \mathcal{B})$).

Uz indeks ili bez indeksa, simbole konstanti obično (mada ne isključivo) označavamo simbolima $a, b, c, \ldots$, funkcijske simbole arnosti veće od 0 simbolima $f, g, h, \ldots$, predikatske simbole simbolima $p, q, r, \ldots$, promenljive simbolima $x, y, z, \ldots$, formule simbolima $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$, skupove formula simbolima $\Gamma, \Delta, \ldots$

Ako su dve formule $\mathcal{A}$ i $\mathcal{B}$ identične, onda to označavamo $\mathcal{A} = \mathcal{B}$ a inače pišemo $\mathcal{A} \neq \mathcal{B}$.

Primer 9.3. *Signatura za problem iz primera 9.1 je $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$. Za ovu signaturu i skup promenljivih $V = \{x, y, z, \ldots\}$, neke od atomičkih formula su $above(x, z)$ i $below(x, y)$, a primer formule (koja nije atomička) je $\forall x \neg above(x, x)$.*

Primer 9.4. *Signatura za problem iz primera 9.2 je $\mathcal{L} = (\{Ana_{/0}, otac_{/1}\}, \{predak_{/2}\})$. Smatramo da je Ana konkretna, jedinstvena osoba, pa je pogodno i prirodno da za tu osobu koristimo simbol konstante. Primetimo da je u izabranoj signaturi otac funkcijski simbol, a predak predikatski simbol. Takvo modelovanje je prirodno jer je (biološki) otac za svaku osobu jedinstven, a predak nije. Za ovu signaturu i skup promenljivih $V = \{x, y, z, \ldots\}$, neki od termova su Ana i $otac(x)$, primer atomičke formule je $predak(otac(x), Ana)$, a primer formule koja nije atomička je $\forall x \neg predak(x, x)$.*

Primer 9.5. *Neki od termova nad signaturom $\mathcal{L} = (\{sokrat_{/0}\}, \{man_{/1}, mortal_{/1}\})$ i skupom promenljivih $V = \{x, y, z, \ldots\}$ su: x, y, sokrat. Neke od atomičkih formula su $man(x)$, $mortal(y)$, $mortal(sokrat)$, a neke od formula su $\forall x\, man(x)$ i $\forall x\, (man(x) \Rightarrow mortal(x))$.*

Primer 9.6. *Za signaturu $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$, i skup promenljivih $V = \{x, y, z, \ldots\}$, neki od termova su e, $\star(x, y)$ i $\star(x, \star(y, z))$, neke od atomičkih formula su $\prec (x, e)$ i $\prec (\star(x, y), z)$, a jedna od (neatomičkih) formula je $\forall x \neg(\prec (x, x))$. Ukoliko se binarni funkcijski i predikatski simboli zapišu infiksno (umesto prefiksno), onda se navedeni termovi i formule zapisuju na sledeći način: e, $x \star y$, $x \star (y \star z)$, $x \prec e$, $(x \star y) \prec z$,*

$\forall x\, \neg(x \prec x)$.

Primer 9.7. *U neformalnom i poluformalnom matematičkom izražavanju, mogu se sresti konstrukcije poput* $\forall x < \delta \ldots$ *Iako, po simbolima koje uključuje, podseća na formulu logike prvog reda, ovo očigledno nije dobro zasnovana formula. Na primer, neprekidnost i ravnomerna neprekidnost (totalne) realne funkcije* f *često se definišu na sledeći način:*

$$(\forall a)(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon)$$

$$(\forall \varepsilon > 0)(\exists \delta > 0)(\forall x_1, x_2)(|x_1 - x_2| < \delta \Rightarrow |f(x_1) - f(x_2)| < \varepsilon)$$

Odgovarajuće (precizno zapisane) dobro zasnovane formule su:

$$(\forall a)(\forall \varepsilon)(\varepsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x)(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon)))$$

$$(\forall \varepsilon)(\varepsilon > 0 \Rightarrow (\exists \delta)(\delta > 0 \wedge (\forall x_1)(\forall x_2)(|x_1 - x_2| < \delta \Rightarrow |f(x_1) - f(x_2)| < \varepsilon)))$$

Definicija 9.3 (Slobodno i vezano pojavljivanje promenljive)**.** *U formulama* $\forall x\mathcal{A}$ *i* $\exists x\mathcal{A}$*, formula* $\mathcal{A}$ *je* doseg *kvantifikatora.*

Pojavljivanje promenljive x *je* vezano *u* $\forall x$ *i* $\exists x$*, kao i ako je u dosegu kvantifikatora* $\forall x$ *ili* $\exists x$*, a inače je* slobodno*. Promenljiva je* vezana *(*slobodna*) u formuli ako i samo ako ima vezano (slobodno) pojavljivanje u toj formuli.*

Primetimo da promenljiva može biti i slobodna i vezana u jednoj formuli.

Primer 9.8. *U formuli* $p(x, y)$*, pojavljivanje promenljive* x *je slobodno i ona je slobodna u ovoj formuli.*

U formuli $p(x, y) \Rightarrow (\forall x)q(x)$ *prvo pojavljivanje promenljive* x *je slobodno, a drugo i treće pojavljivanje je vezano. U ovoj formuli, promenljiva* x *je i slobodna i vezana.*

U formuli $(\forall x)p(x, y) \Rightarrow (\forall x)q(x)$*, sva pojavljivanja promenljive* x *su vezana i promenljiva je vezana u ovoj formuli.*

U sva tri primera, pojavljivanja promenljive y *su slobodna.*

Često se zapisom $\mathcal{A}(x_1, x_2, \ldots, x_n)$ naglašava da formula $\mathcal{A}$ ima slobodne promenljive $x_1, x_2, \ldots, x_n$.

Priroda vezanih promenljivih u formuli bitno je drugačija od prirode slobodnih promenljivih. Naime, značenje formule $(\forall x)\mathcal{A}(x)$ (koje će tek u nastavku biti definisano) ne zavisi od x i isto je kao i značenje formule $(\forall y)\mathcal{A}(y)$ (slično kao što vrednost $\int_0^\pi \sin(x)dx$ ne zavisi od x, iako x figuriše u ovom izrazu). Dakle, sva pojavljivanja vezanih promenljivih mogu se preimenovati, bez uticaja na značenje formule. Na primer, formula $p(x, y) \Rightarrow (\forall x)q(x)$ iz primera 9.8 ima isto značenje (koje će biti precizno definisano u nastavku), kao i formula $p(x, y) \Rightarrow (\forall z)q(z)$. S druge strane, značenje formula zavisi od slobodnih promenljivih u njima.

Formula bez slobodnih promenljivih zove se *zatvorena formula* ili *rečenica*. Ako formula $\mathcal{A}$ ima kao slobodne samo promenljive $x_1, x_2, \ldots, x_k$ onda formulu $(\forall x_1)(\forall x_2)\ldots(\forall x_k)\mathcal{A}$ nazivamo *univerzalnim zatvorenjem* formule $\mathcal{A}$, a formulu $(\exists x_1)(\exists x_2)\ldots(\exists x_k)\mathcal{A}$ *egzistencijalnim zatvorenjem* formule $\mathcal{A}$.

9.1.2 Zamena

U logici prvog reda, razmatramo *zamenu (supstituciju) promenljive termom* i *zamenu (supstituciju) formule formulom*. Zamene promenljive termom biće korišćene u kontekstu unifikacije (poglavlje 9.3.3), a zamene formule formulom u kontekstu transformisanja formula zarad jednostavnijeg ispitivanja valjanosti ili zadovoljivosti. U daljem tekstu ćemo pod terminom *izraz* podrazumevati i termove i formule. Svaka zamena određuje rekurzivno definisanu funkciju kojom se izrazi preslikavaju u izraze. Zamenu izraza e_1 izrazom e_2 zapisujemo $[e_1 \mapsto e_2]$. Zamene obično kraće označavamo malim grčkim slovima, na primer σ. Izraz koji je rezultat primene zamene $\sigma = [e_1 \mapsto e_2]$ na izraz e označavamo sa $e[e_1 \mapsto e_2]$ ili $e\sigma$.

Zamena promenljive termom definiše se u logici prvog reda u istom duhu kao u iskaznoj logici: ako termove i formule reprezentujemo u računarskom programu u vidu stabla, onda zamenu promenljive termom možemo opisati rekurzivnom funkcijom koja sve listove formule koji odgovaraju zadatoj promenljivoj zamenjuje nekim drugim zadatim stablom.

Definicija 9.4 (Zamena promenljive termom (u termu)). *Za promenljivu x, term t koji ne sadrži x i za zamenu $\sigma = [x \mapsto t]$, term dobijen primenom zamene σ na term s definišemo na sledeći način:*

- *ako je $s = x$, onda je $s\sigma = t$;*
- *ako je s simbol konstante ili je $s = y$, gde je $y \neq x$, onda je $s\sigma = s$;*
- *ako je $s = f(t_1, t_2, \ldots, t_n)$, onda je $s\sigma = f(t_1\sigma, t_2\sigma, \ldots, t_n\sigma)$.*

Definicija 9.5 (Zamena promenljive termom (u formuli)). *Za promenljivu x, term t koji ne sadrži x i za zamenu $\sigma = [x \mapsto t]$, formulu dobijenu primenom zamene σ na formulu $\mathcal{A}$ definišemo na sledeći način:*

- *ako je formula $\mathcal{A}$ jednaka $p(t_1, t_2, \ldots, t_n)$, onda je formula $\mathcal{A}\sigma$ jednaka $p(t_1\sigma, t_2\sigma, \ldots, t_n\sigma)$;*
- *ako je formula $\mathcal{A}$ jednaka $\top$ ili $\bot$ ili je oblika $\forall x\mathcal{B}$ ili $\exists x\mathcal{B}$, onda je formula $\mathcal{A}\sigma$ jednaka $\mathcal{A}$;*
- *ako je formula $\mathcal{A}$ oblika $\neg\mathcal{B}$, onda je formula $A\sigma$ jednaka $\neg(\mathcal{B}\sigma)$;*
- *ako je formula $\mathcal{A}$ oblika $\mathcal{B}_1 \wedge \mathcal{B}_2$, $\mathcal{B}_1 \vee \mathcal{B}_2$, $\mathcal{B}_1 \Rightarrow \mathcal{B}_2$ ili $\mathcal{B}_1 \Leftrightarrow \mathcal{B}_2$, onda je formula $A\sigma$ jednaka (redom) $\mathcal{B}_1\sigma \wedge \mathcal{B}_2\sigma$, $\mathcal{B}_1\sigma \vee \mathcal{B}_2\sigma$, $\mathcal{B}_1\sigma \Rightarrow \mathcal{B}_2\sigma$ ili $\mathcal{B}_1\sigma \Leftrightarrow \mathcal{B}_2\sigma$;*
- *ako je formula $\mathcal{A}$ oblika $(\forall y)\mathcal{B}$ ili $(\exists y)\mathcal{B}$, gde je $x \neq y$ i y se ne pojavljuje u t, tada je $\mathcal{A}\sigma$ jednako (redom) $(\forall y)(\mathcal{B}\sigma)$ ili $(\exists y)(\mathcal{B}\sigma)$.*

Ograničenje u poslednjem pravilu u prethodnoj definiciji da se y ne pojavljuje u t obezbeđuje da nijedna promenljiva iz t ne sme da postane vezana u rezultujućoj formuli (jer bi to moglo bitno da menja značenje polazne formule). Inače bi, na primer, $((\exists y)predak(x, y))[x \mapsto y]$ bilo jednako $(\exists y)predak(y, y)$ a ta formula ima drugačije značenje od polazne formule (značenje formula će biti tek definisano u nastavku). Ukoliko se y pojavljuje u t, onda je potrebno najpre preimenovati promenljivu y u z u formuli $\mathcal{A}$ (pri čemu je z neka promenljiva koja se ne pojavljuje ni u $\mathcal{A}$ ni u t), pa tek na rezultujuću formulu primeniti zamenu $[x \mapsto t]$. Tako je rezultat zamene $((\exists y)predak(x, y))[x \mapsto y]$ jednak $(\exists z)predak(y, z)$.

Definicija 9.6 (Uopštena zamena). Uopštena zamena $\sigma = [x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n]$ *je niz uzastopnih zamena $[x_1 \mapsto t_1]$, $[x_2 \mapsto t_2]$, $\ldots$, $[x_n \mapsto t_n]$ gde su x_i promenljive a t_i su termovi, pri čemu važi $x_i \neq x_j$ za $i \neq j$, kao i da t_i ne sadrži x_j, za bilo koje i i j.*

U nastavku ćemo umesto termina *uopštena zamena* govoriti samo *zamena* ili *supstitucija*.

Definicija 9.7 (Kompozicija zamena). *Za zamene $\phi = [x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n]$ i $\lambda = [y_1 \mapsto s_1, y_2 \mapsto s_2, \ldots, y_m \mapsto s_m]$, pri čemu nijedan od termova y_i, s_i ne sadrži nijednu od promenljivih x_j,* kompozicija zamena $\phi\lambda$ *je zamena $[x_1 \mapsto t_1\lambda, x_2 \mapsto t_2\lambda, \ldots, x_n \mapsto t_n\lambda, y_1 \mapsto s_1, y_2 \mapsto s_2, \ldots, y_m \mapsto s_m]$.*

Primer 9.9. *Za $\sigma = [x \mapsto f(y)]$ i $s = g(a, x)$ važi $s\sigma = g(a, f(y))$.*
Za $\sigma = [x \mapsto f(z), y \mapsto a]$, $s = g(a, x)$ i $t = g(y, g(x, y))$ važi $s\sigma = g(a, f(z))$ i $t\sigma = g(a, g(f(z), a))$.
Za $\sigma = [x \mapsto sokrat, y \mapsto platon]$ i $s = učitelj(x, y)$ važi $s\sigma = učitelj(sokrat, platon)$.
Za $\sigma = [x \mapsto a]$ i $s = u(x, x)$ važi $s\sigma = u(a, a)$.
Za $\phi = [x \mapsto f(y)]$ i $\lambda = [y \mapsto g(z)]$, važi $\phi\lambda = [x \mapsto f(g(z)), y \mapsto g(z)]$.

Pored zamene promenljive termom, potreban nam je i pojam zamene formule formulom, uveden narednom definicijom.

Definicija 9.8 (Zamena formule formulom (u formuli)). *Za formule $\mathcal{C}$ i $\mathcal{D}$ takve da u $\mathcal{D}$ nema slobodnih promenljivih koje nisu slobodne u $\mathcal{C}$ i za zamenu $\sigma = [\mathcal{C} \mapsto \mathcal{D}]$, formulu dobijenu primenom zamene σ na formulu $\mathcal{A}$ definišemo na sledeći način:*

- *ako važi $\mathcal{A} = \mathcal{C}$, onda je $\mathcal{A}\sigma = \mathcal{D}$;*
- *ako važi $\mathcal{A} \neq \mathcal{C}$, onda:*

- *ako je formula $\mathcal{A}$ atomička, onda je $\mathcal{A}\sigma = \mathcal{A}$;*
- *ako za neku formulu $\mathcal{B}$ važi da je formula $\mathcal{A}$ jednaka $\neg\mathcal{B}$, $\forall x\mathcal{B}$ ili $\exists x\mathcal{B}$, onda je formula $\mathcal{A}\sigma$ jednaka (redom) $\neg(\mathcal{B}\sigma)$, $(\forall x)(\mathcal{B}\sigma)$ ili $(\exists x)(\mathcal{B}\sigma)$;*
- *ako za neke formule $\mathcal{B}_1$ i $\mathcal{B}_2$ važi da je formula $\mathcal{A}$ jednaka $\mathcal{B}_1 \wedge \mathcal{B}_2$, $\mathcal{B}_1 \vee \mathcal{B}_2$, $\mathcal{B}_1 \Rightarrow \mathcal{B}_2$ ili $\mathcal{B}_1 \Leftrightarrow \mathcal{B}_2$, onda je formula $A\sigma$ jednaka (redom) $\mathcal{B}_1\sigma \wedge \mathcal{B}_2\sigma$, $\mathcal{B}_1\sigma \vee \mathcal{B}_2\sigma$, $\mathcal{B}_1\sigma \Rightarrow \mathcal{B}_2\sigma$ ili $\mathcal{B}_1\sigma \Leftrightarrow \mathcal{B}_2\sigma$.*

Navedena definicija uvodi uslov za primenu zamene oblika $\mathcal{A}[\mathcal{C} \mapsto \mathcal{D}]$: u formuli $\mathcal{D}$ nema slobodnih promenljivih koje nisu slobodne u $\mathcal{C}$. Taj uslov sprečava da se promeni očekivano značenje rezultujuće formule, na primer, tako što neka slobodna promenljiva iz $\mathcal{D}$ postaje vezana.

Primer 9.10. *Ako je formula $\mathcal{A}$ jednaka $r(y) \wedge (p(x) \vee q(x))$, a zamena σ jednaka $[p(x) \vee q(x) \mapsto r(x)]$, onda je $\mathcal{A}\sigma = r(y) \wedge r(x)$.*

Ako je formula $\mathcal{A}$ jednaka $r(y) \wedge (p(x) \vee q(x))$, a zamena σ jednaka $[p(x) \vee q(x) \mapsto q(x) \vee p(x)]$, onda je $\mathcal{A}\sigma = r(y) \wedge (q(x) \vee p(x))$.

Ako je formula $\mathcal{A}$ jednaka $r(y) \wedge (\forall x)(\neg p(x))$, a zamena σ jednaka $[(\forall x)(\neg p(x)) \mapsto \neg(\exists x)p(x)]$, onda je $\mathcal{A}\sigma = r(y) \wedge \neg(\exists x)p(x)$.

Primer 9.11. *Ako je formula $\mathcal{A}$ (nad jezikom iz primera 9.4), jednaka $predak(otac(x), y)$, a zamena σ jednaka $[y \mapsto Ana]$, onda je $\mathcal{A}\sigma = predak(otac(x), Ana)$.*

Ako je formula $\mathcal{A}$ jednaka $\forall x\, \neg predak(x, x)$, a zamena σ jednaka $[x \mapsto Ana]$, onda je $\mathcal{A}\sigma = \mathcal{A}$.

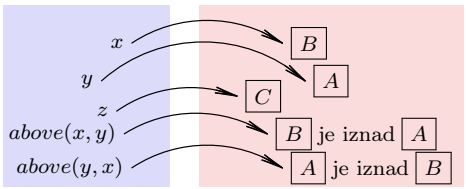
Ako formule reprezentujemo u računarskom programu u vidu stabla, onda zamenu formule formulom možemo, slično kao u iskaznom slučaju, opisati rekurzivnom funkcijom koja zamenjuje sva podstabla koja odgovaraju jednoj formuli nekim drugim zadatim stablom.

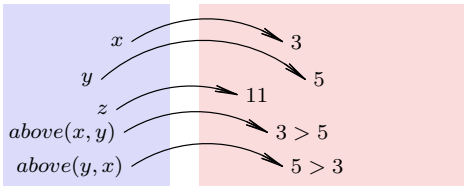
9.1.3 Semantika logike prvog reda

Semantika logike prvog reda govori o *značenju* formula. Kao i u slučaju iskazne logike, osnovna ideja semantike je da istinitosne vrednosti formula definiše u skladu sa uobičajenim, svakodnevnim rasuđivanjem. U odnosu na iskazni slučaj, stvari komplikuju kvantifikatori, kao i drugačija priroda promenljivih. Naime, promenljive se ne preslikavaju, kao u iskaznoj logici, u skup $\{0, 1\}$, nego se mogu preslikavati u elemente proizvoljnog skupa koji zovemo *univerzum* (na primer, u skup celih brojeva ili u skup stanovnika neke države).[2] Značenje formula, dodatno, nije određeno samo načinom na koji su promenljivim pridružene vrednosti, nego i time šta odgovara funkcijskim i predikatskim simbolima – to će biti neke konkretne funkcije i relacije nad izabranim domenom. Stoga istinitosna vrednost formule zavisi od više izbora i za različite izbore može da bude drugačija. Reći ćemo da je formula *valjana* ako je tačna za svaki od tih izbora.

Razmotrimo, na primer, formulu $above(x, y)$. Možemo je interpretirati na mnogo različitih načina. Jedan je da smatramo da x i y označavaju kutije naslagane na nekoj konkretnoj gomili, a da predikatskom simbolu *above* odgovara relacija „biti iznad“ za kutije na toj konkretnoj gomili. Drugi način je da smatramo da x i y označavaju cele brojeve, a da predikatskom simbolu *above* odgovara relacija $>$ nad celim brojevima. U svakom slučaju, formulama pridružujemo značenje vezano za neki konkretan skup i neke konkretne funkcije i relacije nad njim i u stanju smo da za bilo koje elemente tog skupa efektivno izračunamo vrednosti tih funkcija i efektivno odredimo da li je zadovoljena neka relacija (na primer, u stanju smo da efektivno proverimo da li važi $3 > 5$, to je jednostavan uslov nad celim brojevima). Naredna slika ilustruje dve pomenute interpretacije. Na svakoj slici levo je svet sintakse, a desno – svet semantike, levo su termovi i formule, a desno – njihovo izabrano značenje. U interpretaciji ilustrovanoj na slici (dole), promenljivim x i y pridružene su vrednosti 3 i 5, a predikatskom simbolu *above* relacija $>$, te se formula $above(x, y)$ preslikava u vrednost 0, jer $3 > 5$ je netačno nad celim brojevima.

[2] Po pridruženom značenju, promenljivim iskazne logike u logici prvog reda ne odgovaraju promenljive, već predikatski simboli arnosti nula (koji se preslikavaju u skup $\{0, 1\}$).





Uvedimo sada pojam interpretacije precizno. Interpretacija je određena $\mathcal{L}$-strukturom (univerzumom i izabranim značenjem funkcijskih i predikatskih simbola) i *valuacijom* (vrednošću promenljivih).

Definicija 9.9 ($\mathcal{L}$-*struktura*)**.** *Za datu signaturu $\mathcal{L}$, $\mathcal{L}$-*struktura $\mathfrak{D}$ *je par (D, I), gde je D skup, a I funkcija i važi:*

- *D je neprazan skup i zovemo ga* univerzum *(ponekad i* domen*);*
- *svakom simbolu konstante c iz $\mathcal{L}$ (tj. svakom funkcijskom simbolu arnosti 0), funkcija I pridružuje jedan element c_I iz D;*
- *svakom funkcijskom simbolu f iz $\mathcal{L}$ arnosti n, $n > 0$, funkcija I pridružuje jednu totalnu funkciju f_I iz D^n u D;*
- *svakom predikatskom simbolu p iz $\mathcal{L}$ arnosti n, $n > 0$, funkcija I pridružuje jednu relaciju, tj. totalnu funkciju p_I iz D^n u $\{0, 1\}$.*

Primer 9.12. *Za signaturu $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$ (iz primera 9.3), jedna moguća $\mathcal{L}$-*struktura *je $(\mathbf{K}, I)$, gde je $\mathbf{K}$ konkretan skup kutija prikazanih na slici u okviru primera 9.1. Predikatski simboli above i below preslikavaju se u relacije „jeste iznad“ i „jeste ispod“ nad tim konkretnim kutijama, u konkretnom odnosu koji imaju.*

Primer 9.13. *Za signaturu iz primera 9.4, jedna moguća $\mathcal{L}$-*struktura *je $(\mathbf{O}, I)$, gde je $\mathbf{O}$ skup svih ljudi koji su ikad živeli, I je funkcija koja simbol Ana preslikava u opersku pevačicu Anu Netrebko, funkcijski simbol otac u funkciju koja svakoj osobi pridružuje njenog oca, a predikatski simbol predak u uobičajenu relaciju „biti predak“ nad ljudima.*

*Moguća je, međutim, i $\mathcal{L}$-*struktura *$(\mathbf{N}, I)$, gde je $\mathbf{N}$ skup prirodnih brojeva, I je funkcija koja simbol Ana preslikava u broj 0 iz N, funkcijski simbol otac u funkciju sledbenika, a predikatski simbol predak u relaciju $>$ nad prirodnim brojevima.*

Primer 9.14. *Za signaturu $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$, iz primera 9.6, jedna moguća $\mathcal{L}$-*struktura *je par $(\mathbf{N}, I)$, gde je $\mathbf{N}$ skup prirodnih brojeva, a I funkcija koja simbol e preslikava u prirodni broj 1, funkcijski simbol $\star$ u operaciju množenja prirodnih brojeva, a predikatski simbol $\prec$ u relaciju $<$ nad prirodnim brojevima.*

Definicija 9.10 (Valuacija)**.** Valuacija *v za skup promenljivih V u odnosu na domen D je preslikavanje koje svakom elementu iz V dodeljuje jedan element iz D. Ako je $v(x) = d_j$, onda kažemo da je d* vrednost *promenljive x u valuaciji v.*

Ako su v i w valuacije za isti skup promenljivih i isti domen, onda sa $v \sim_x w$ označavamo da je $v(y) = w(y)$ za svaku promenljivu y različitu od x, pri čemu vrednosti $v(x)$ i $w(x)$ mogu a ne moraju biti iste.

Primer 9.15. *Za domen iz primera 9.12, valuacija v može da slika promenljive x i y u (konkretne) kutije* $\boxed{B}$ *i* $\boxed{A}$*, tj.* $v(x) = \boxed{B}$ *i* $v(y) = \boxed{A}$.

Interpretacija proširuje valuaciju i pridružuje značenje i drugim termovima i formulama.

Definicija 9.11 (Interpretacija)**.** Vrednost *(ili* značenje*) terma t u interpretaciji I_v, određenoj $\mathcal{L}$-strukturom $\mathfrak{D}$ i valuacijom v, označavamo sa $I_v(t)$ i definišemo na sledeći način:*

- *ako je term t jednak nekoj promenljivoj x, onda je $I_v(t) = v(x)$;*
- *ako je term t jednak nekom simbolu konstante c, onda je $I_v(t) = c_I$;*
- *ako je term t jednak $f(t_1, t_2, \ldots, t_n)$ onda je $I_v(t) = f_I(I_v(t_1), I_v(t_2), \ldots, I_v(t_n))$.*

Istinitosnu vrednost *(ili* značenje*) formule u interpretaciji I_v određenoj $\mathcal{L}$-strukturom $\mathfrak{D}$ i valuacijom v, definišemo na sledeći način:*

- $I_v(\top) = 1$ *i* $I_v(\bot) = 0$;
- $I_v(p(t_1, t_2, \ldots, t_n)) = p_I(I_v(t_1), I_v(t_2), \ldots, I_v(t_n))$;
- $I_v(\neg\mathcal{A}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = 0 \\ 0, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \wedge \mathcal{B}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = 1 \text{ i } I_v(\mathcal{B}) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \vee \mathcal{B}) = \begin{cases} 0, & \text{ako je } I_v(\mathcal{A}) = 0 \text{ i } I_v(\mathcal{B}) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \Rightarrow \mathcal{B}) = \begin{cases} 0, & \text{ako je } I_v(\mathcal{A}) = 1 \text{ i } I_v(\mathcal{B}) = 0 \\ 1, & \text{inače} \end{cases}$
- $I_v(\mathcal{A} \Leftrightarrow \mathcal{B}) = \begin{cases} 1, & \text{ako je } I_v(\mathcal{A}) = I_v(\mathcal{B}) \\ 0, & \text{inače} \end{cases}$
- $I_v((\forall x)\mathcal{A}) = \begin{cases} 1, & \text{ako za svaku valuaciju } w \text{ takvu} \\ & \text{da je } w \sim_x v \text{ važi } I_w(\mathcal{A}) = 1 \\ 0, & \text{inače} \end{cases}$
- $I_v((\exists x)\mathcal{A}) = \begin{cases} 1, & \text{ako postoji valuacija } w \text{ takva} \\ & \text{da je } w \sim_x v \text{ i } I_w(\mathcal{A}) = 1 \\ 0, & \text{inače} \end{cases}$

Objasnimo dodatno definiciju značenja za $I_v((\exists x)\mathcal{A})$. Cilj je određivanje vrednosti formule $(\exists x)\mathcal{A}$ svesti na određivanje vrednosti jednostavnije formule, formule $\mathcal{A}$. Formula $\mathcal{A}$ zavisi od promenljive x, ali i od nekih drugih promenljivih (od svih promenljivih koje u njoj imaju slobodna pojavljivanja). Valuacija v svim tim promenljivim dodeljuje neke vrednosti. I intuitivno je očekivano da istinitosna vrednost formule $(\exists x)\mathcal{A}$ u interpretaciji I_v ne zavisi od $v(x)$, ali da zavisi od vrednosti drugih slobodnih promenljivih u v. Razmotrimo primer $(\exists x)p(x, y)$ i pitanje da li je ova formula tačna u interpretaciji I_v. Intuitivno, tačna je ako možemo pogodno da izaberemo vrednost za x, da je vrednost y nepromenjena, određena valuacijom v i da je za takve vrednosti promenljivih formula $p(x, y)$ tačna. Ako smo nekako pogodno izabrali vrednost za x, a zadržali vrednost za y, time smo, zapravo, definisali novu valuaciju – w, koja ima istu vrednost za y kao i valuacija v. Dakle, ako smo mogli da izaberemo takvu valuaciju, onda je $I_v((\exists x)p(x, y)) = 1$. Opšti slučaj, slučaj $(\exists x)\mathcal{A}$, je analogan: ako možemo da izaberemo vrednost za x, zadržavši vrednosti promenljivih kao u v, tako da je formula $\mathcal{A}$ tačna, onda je $I_v((\exists x)\mathcal{A}) = 1$. Drugim rečima, ako postoji valuacija w takva da je $w \sim_x v$ i $I_w(\mathcal{A}) = 1$, onda je $I_v((\exists x)\mathcal{A}) = 1$. Analogno objašnjenje važi i za definiciju značenja za $I_v((\forall x)\mathcal{A})$.

Može se dokazati da je datom definicijom svakoj formuli $\mathcal{A}$ nad signaturom $\mathcal{L}$ i skupom V pridružena (jedinstvena) vrednost $I_v(\mathcal{A})$. Primetimo da $I_v(\mathcal{A})$ zavisi od $v(x)$ samo ako promenljiva x ima slobodna pojavljivanja u formuli $\mathcal{A}$. Vrednost $I_v(\mathcal{A})$, dakle, zavisi samo od slobodnih promenljivih u formuli $\mathcal{A}$. Specijalno, ako je $\mathcal{A}$ rečenica, vrednost $I_v(\mathcal{A})$ uopšte ne zavisi od v.

Primer 9.16. *Za $\mathcal{L}$-strukturu iz primera 9.12, ako je, na primer, $v(x) = \boxed{B}$ i $v(y) = \boxed{A}$, onda je*
$I_v(above(x,y)) =$ *„jeste iznad“*$(I_v(x), I_v(y)) =$ *„jeste iznad“*$(\boxed{B}, \boxed{A}) = 1$,
jer za konkretnu gomilu kutija sa slike iz primera 9.1 jeste tačno da je $\boxed{B}$ iznad $\boxed{A}$. Slično,
$I_v(above(y,x)) =$ *„jeste iznad“*$(I_v(y), I_v(x)) =$ *„jeste iznad“*$(\boxed{A}, \boxed{B}) = 0$.
Ako je, na primer, $v(x) = \boxed{C}$ i $v(y) = \boxed{A}$, onda je
$I_v(above(x,y)) =$ *„jeste iznad“*$(I_v(x), I_v(y)) =$ *„jeste iznad“*$(\boxed{C}, \boxed{A}) = 0$.

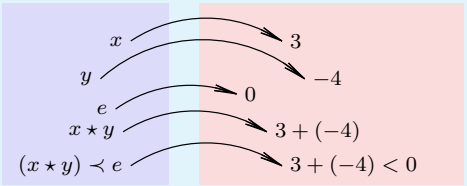
Primer 9.17. *Jedna moguća $\mathcal{L}$-*struktura *za signaturu $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$ (iz primera 9.3) je $(\mathbf{Z}, I)$, gde je $\mathbf{Z}$ skup celih brojeva, a I je funkcija koja predikatske simbole above i below preslikava u relacije $>$ i $<$ nad celim brojevima.*

Ako je, na primer, $v(x) = 3$ i $v(y) = 5$, onda je $I_v(above(x,y))$ jednako $>(I_v(x), I_v(y))$ tj. $>(3,5) = 0$, jer za cele brojeve 3 i 5 je $3 > 5$ netačno.

Vrednost $I_v(\exists x\, above(x,x))$ jednaka je 0, jer ne postoji valuacija w za koju važi $w \sim_x v$ (kako nema relevantnih promenljivih sem x, ovo ograničenje nije relevantno) takva da je $I_w(above(x,x)) = 1$, jer ne postoji ceo broj c (jednak $w(x)$) takav da je $c > c$.

Vrednost $I_v(\forall x\, \neg above(x,x))$ jednaka je 1, jer u svakoj valuaciji w važi da je $I_w(\neg above(x,x)) = 1$, tj. $I_w(above(x,x)) = 0$, jer je $c > c$ netačno za svaki ceo broj c.

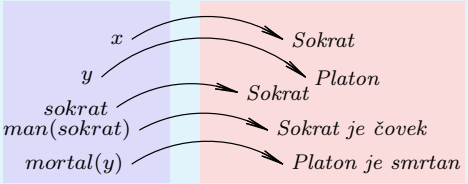
Primer 9.18. *Za signaturu $\mathcal{L} = (\{e_{/0}, \star_{/2}\}, \{\prec_{/2}\})$, iz primera 9.6, jedna moguća $\mathcal{L}$-*struktura *je par $(\mathbf{Z}, I)$, gde je $\mathbf{Z}$ skup celih brojeva, a I funkcija koja simbol e preslikava u ceo broj 0, funkcijski simbol $\star$ u operaciju sabiranja nad celim brojevima, a predikatski simbol $\prec$ u relaciju $<$ nad celim brojevima. Ako je, na primer, $v(x) = 3$, $v(y) = -4$, onda je (koristeći infiksni zapis) $I_v((x \star y) \prec e)$ jednako $I_v(x) + I_v(y) < I_v(e)$, tj. $3 + (-4) < 0$, tj. jednako 1.*



Za istu signaturu moguća je slična interpretacija, sa univezumom koji je skup prirodnih, a ne celih brojeva.

*Za istu signaturu jedna moguća $\mathcal{L}$-*struktura *je i par $(\mathbf{D}, I)$, gde je $\mathbf{D}$ skup dana u nedelji — $\{ponedeljak, utorak, sreda, četvrtak, petak, subota, nedelja\}$, a I funkcija koja simbol e preslikava (na primer) u element $nedelja$, funkcijski simbol $\star$ u funkciju koja za dva dana vraća dan između njih (za ponedeljak i petak, to je sreda, a za ponedeljak i utorak, to je, recimo, ponedeljak) i simbol $\prec$ preslikava u relaciju „prethodni dan“.*

Primer 9.19. *Za signaturu $\mathcal{L} = (\{sokrat_{/0}\}, \{man_{/1}, mortal_{/1}\})$ iz primera 9.5 jedna moguća $\mathcal{L}$-*struktura *je par $(\mathbf{D}, I)$, gde je $\mathbf{D}$ skup svih živih bića, a I funkcija koja simbol sokrat preslikava (na primer) u osobu Sokrat, predikatski simbol man u relaciju „biti čovek”, predikatski simbol mortal u relaciju „biti smrtan”.*



*Za istu signaturu jedna $\mathcal{L}$-*struktura *je i par $(\mathbf{N}, I)$, gde je $\mathbf{N}$ skup prirodnih brojeva, a I funkcija koja simbol sokrat preslikava (na primer) u broj 0, predikatski simbol man u unarnu relaciju „biti složen broj”, predikatski simbol mortal u unarnu relaciju „biti paran broj”.*

Sve navedeno još uvek deluje čudno, pa i beskorisno: ukoliko formulama možemo pridružiti skoro proizvoljno značenje, kakva korist može biti od njih? Zaista, u primenama se obično na neki način sužava skup mogućih interpretacija, na one pogodne za konkretan problem koji se rešava. Na primer, u analizi programskog koda može da bude pogodna interpretacija nad strukturom celih brojeva. U nekim takvim situacijama, valjanost formula može da se ispituje specijalizovanim algoritmima koji podrazumevaju jednu, fiksiranu interpretaciju. Alternativno, u mnogim primenama u kojima nije moguće razmatrati samo jednu fiksiranu interpretaciju ili kada ne postoji specijalizovani algoritam za ispitivanje valjanosti u nekim fiksiranim interpretacijama, formule se konstruišu na način koji eliminiše sve irelevantne interpretacije. To se postiže formulisanjem pogodnih *aksioma* i ispituje se da li je neka formula njihova logička posledica, a to pitanje svodi se na ispitivanje valjanosti u svim mogućim interpretacijama. O oba opisana scenarija biće reči u daljem tekstu.

Definicija 9.12 (Zadovoljivost i valjanost)**.** *Ako je interpretacija I_v određena $\mathcal{L}$-strukturom $\mathfrak{D}$ i valuacijom v i ako za formulu $\mathcal{A}$ nad signaturom $\mathcal{L}$ važi $I_v(\mathcal{A}) = 1$, onda kažemo da je formula $\mathcal{A}$* tačna u interpretaciji *I_v, da je par $(\mathfrak{D}, v)$* model *(eng.* model*) formule $\mathcal{A}$ i pišemo $(\mathfrak{D}, v) \models \mathcal{A}$.*

Formula $\mathcal{A}$ nad signaturom $\mathcal{L}$ je zadovoljiva *(eng.* satisfiable*) ako postoje $\mathcal{L}$-struktura $\mathfrak{D}$ i valuacija v takve da važi $(\mathfrak{D}, v) \models \mathcal{A}$, a inače kažemo da je* kontradiktorna *(eng.* contradictory*) ili* nezadovoljiva *(eng.* unsatisfiable*). Formula $\mathcal{A}$ nad signaturom $\mathcal{L}$ je* poreciva *(eng.* falsifiable*) ako postoje $\mathcal{L}$-struktura $\mathfrak{D}$ i valuacija v takve da ne važi $(\mathfrak{D}, v) \models \mathcal{A}$.*

Ako za neku $\mathcal{L}$-strukturu $\mathfrak{D}$ važi $(\mathfrak{D}, v) \models \mathcal{A}$ za svaku valuaciju v, onda kažemo da je formula $\mathcal{A}$ valjana *(eng.* valid*) u $\mathfrak{D}$, da je $\mathfrak{D}$* model *formule $\mathcal{A}$ i pišemo $\mathfrak{D} \models \mathcal{A}$.*

Ako je formula $\mathcal{A}$ nad signaturom $\mathcal{L}$ valjana u svakoj $\mathcal{L}$-strukturi, onda za tu formulu kažemo da je valjana *i pišemo $\models \mathcal{A}$.*

Analogne definicije uvodimo za skupove formula. Na primer, skup formula Γ je *konzistentan* (ili *zadovoljiv*) ako ima bar jedan model, a inače kažemo da je skup Γ *nekonzistentan*, *nezadovoljiv*, ili *kontradiktoran.*

Primer 9.20. *Formula $\exists x(x \prec e)$ tačna je u prvoj interpretaciji iz primera 9.18 (za univerzum celih brojeva), a nije tačna u drugoj interpretaciji (za univerzum prirodnih brojeva). Dakle, ona nije valjana.*

Primer 9.21. *Formula $\forall x(man(x) \Rightarrow mortal(x))$ je tačna u prvoj interpretaciji iz primera 9.19 (za univerzum koji čine sva živa bića), a nije tačna u drugoj interpretaciji (za univerzum prirodnih brojeva). Dakle, ona nije valjana.*

Primer 9.22. *Tvrdnja „postoji država koja se graniči sa Italijom i Austrijom“ može se formulisati u terminima logike prvog reda na sledeći način:*

$\exists x\ (država(x) \wedge graničeSe(italija, x) \wedge graničeSe(austrija, x))$

gde su $italija$ i $austrija$ simboli konstanti, a $država_{/1}$ i $graničeSe_{/2}$ su predikatski simboli.

Univerzum za prirodnu interpretaciju čini skup svih država sveta, predikatski simboli su interpretirani na prirodan, intuitivan način i u odgovarajućoj interpretaciji navedena formula je tačna za proizvoljnu valuaciju (jer formula nema slobodnih promenljivih, pa njena vrednost ne zavisi od valuacije).

Primer 9.23. *Formule $\forall x\ (p(x) \Rightarrow q(y))$ i $(\forall x\ p(x)) \Rightarrow q(y)$ koje se razlikuju po dosegu kvantifikatora $\forall x$ mogu imati različita značenja. Jedna moguća interpretacija ovih formula određena je domenom koji čine svi studenti koji pohađaju neki kurs, $p(x)$ se interpretira kao „student X će položiti ispit“, a $q(y)$ se interpretira kao „student Y će se obradovati“ (gde su X i Y vrednosti promenljivih x i y u interpretaciji). Prva formula se, onda, interpretira kao „za bilo kojeg studenta koji pohađa kurs važi: ako taj student položi ispit, Y će se obradovati“, tj. „ako neki student položi ispit, Y će se obradovati“, što odgovara i značenju formule $(\exists x\ p(x)) \Rightarrow q(y))$, a druga kao „ako svaki student koji pohađa kurs položi ispit, Y će se obradovati“.*

Već i na osnovu relativno jednostavnih primera, vidi se da ispitivanje valjanosti ili valjanosti u nekoj $\mathcal{L}$-strukturi neposredno, na osnovu definicije, može biti veoma mukotrpno. I zaista, takvo ispitivanje se i ne koristi u primenama. Umesto toga, obično se koriste namenske procedure opisane u narednim poglavljima. One implicitno koriste definicije valjanosti i u stanju su da utvrde da li je neka formula valjana.

9.2 Logičke posledice i logički ekvivalentne formule

Često je veoma važno pitanje da li je neka formula posledica nekih drugih formula. Ovo pitanje može se opisati u terminima pojma *logičke posledice*, analogno kao u slučaju iskazne logike.

Definicija 9.13 (Logička posledica i logička ekvivalencija)**.** *Kažemo da je formula $\mathcal{A}$* logička posledica *(eng.* logical consequence*) skupa formula Γ nad istom signaturom $\mathcal{L}$ i pišemo $\Gamma \models \mathcal{A}$, ako je svaki model za Γ istovremeno i model za $\mathcal{A}$.*

Kažemo da su formule $\mathcal{A}$ i $\mathcal{B}$ nad istom signaturom $\mathcal{L}$ logički ekvivalentne *(eng.* logically equivalent*) i pišemo $\mathcal{A} \equiv \mathcal{B}$ ako važi $\{\mathcal{A}\} \models \mathcal{B}$ i $\{\mathcal{B}\} \models \mathcal{A}$.*

Ako ne važi $\Gamma \models \mathcal{A}$, onda to zapisujemo $\Gamma \not\models \mathcal{A}$. Kao i u iskaznoj logici, formula je logička posledica praznog skupa formula ako i samo ako je valjana. Ako je formula $\mathcal{A}$ logička posledica praznog skupa formula, onda umesto $\{\} \models \mathcal{A}$, to zapisujemo kraće $\models \mathcal{A}$, baš kao što već zapisujemo valjane formule. Kao i u iskaznoj logici, ako je skup Γ kontradiktoran, onda je svaka formula njegova logička posledica. Specijalno, svaka formula je logička posledica skupa $\{\bot\}$. Ako je skup Γ konačan, onda umesto $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k\} \models \mathcal{A}$ pišemo kraće $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k \models \mathcal{A}$.

Primer 9.24. *Ako sa Γ označimo skup formula (iz primera 9.3):*
$\{\ \forall x \forall y(above(x,y) \Rightarrow \neg above(y,x)),\ \forall x \forall y(above(x,y) \Leftrightarrow below(y,x)),\ \forall x \forall y \forall z(above(x,y) \wedge above(y,z) \Rightarrow above(x,z))\ \}$,
onda se može pokazati da važi: $\Gamma \models \forall x \forall y \forall z(above(y,x) \wedge below(z,x) \Rightarrow above(y,z))$.

Primer 9.25. *Ako sa Γ označimo skup formula (iz primera 7.2):*
$\{\ \forall x \forall y \forall z\ (predak(x,y) \wedge predak(y,z) \Rightarrow predak(x,z)),\ \forall x\ predak(otac(x),x)\ \}$
onda se može pokazati da važi:

$$\Gamma \models predak(otac(otac(Ana)), Ana)$$

Ako važi $\mathcal{A} \equiv \mathcal{B}$, tj. ako je svaki model za $\mathcal{A}$ istovremeno i model za $\mathcal{B}$ i obratno, onda u bilo kojoj valuaciji formule $\mathcal{A}$ i $\mathcal{B}$ imaju jednake vrednosti. Tvrđenja oblika $\mathcal{A} \equiv \mathcal{B}$ zovemo *logičkim ekvivalencijama*. Relacija $\equiv$ je, očigledno, relacija ekvivalencije nad skupom formula. Naredna teorema daje vezu između meta i objektnog nivoa: uslov logičke posledice može se svesti na uslov valjanosti i obratno. Ako postoji efektivan način za rešavanje jednog problema onda postoji i efektivan način za rešavanje drugog.

Teorema 9.1. *Za proizvoljne formule $\mathcal{A}$, $\mathcal{B}$, $\mathcal{B}_1$, $\mathcal{B}_2, \ldots, \mathcal{B}_k$ nad istom signaturom $\mathcal{L}$ važi:*

- *$\mathcal{A} \models \mathcal{B}$ akko je formula $\mathcal{A} \Rightarrow \mathcal{B}$ valjana;*
- *$\mathcal{A} \equiv \mathcal{B}$ akko je formula $\mathcal{A} \Leftrightarrow \mathcal{B}$ valjana;*
- *$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k \models \mathcal{A}$ akko $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \ldots \wedge \mathcal{B}_k \models \mathcal{A}$ tj.*
  akko je formula $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \ldots \wedge \mathcal{B}_k \Rightarrow \mathcal{A}$ valjana tj.
  akko je formula $\mathcal{B}_1 \wedge \mathcal{B}_2 \wedge \ldots \wedge \mathcal{B}_k \wedge \neg\mathcal{A}$ nezadovoljiva.

Ako se u proizvoljnoj logičkoj ekvivalenciji iskazne logike, svaka iskazna formula zameni formulom logike prvog reda, biće očigledno dobijena logička ekvivalencija logike prvog reda, na primer, $\neg\neg\mathcal{A} \equiv \mathcal{A}$. Postoje i logičke ekvivalencije logike prvog reda koje ne mogu biti dobijene na taj način.

Primer 9.26. *Neke od shema logičkih ekvivalencija logike prvog reda su:*

$\neg(\exists x)\mathcal{A}$	$\equiv$	$(\forall x)\neg\mathcal{A}$	*De Morganov zakon*
$\neg(\forall x)\mathcal{A}$	$\equiv$	$(\exists x)\neg\mathcal{A}$	*De Morganov zakon*
$\forall x\forall y\mathcal{A}$	$\equiv$	$\forall y\forall x\mathcal{A}$	*zamena poretka kvantifikatora*
$\exists x\exists y\mathcal{A}$	$\equiv$	$\exists y\exists x\mathcal{A}$	*zamena poretka kvantifikatora*
$(\exists x)(\mathcal{A}\vee\mathcal{B})$	$\equiv$	$(\exists x)\mathcal{A}\vee(\exists x)\mathcal{B}$	*zakon distributivnosti $\exists$ prema $\vee$*
$(\forall x)(\mathcal{A}\wedge\mathcal{B})$	$\equiv$	$(\forall x)\mathcal{A}\wedge(\forall x)\mathcal{B}$	*zakon distributivnosti $\forall$ prema $\wedge$*
$(\exists x)(\mathcal{A}\wedge\mathcal{B})$	$\equiv$	$(\exists x)\mathcal{A}\wedge\mathcal{B}$	*zakon distributivnosti $\exists$ prema $\wedge$ (pri čemu $\mathcal{B}$ ne sadrži slobodna pojavljivanja promenljive x)*
$(\forall x)(\mathcal{A}\vee\mathcal{B})$	$\equiv$	$(\forall x)\mathcal{A}\vee\mathcal{B}$	*zakon distributivnosti $\forall$ prema $\vee$ (pri čemu $\mathcal{B}$ ne sadrži slobodna pojavljivanja promenljive x)*
$(\forall x)\mathcal{A}$	$\equiv$	$(\forall y)(\mathcal{A}[x\mapsto y])$	*zakon o preimenovanju vezane promenljive (pri čemu $\mathcal{A}$ ne sadrži slobodna pojavljivanja promenljive y)*
$(\exists x)\mathcal{A}$	$\equiv$	$(\exists y)(\mathcal{A}[x\mapsto y])$	*zakon o preimenovanju vezane promenljive (pri čemu $\mathcal{A}$ ne sadrži slobodna pojavljivanja promenljive y)*

Logička ekvivalencija $\forall x\forall y\mathcal{A}\equiv\forall y\forall x\mathcal{A}$ govori da dva univerzalna kvantifikatora mogu da zamene mesta, a isto važi i za egzistencijalne kvantifikatore. To suštinski znači da u bloku kvantifikatora iste vrste poredak tih kvantifikatora nije bitan. S druge strane, univerzalni i egzistencijalni kvantifikator ne mogu, u opštem slučaju, da razmenjuju mesta. Na primer, formule $(\forall x)(\exists y)\mathcal{A}$ i $(\exists y)(\forall x)\mathcal{A}$ nisu u opštem slučaju logički ekvivalentne.

Logičke ekvivalencije $\forall x\mathcal{A}\equiv\forall y\mathcal{A}[x\mapsto y]$ i $\exists x\mathcal{A}\equiv\exists y\mathcal{A}[x\mapsto y]$ govore o tome da se vezane promenljive mogu preimenovati bez uticaja na istinitosnu vrednost formule, kao što je ranije već nagovešteno.

Naredna teorema kaže da ako se u formuli $\mathcal{A}$ zameni neka njena potformula logički ekvivalentnom formulom, biće dobijena formula koja je logički ekvivalentna formuli $\mathcal{A}$.

Teorema 9.2 (Teorema o zameni)**.** *Ako važi $\mathcal{C}\equiv\mathcal{D}$, onda je $\mathcal{A}[\mathcal{C}\mapsto\mathcal{D}]\equiv\mathcal{A}$.*

Primer 9.27. *Na osnovu logičkih ekvivalencija iz primera 8.14 i 9.26 važi* $\neg(\exists x)(\mathcal{A}\wedge\neg\mathcal{B})\equiv(\forall x)\neg(\mathcal{A}\wedge\neg\mathcal{B})\equiv(\forall x)(\neg\mathcal{A}\vee\neg\neg\mathcal{B})\equiv(\forall x)(\neg\mathcal{A}\vee\mathcal{B})\equiv(\forall x)(\mathcal{A}\Rightarrow\mathcal{B})$.

Iz $\neg(\exists x)(\mathcal{A}\wedge\neg\mathcal{B})\equiv(\forall x)(\mathcal{A}\Rightarrow\mathcal{B})$*, na osnovu teoreme 9.1 sledi da je formula* $\neg(\exists x)(\mathcal{A}\wedge\neg\mathcal{B})\Leftrightarrow(\forall x)(\mathcal{A}\Rightarrow\mathcal{B})$ *valjana.*

9.3 Ispitivanje zadovoljivosti i valjanosti u logici prvog reda

U praktičnim primenama, onda kada je neki cilj formulisan na jeziku logike prvog reda, centralni problem postaje ispitivanje da li je neka iskazna formula valjana, da li je zadovoljiva i slično. Početno pitanje je da li su ti problemi uopšte odlučivi, tj. da li postoje algoritmi koji *uvek* mogu da daju odgovor na njih. Ukoliko takvi algoritmi ne postoje, pitanje je da li postoje za neke klase formula logike prvog reda. Slično kao u iskaznoj logici, problemi ispitivanja valjanosti i zadovoljivosti su povezani: formula je valjana ako i samo ako je njena negacija nezadovoljiva.

9.3.1 Logika prvog reda i odlučivost

Za razliku od iskazne logike, u logici prvog reda problem ispitivanja valjanosti nije odlučiv, tj. ne postoji algoritam koji za proizvoljnu formulu može da utvrdi da li je valjana ili nije. Naravno, isto važi i za problem ispitivanja (ne)zadovoljivosti. Ipak, ovi problemi su poluodlučivi: postoji algoritam koji za proizvoljnu valjanu formulu može da utvrdi da je valjana, ali ne može za proizvoljnu formulu koja nije valjana da utvrdi da nije valjana (obično se u takvim situacijama algoritam ne zaustavlja). Kako je formula valjana akko je njena negacija nezadovoljiva, očito je i problem nezadovoljivosti poluodlučiv: postoji algoritam koji za proizvoljnu nezadovoljivu formulu može da utvrdi da je nezadovoljiva, ali ne može za proizvoljnu zadovoljivu formulu da utvrdi da je zadovoljiva. Kako je problem ispitivanja valjanosti neodlučiv, neodlučiv je (mada je poluodlučiv) i problem ispitivanja da li važi $\Gamma\models\Phi$ (za proizvoljan skup Γ i formulu Φ). Procedure za proveravanje valjanosti i zadovoljivosti u logici prvog reda zovu se *dokazivači za logiku prvog reda*. Te procedure ne ispituju valjanost i zadovoljivost neposredno na osnovu definicije, već koristeći elegantnije postupke (kao što je *rezolucija* koja

je opisana u nastavku). Zahvaljujući takvim opštim procedurama, za mnoge valjane formule može se efikasno dokazati da su valjane.

U praktičnim primenama, često je za neko tvrđenje relevantna samo jedna konkretna $\mathcal{L}$-struktura $\mathcal{D}$ (na primer, za univerzum koji čine realni brojevi, ili građani neke države, ili skup kutija u nekoj sobi). To tvrđenje može biti valjano u toj konkretnoj strukturi $\mathcal{D}$, a da nije valjano (te ne može biti dokazano dokazivačem za logiku prvog reda opšte namene). Dodatno, ukoliko se razmatra samo jedna vrsta modela (tj. pitanje $\mathcal{D} \models \Phi$), onda za neke vrste formula ispitivanje valjanosti može da bude odlučiv problem.

Primer 9.28. *Ako se razmatraju formule nad signaturom (opisanom u primeru 9.3)* $\mathcal{L} = (\{\}, \{above_{/2}, below_{/2}\})$ *i samo* $\mathcal{L}$*-struktura (opisana u primeru 9.12) u kojoj je univerzum skup konkretnih kutija prikazanih na slici u okviru primera 9.1, onda je tako specijalizovan problem ispitivanja valjanosti odlučiv (može se dizajnirati jednostavan algoritam koji rešava ovaj problem).*

Primer 9.29. *Razmotrimo signaturu* $\mathcal{L} = (\{0_{/0}, s_{/1}, +_{/2}\}, \{<_{/2}, =_{/2}\})$ *i* $\mathcal{L}$*-strukturu* $(\mathbf{Z}, I)$*, gde je* $\mathbf{Z}$ *skup celih brojeva, a* I *funkcija koja (prirodno) simbol* 0 *preslikava u ceo broj* 0*, simbol* s *preslikava u operaciju sledbenik, funkcijski simbol* $+$ *u operaciju sabiranja nad celim brojevima, a predikatske simbole* $<$ *i* $=$ *u relacije* $<$ *i* $=$ *nad celim brojevima. Ovakva* $\mathcal{L}$*-struktura određuje teoriju koju zovemo* linearna aritmetika nad celim brojevima *i označavamo* LIA*. Ako je formula* Φ *valjana u ovoj* $\mathcal{L}$*-strukturi, onda to zapisujemo* $\models_{LIA} \Phi$*. Na primer, valjane su u* LIA *formule* $\forall x \exists y\ (x < y + s(0))$ *i* $x + y = y + x$ *(iako nisu valjane u nekim drugim interpretacijama). Ispitivanje da li važi* $\models_{LIA} \Phi$ *je odlučiv problem.*

Za ispitivanje valjanosti u konkretnoj strukturi opisanoj u primeru 9.29 i drugim sličnim situacijama koriste se specijalizovani algoritmi, koje obično zovemo SMT rešavači (od engleskog *satisfiability modulo theory* — zadovoljivost u odnosu na teoriju, jer oni obično problem valjanosti svode na problem zadovoljivosti). Rešavač za ispitivanje $\models_{LIA} \Phi$ zovemo SMT *rešavač za teoriju* LIA. Postoji mnoštvo takvih rešavača i obično su izgrađeni oko nekog SAT rešavača. Neki od često korišćenih SMT rešavača su rešavači za linearnu aritmetiku nad celim brojevima, za linearnu aritmetiku nad racionalnim brojevima, za teoriju neinterpretiranih funkcija, za teoriju nizova, za teoriju lista, itd. Problemi zadovoljivosti (pa i valjanosti) u nabrojanim slučajevima su odlučivi i SMT rešavači predstavljaju *procedure odlučivanja*: vraćaju odgovor „da“ ako je zadata formula zadovoljiva u relevantnoj strukturi i „ne“, inače. SMT rešavači imaju brojne primene, posebno u verifikaciji softvera i hardvera i u rešavanju raznovrsnih problema ograničenja.

U mnogim primenama, za zadata tvrđenja nisu relevantne sve moguće interpretacije i $\mathcal{L}$-strukture, već samo $\mathcal{L}$-strukture koje ispunjavaju određene uslove. Ti uslovi $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ nameću neke specifičnosti željenom značenju formula i čine *aksiome* konkretne teorije (na primer, geometrije). Tada se ispitivanje valjanosti formule Φ pod zadatim uslovima svodi na ispitivanje da li važi $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \models \Phi$. Obično nije lako napraviti namenski algoritam koji može da proveri takva tvrđenja, tj. nije lako napraviti specijalizovani rešavač (kakvi su SMT rešavači) za proizvoljnu teoriju zadatu aksiomama. Zato se u takvim situacijama obično dokazivanje tvrđenja oblika:

$\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \models \Phi$

svodi na dokazivanje da je naredna formula nezadovoljiva (u proizvoljnoj interpretaciji):

$\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \ldots \wedge \mathcal{A}_n \wedge \neg\Phi$

Time se od specifičnog razmatranja (vezanog za neke konkretne interpretacije) dolazi do opšteg razmatranja i nezadovoljivost navedene formule može da se ispita nekom opštom procedurom za ispitivanje nezadovoljivosti u logici prvog reda (zbog čega dokazivače za logiku prvog reda možemo neformalno da zovemo i *univerzalnim rešavačima za logiku prvog reda*).

Primer 9.30. *Ako je* $\Gamma = \{\ \forall x \forall y \forall z\ \ (predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z))\ ,\ \ \forall x\ predak(otac(x), x)\ \}$ *onda važi* $\Gamma \models predak(otac(otac(Ana)), Ana)$ *ako i samo ako je formula* $\forall x \forall y \forall z\ (predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z))\ \wedge\ \forall x\ predak(otac(x), x)\ \ \wedge \neg predak(otac(otac(Ana)), Ana)$ *nezadovoljiva.*

Opisana dva scenarija dva su najčešća scenarija u praktičnim primenama logike prvog reda: ili se koristi namenski rešavač za valjanost u nekoj konkretnoj interpretaciji ili se koristi dokazivač za logiku prvog reda uz pogodno dizajnirane aksiome.

9.3.2 Normalne forme

Iako se zadovoljivost i valjanost mogu ispitivati za formule proizvoljnog oblika, daleko je jednostavnije algoritme ispitivanja formulisati za formule nekog posebnog oblika. Zbog toga se definišu *normalne forme* i algoritmi

kojima se neka formula transformiše u te normalne forme. Pod *transformacijom* se podrazumeva konstruisanje formule koja je, na primer, logički ekvivalentna polaznoj formuli i zadovoljava neka sintaksička ograničenja. U nastavku će biti opisan niz koraka koji vodi od formule čija se zadovoljivost razmatra do formule koja ima specifičnu sintaksičku formu i zadovoljiva je ako i samo ako je zadovoljiva polazna formula.

Definicija 9.14 (Preneks normalna forma)**.** *Kažemo da je formula u* preneks normalnoj formi *ako je ona oblika*

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \mathcal{A}$$

pri čemu je Q_i ili $\forall$ ili $\exists$ i $\mathcal{A}$ ne sadrži kvantifikatore, kao ni slobodne promenljive osim (eventualno) promenljivih $x_1, x_2, \dots, x_n$.

Ako je rečenica (zatvorena formula) $\mathcal{A}$ logički ekvivalentna formuli $\mathcal{B}$ i formula $\mathcal{B}$ je u preneks normalnoj formi, onda kažemo da je formula $\mathcal{B}$ preneks normalna forma formule $\mathcal{A}$. Jedna formula može da ima više preneks normalnih formi (na primer, i formula $(\forall x)(\forall y)(\mathcal{A}(x) \wedge \mathcal{B}(y))$ i formula $(\forall y)(\forall x)(\mathcal{B}(y) \wedge \mathcal{A}(x))$ su preneks normalne forme formule $(\forall x)\mathcal{A}(x) \wedge (\forall y)\mathcal{B}(y)$). Slično, jedna formula koja je u preneks normalnoj formi može biti preneks normalna forma za više formula. Transformisanje formule u preneks normalnu formu može biti realizovano procedurom prikazanom na slici 9.1. Kada se u proceduri govori o „primeni neke logičke ekvivalencije“, misli se, kao i ranije, na korišćenje logičkih ekvivalencija na osnovu teoreme o zameni (teorema 9.2). Radi jednostavnosti procedure i rezultujuće formule, iako to nije neophodno, najpre se eliminišu veznici $\Leftrightarrow$ i $\Rightarrow$.

Procedura na ulazu očekuje zatvorenu formulu, formulu bez slobodnih promenljivih. To nije bitno ograničenje, jer se procedura obično koristi u okviru šireg postupka ispitivanja zadovoljivosti, a tada se, u ranijim fazama, za svaku promenljivu može odrediti način na koji je kvantifikovana.

Algoritam: PRENEX

Ulaz: Zatvorena formula logike prvog reda

Izlaz: Preneks normalna forma zadate formule

1: **dok god** je to moguće **radi**
2: primeni neku od logičkih ekvivalencija:
$\mathcal{A} \Leftrightarrow \mathcal{B} \equiv (\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A})$,
$\mathcal{A} \Rightarrow \mathcal{B} \equiv \neg\mathcal{A} \vee \mathcal{B}$;
3: **dok god** je to moguće **radi**
4: primeni neku od logičkih ekvivalencija:
$\neg(\mathcal{A} \wedge \mathcal{B}) \equiv \neg\mathcal{A} \vee \neg\mathcal{B}$,
$\neg(\mathcal{A} \vee \mathcal{B}) \equiv \neg\mathcal{A} \wedge \neg\mathcal{B}$,
$\neg(\forall x)\mathcal{A} \equiv (\exists x)\neg\mathcal{A}$,
$\neg(\exists x)\mathcal{A} \equiv (\forall x)\neg\mathcal{A}$;
5: **dok god** je to moguće **radi**
6: primeni neku od logičkih ekvivalencija:
$(\forall x\mathcal{A}) \wedge \mathcal{B} \equiv (\forall x)(\mathcal{A} \wedge \mathcal{B})$,
$(\forall x\mathcal{A}) \vee \mathcal{B} \equiv (\forall x)(\mathcal{A} \vee \mathcal{B})$,
$\mathcal{B} \wedge (\forall x)\mathcal{A} \equiv (\forall x)(\mathcal{B} \wedge \mathcal{A})$
$\mathcal{B} \vee (\forall x)\mathcal{A} \equiv (\forall x)(\mathcal{B} \vee \mathcal{A})$,
$(\exists x\mathcal{A}) \wedge \mathcal{B} \equiv (\exists x)(\mathcal{A} \wedge \mathcal{B})$,
$(\exists x\mathcal{A}) \vee \mathcal{B} \equiv (\exists x)(\mathcal{A} \vee \mathcal{B})$,
$\mathcal{B} \wedge (\exists x)\mathcal{A} \equiv (\exists x)(\mathcal{B} \wedge \mathcal{A})$,
$\mathcal{B} \vee (\exists x)\mathcal{A} \equiv (\exists x)(\mathcal{B} \vee \mathcal{A})$,
pri čemu x nema slobodna pojavljivanja u formuli $\mathcal{B}$; ako x ima slobodna pojavljivanja u $\mathcal{B}$, onda treba najpre preimenovati promenljivu x u formuli $(\forall x)\mathcal{A}$ (odnosno u formuli $(\exists x)\mathcal{A}$).

Slika 9.1: Algoritam PRENEX.

Primer 9.31. *Razmotrimo formulu*

$$\forall x\, p(x) \wedge \forall x \exists y \forall z (q(y, z) \Rightarrow r(g(x), y)) \,.$$

Primenom algoritma PRENEX najpre se (u okviru koraka 1) dobija formula

$$\forall x\, p(x) \wedge \forall x \exists y \forall z(\neg q(y,z) \vee r(g(x),y)) .$$

Nakon koraka

$$\forall x(p(x) \wedge \forall x \exists y \forall z(\neg q(y,z) \vee r(g(x),y))) ,$$

kako je promenljiva x slobodna u $p(x)$, najpre ćemo preimenovati vezanu promenljivu x u u (u okviru formule $\forall x \exists y \forall z(\neg q(y,z) \vee r(g(x),y))$):

$$\forall x(p(x) \wedge \forall u \exists y \forall z(\neg q(y,z) \vee r(g(u),y))) .$$

Nakon toga kvantifikatori $\forall u$, $\exists y$, $\forall z$ mogu, jedan po jedan, biti pomereni na početak formule:

$$\forall x \forall u \exists y \forall z(p(x) \wedge (\neg q(y,z) \vee r(g(u),y))) .$$

Korektnost navedenog algoritma može se dokazati slično kao korektnost procedure za transformisanje formule u konjunktivnu normalnu formu (teorema 8.5).

Teorema 9.3 (Korektnost algoritma PRENEX). *Algoritam PRENEX se zaustavlja i zadovoljava sledeće svojstvo: ako je $\mathcal{A}$ ulazna (zatvorena) formula, onda je izlazna formula $\mathcal{A}'$ u preneks normalnoj formi i važi $\mathcal{A} \equiv \mathcal{A}'$.*

U nekim situacijama moguće je primeniti neki korak navedenog algoritma na više od jednog načina. Na primer, formulu $(\forall x)p(x) \wedge (\exists y)q(y)$ moguće je transformisati i u $(\forall x)(p(x) \wedge (\exists y)q(y))$ i u $(\exists y)((\forall x)p(x) \wedge q(y))$. Obe ove formule su, naravno, logički ekvivalentne sa polaznom formulom. Ipak, u situacijama kada postoji izbor, uvek ćemo radije egzistencijalni kvantifikator učiniti spoljnim u odnosu na univerzalni nego obratno. Takav prioritet uvodimo zarad jednostavnijeg koraka skolemizacije (o kojem će biti reči u nastavku).

Definicija 9.15 (Konjunktivna normalna forma). *Formula logike prvog reda koja ne sadrži kvantifikatore je u* konjunktivnoj normalnoj formi *ako je oblika*

$$\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \ldots \wedge \mathcal{A}_n$$

pri čemu je svaka od formula $\mathcal{A}_i$ $(1 \leq i \leq n)$ klauza (tj. disjunkcija literala).

Konjunktivna normalna forma formule predikatske logike (bez kvantifikatora) može se dobiti na način analogan slučaju iskazne logike (videti poglavlje 8.3.2). Dobijena formula logički je ekvivalentna polaznoj formuli.

Primer 9.32. *Konjunktivna normalna forma formule*

$$p(x) \wedge (q(y,z) \Rightarrow r(g(u),y))$$

je formula

$$p(x) \wedge (\neg q(y,z) \vee r(g(u),y)) .$$

Nakon primene algoritama PRENEX i KNF, formula je u obliku

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \mathcal{A},$$

pri čemu je Q_i ili $\forall$ ili $\exists$ i formula $\mathcal{A}$ je u konjunktivnoj normalnoj formi. Za razmatranje zadovoljivosti ovakve formule bilo bi još pogodnije ako bi svi kvantifikatori Q_i bili univerzalni. Međutim, ne može se proizvoljna formula (na primer, formula $(\exists x)p(x)$) transformisati u formulu takvog oblika a da joj je, pri tome, logički ekvivalentna. No, moguće je nešto drugo: proizvoljna formula može se transformisati u formulu oblika $\forall x_1 \forall x_2 \ldots \forall x_n \mathcal{A}$ koja je zadovoljiva ako i samo ako je zadovoljiva polazna formula (tada kažemo da su te dve formule *slabo ekvivalentne*, analogno kao u slučaju iskazne logike). Tu transformaciju, transformaciju koja „eliminiše“ egzistencijalne kvantifikatore, zovemo *skolemizacija* (po matematičaru Skolemu koji ih je prvi koristio). Ona se zasniva na proširivanju polazne signature novim funkcijskim simbolima. Te dodatne funkcijske simbole zovemo *Skolemovim konstantama* (za funkcijske simbole arnosti 0) i *Skolemovim funkcijama*.

Pretpostavimo da rečenica počinje egzistencijalnim kvantifikatorom: $\exists y \mathcal{A}$. Treba izabrati novi simbol konstante d koji se ne pojavljuje u signaturi, obrisati kvantifikator i zameniti promenljivu y simbolom d. Na taj

način formula $\exists y\mathcal{A}$ transformiše se u formulu $\mathcal{A}[y \mapsto d]$. Može se dokazati da je formula $\exists y\mathcal{A}$ zadovoljiva ako i samo ako je formula $\mathcal{A}[y \mapsto d]$ zadovoljiva.

Ako rečenica počinje nizom od n univerzalnih kvantifikatora: $\forall x_1\ \forall x_2\ \ldots\ \forall x_n\ \exists y\ \mathcal{A}$, onda uvodimo novi funkcijski simbol f arnosti n koji do tada nije postojao u signaturi. Polazna formula biće onda transformisana u formulu $\forall x_1 \forall x_2 \ldots \forall x_n\ \mathcal{A}[y \mapsto f(x_1, x_2, \ldots, x_n)]$. Može se dokazati da je formula $\forall x_1 \forall x_2 \ldots \forall x_n \exists y \mathcal{A}$ zadovoljiva ako i samo ako je formula $\forall x_1 \forall x_2 \ldots \forall x_n \mathcal{A}[y \mapsto f(x_1, x_2, \ldots, x_n)]$ zadovoljiva. (Primetimo da je uvođenje nove konstante samo specijalni slučaj uvođenja novog funkcijskog simbola.)

Primer 9.33. *Skolemizacijom se formula*

$$\forall x \forall u \exists y \forall z\ (p(x) \wedge (\neg q(y, z) \vee r(g(u), y)))$$

transformiše u formulu

$$\forall x \forall u \forall z\ (p(x) \wedge (\neg q(h(x, u), z) \vee r(g(u), h(x, u)))) .$$

Teorema 9.4 (Teorema o skolemizaciji)**.** *Ako je formula $\mathcal{B}$ nad signaturom $\mathcal{L}'$ dobijena skolemizacijom od rečenice $\mathcal{A}$ nad signaturom $\mathcal{L}$ koja je u preneks normalnoj formi, onda je $\mathcal{A}$ zadovoljiva ako i samo ako je $\mathcal{B}$ zadovoljiva.*

Transformisanje formule u preneks normalnu formu, pa transformisanje dela formule bez kvantifikatora u konjunktivnu normalnu formu, pa eliminisanje skolemizacijom egzistencijalnih kvantifikatora, jednog po jednog, dovodi do specifične sintaksičke forme koju zovemo *klauzalna forma*:

Definicija 9.16 (Klauzalna forma)**.** *Formula je u* klauzalnoj *formi ako je oblika*

$$\forall x_1 \forall x_2 \ldots \forall x_n \mathcal{A}$$

gde je $\mathcal{A}$ formula bez kvantifikatora, u konjunktivnoj normalnoj formi i bez slobodnih promenljivih osim, eventualno, promenljivih $x_1, x_2, \ldots, x_n$.

Ako je formula $\forall x_1 \forall x_2 \ldots \forall x_n \mathcal{A}$ u klauzalnoj formi, onda se često u zapisu izostavljaju kvantifikatori i piše samo $\mathcal{A}$, podrazumevajući da se misli na univerzalno zatvorenje formule $\mathcal{A}$.

Teorema 9.5. *Neka je formula $\mathcal{B}$ (u klauzalnoj formi) dobijena od rečenice $\mathcal{A}$ uzastopnom primenom sledećih postupaka:*

- *transformisanje formule u preneks normalnu formu;*
- *transformisanje dela formule bez kvantifikatora u konjunktivnu normalnu formu;*
- *skolemizacija.*

Tada je formula $\mathcal{A}$ zadovoljiva ako i samo ako je $\mathcal{B}$ zadovoljiva.

Navedena teorema kaže da transformisanje formule u klauzalnu formu ne daje njoj logički ekvivalentnu formulu, već formulu koja joj je samo slabo ekvivalentna. Međutim, to je dovoljno u kontekstu ispitivanja zadovoljivosti: ako se ispituje zadovoljivost rečenice $\mathcal{A}$, dovoljno je ispitati zadovoljivost formule $\mathcal{B}$ koja je u klauzalnoj formi i slabo ekvivalentna sa formulom $\mathcal{A}$. To daje i put za dokazivanje da je neka formula $\mathcal{A}$ valjana: dovoljno je dokazati da je formula $\neg\mathcal{A}$ nezadovoljiva, pa je dovoljno i dokazati da je klauzalna forma formule $\neg\mathcal{A}$ nezadovoljiva.

Primer 9.34. *Ako je* $\Gamma = \{\ \forall x \forall y \forall z\ \ (predak(x, y) \wedge predak(y, z) \Rightarrow\ predak(x, z))\ ,\ \forall x\ predak(otac(x), x)\ \}$ *onda važi*
$\Gamma \models predak(otac(otac(Ana)), Ana)$ *ako i samo ako je formula*
$\forall x \forall y \forall z\ \ (predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z)) \wedge \forall x\ predak(otac(x), x) \wedge \neg predak(otac(otac(Ana)), Ana)$
nezadovoljiva. Klauzalna forma ove formule je
$\forall x \forall y \forall z \forall u\ \ (\neg predak(x, y) \vee \neg predak(y, z) \vee predak(x, z)) \wedge predak(otac(u), u) \wedge \neg predak(otac(otac(Ana)), Ana)$
i za nju je potrebno proveriti da li je nezadovoljiva.

Primer 9.35. *Formula $\mathcal{A} = (\forall x)p(x,x) \Rightarrow (\forall y)p(y,y)$ nad signaturom $\mathcal{L}$ je valjana. To se može dokazati na sledeći način.*

Formula $\neg\mathcal{A}$ je jednaka $\neg((\forall x)p(x,x) \Rightarrow (\forall y)p(y,y))$ i njena preneks normalna forma je $(\exists y)(\forall x)(p(x,x) \wedge \neg p(y,y))$. Skolemizacijom dobijamo formulu $p(x,x) \wedge \neg p(c,c)$, gde je c novi simbol konstante. Neka je $\mathcal{L}'$ signatura dobijena proširivanjem signature $\mathcal{L}$ simbolom c. Može se pokazati da je formula $p(x,x) \wedge \neg p(c,c)$ nezadovoljiva – ili neposredno koristeći definiciju zadovoljivosti ili koristeći metod rezolucije opisan u nastavku. Kako je formula $p(x,x) \wedge \neg p(c,c)$ nezadovoljiva, polazna formula $\mathcal{A}$ je valjana.

9.3.3 Unifikacija

Problem unifikacije je problem ispitivanja da li postoji supstitucija koja čini dva izraza (dva terma ili dve formule) jednakim.

Definicija 9.17 (Unifikabilnost i unifikator)**.** *Ako za izraze e_1 i e_2 postoji supstitucija σ takva da važi $e_1\sigma = e_2\sigma$, onda kažemo da su izrazi e_1 i e_2* unifikabilni *i da je supstitucija σ* unifikator *(eng.* unifier*) za ta dva izraza.*

Primer 9.36. *Neka je term t_1 jednak $g(x,z)$, neka je term t_2 jednak $g(y,f(y))$ i neka je σ supstitucija $[y \mapsto x, z \mapsto f(x)]$. Tada je i $t_1\sigma$ i $t_2\sigma$ jednako $g(x,f(x))$, pa su termovi t_1 i t_2 unifikabilni, a σ je (jedan) njihov unifikator. Unifikator termova t_1 i t_2 je na primer, i $[x \mapsto a, y \mapsto a, z \mapsto f(a)]$. Termovi $g(x,x)$ i $g(y,f(y))$ nisu unifikabilni.*

Dva unifikabilna izraza mogu da imaju više unifikatora. Za dva unifikatora σ_1 i σ_2 kažemo da su jednaka do na preimenovanje promenljivih ako postoji supstitucija λ koja je oblika $[v_1' \mapsto v_1'', v_2' \mapsto v_2'', \ldots, v_n' \mapsto v_n'']$, pri čemu su v_i' i v_i'' simboli promenljivih i važi $\sigma_1\lambda = \sigma_2$.

Primer 9.37. *Naredni parovi unifikatora izraza $g(x,z)$ i $g(y,f(y))$ jednaki su do na preimenovanje promenljivih:*

- *$\sigma_1 = [y \mapsto x, z \mapsto f(x)]$, $\sigma_2 = [y \mapsto x, z \mapsto f(x)]$, zaista, za $\lambda = [\,]$ važi $\sigma_1\lambda = \sigma_2$;*
- *$\sigma_1 = [y \mapsto x, z \mapsto f(x)]$, $\sigma_2 = [x \mapsto y, z \mapsto f(y)]$, zaista, za $\lambda = [x \mapsto y]$ važi $\sigma_1\lambda = \sigma_2$;*
- *$\sigma_1 = [y \mapsto x, z \mapsto f(x)]$, $\sigma_2 = [x \mapsto u, y \mapsto u, z \mapsto f(u)]$, zaista, za $\lambda = [x \mapsto u]$ važi $\sigma_1\lambda = \sigma_2$.*

Definicija 9.18 (Najopštiji unifikator)**.** *Supstitucija σ je* najopštiji unifikator *(eng.* most general unifier*) za izraze e_1 i e_2 ako svaki unifikator τ izraza e_1 i e_2 može biti predstavljen kao kompozicija supstitucije σ i neke supstitucije μ.*

Primer 9.38. *Izrazi $predak(otac(u),u)$ i $predak(x,y)$ su unifikabilni i njihov najopštiji unifikator je $[x \mapsto otac(u), y \mapsto u]$.*

Svaka dva unifikabilna izraza imaju najopštiji unifikator, jedinstven do na preimenovanje promenljivih. Na slici 9.2 dat je opis opšteg algoritma za određivanje najopštijeg unifikatora za niz parova izraza. Algoritam unifikacije ili vraća traženu supstituciju ili se zaustavlja sa neuspehom, ukazujući na to da tražena supstitucija ne postoji. Primetimo da je korak 6 algoritma *(application)* (kao i neke druge korake algoritma) moguće u opštem slučaju primeniti na više načina. Bilo koji od tih načina vodi istom rezultatu — neuspehu (ako ne postoji traženi unifikator) ili jednom od unifikatora koji se mogu razlikovati samo do na preimenovanje promenljivih.

Primer 9.39. *Ilustrujmo rad algoritma* Najopštiji unifikator *na primeru sledeća dva para termova:*
$(g(y),x)$, $(f(x,h(x),y), f(g(z),w,z))$
Primenom koraka 3 (orientation) *dobijamo*
$(x,g(y)), (f(x,h(x),y), f(g(z),w,z))$.
Primenom koraka $4(a)$ (decomposition) *dobijamo*
$(x,g(y)), (x,g(z)), (h(x),w), (y,z)$.

Algoritam: Najopštiji unifikator

Ulaz: Niz parova izraza $(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)$

Izlaz: Najopštiji unifikator (ako on postoji) σ takav da važi $s_1\sigma = t_1\sigma, s_2\sigma = t_2\sigma, \ldots, s_n\sigma = t_n\sigma$.

dok god je moguće primeniti neko od navedenih pravila **radi**
    {Korak 1 (*factoring*):}
    ako postoji par koji ima više od jednog pojavljivanja **onda**
        obriši sva njegova pojavljivanja osim jednog;
    {Korak 2 (*tautology*):}
    ako postoji par (t, t) **onda**
        obriši ga;
    {Korak 3 (*orientation*):}
    ako postoji par (t, x), gde je x promenljiva, a t nije promenljiva **onda**
        zameni par (t, x) parom (x, t);
    ako postoji par (s, t), gde ni s ni t nisu promenljive **onda**
        ako je s jednako $\varphi(u_1, u_2, \ldots, u_k)$ i t je jednako $\varphi(v_1, v_2, \ldots, v_k)$ (gde je φ funkcijski ili predikatski simbol) **onda**
            {Korak 4(a) (*decomposition*):}
            dodaj parove $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ i obriši par (s, t);
        inače
            {Korak 4(b) (*collision*):}
            zaustavi rad i kao rezultat vrati *neuspeh*;
    {Korak 5 (*cycle*): }
    ako postoji par (x, t) takav da je x promenljiva i t term koji sadrži x **onda**
        zaustavi rad i kao rezultat vrati *neuspeh*;
    {Korak 6 (*application*): }
    ako postoji (x, t), gde je x promenljiva a t term koji ne sadrži x i x se pojavljuje i u nekim drugim parovima **onda**
        primeni supstituciju $[x \mapsto t]$ na sve druge parove;
vrati tekući skup parova kao najopštiji unifikator.

Slika 9.2: Algoritam Najopštiji unifikator.

Primenom koraka 3 (orientation) *dobijamo*
$(x, g(y)), (x, g(z)), (w, h(x)), (y, z)$.
Korak 6 (application) *moguće je primeniti na više načina. Primenom za par* (y, z) *dobijamo*
$(x, g(z)), (x, g(z)), (w, h(x)), (y, z)$.
Primenom koraka 1 (factoring) *dobijamo*
$(x, g(z)), (w, h(x)), (y, z)$.
Primenom koraka 6 (application) *dobijamo*
$(x, g(z)), (w, h(g(z))), (y, z)$.
Ovaj niz parova određuje traženi najopštiji unifikator $\sigma = [x \mapsto g(z),\ w \mapsto h(g(z)),\ y \mapsto z]$.

Primer 9.40. *Razmotrimo sledeći par izraza* $(g(x, x), g(y, f(y)))$.
Primenom koraka $4(a)$ (decomposition) *dobijamo dva para izraza:*
$(x, y), (x, f(y))$.
Korak 6 (application) *može se primeniti na dva načina. Primenom za par* (x, y) *dobija se* (x, y), $(y, f(y))$, *odakle se, primenom koraka* 5 (cycle) *dolazi do neuspeha. Dakle, polazni izrazi nisu unifikabilni.*
Naravno, isti zaključak bio bi dobijen i da je korak 6 (application) *bio primenjen za par* $(x, f(y))$. *Tada se dobija* $(f(y), y)$, $(x, f(y))$, *odakle se, primenom koraka 3* (orientation) *i koraka* 5 (cycle) *dolazi do neuspeha.*

Teorema 9.6 (Korektnost algoritma Najopštiji unifikator). *Algoritam Najopštiji unifikator zadovoljava sledeće uslove:*

- *zaustavlja se;*
- *ako vrati supstituciju, onda je ona najopštiji unifikator za dati niz parova izraza;*
- *ako se algoritam zaustavi i vrati neuspeh, onda ne postoji unifikator za dati niz parova izraza.*

Navedeni algoritam za unifikaciju nije efikasan. Postoje znatno efikasniji algoritmi i mnogi od njih zasnovani su na korišćenju pogodnih struktura podataka i implicitnom primenjivanju supstitucije (iz koraka 6). Neki od tih algoritama imaju linearnu vremensku složenost (po broju polaznih parova) i unifikatore predstavljaju implicitno (u vidu grafova). No, u opštem slučaju, najopštiji unifikator može imati i eksponencijalnu dužinu (po broju polaznih parova), te ga nije moguće eksplicitno predstaviti u linearnom vremenu. To ilustruje sledeći primer.

Primer 9.41. *Za skup parova*

$$(x_1, f(x_0, x_0))$$
$$(x_2, f(x_1, x_1))$$
$$\dots$$
$$(x_n, f(x_{n-1}, x_{n-1}))$$

Najopštiji unifikator sadrži zamenu $x_n \mapsto t$, gde je t term koji sadrži samo simbole x_0 i f, pri čemu ima $2^n - 1$ pojavljivanja simbola f.

Unifikacija ima mnoge primene. Neke od njih su u transformisanju izraza korišćenjem raspoloživih pravila, a neke u metodu rezolucije.

Primer 9.42. *U primeru 8.17, obrazloženo je zašto se u C programu uslov*

```
if (!(!a && !b) || c)
```

može zameniti sledećim redom:

```
if (a || b || c)
```

Kompilatori vrše takve izmene tokom prevođenja, na način pojednostavljeno opisan u nastavku. Kompilator može da ima raspoloživa pravila poput

`!(!x && !y)` $\longmapsto$ `x || y`.

Kompilator razmatra izraz `!(!a && !b) || c` *i njegove podizraze i ispituje da li na njih može da primeni neko od raspoloživih pravila. Pravilo je primenljivo ako se njegova leva strana može unifikovati sa izrazom koji se razmatra. Pritom, promenljive koje figurišu u izrazu iz programa smatraju se konstantama (zato se ova vrsta unifikacije zove* jednosmerna unifikacija*). U našem primeru,* `!(!x && !y)` *može da se unifikuje sa podizrazom* `!(!a && !b)` *za unifikator* $\sigma = [\mathtt{x} \mapsto \mathtt{a}, \mathtt{y} \mapsto \mathtt{b}]$. *Važi* $(\mathtt{x} \ \mathtt{||} \ \mathtt{y})\sigma = \mathtt{a} \ \mathtt{||} \ \mathtt{b}$, *te se primenom zamene od izraza* `if (!(!a && !b) || c)` *dobija izraz* `if (a || b || c)`.

Ovim je, zapravo, opisan i mehanizam primene logičkih ekvivalencija u algoritmima PRENEX i KNF.

9.3.4 Metod rezolucije

Metod rezolucije (eng. *resolution method*) formulisao je Alen Robinson (Alan Robinson) 1965. godine, sledeći mnogobrojne prethodne rezultate. To je postupak za ispitivanje (ne)zadovoljivosti formule logike prvog reda u klauzalnoj formi, tj. za ispitivanje (ne)zadovoljivosti skupa klauza logike prvog reda. Metod se može pojednostaviti tako da je primenljiv za ispitivanje (ne)zadovoljivosti skupa klauza iskazne logike.

Formula koja je u konjunktivnoj normalnoj formi može da ima konjunkte koji se ponavljaju, a njeni konjunkti mogu da imaju literale koji se ponavljaju. Međutim, na osnovu asocijativnosti i komutativnosti konjunkcije i disjunkcije, kao i na osnovu logičkih ekvivalencija $\mathcal{A} \wedge \mathcal{A} \equiv \mathcal{A}$ i $\mathcal{A} \vee \mathcal{A} \equiv \mathcal{A}$, takva ponavljanja mogu da se eliminišu i formula koja je u konjunktivnoj normalnoj formi može da se zameni (logički ekvivalentnom) formulom koja je konjunkcija različitih klauza od kojih je svaka disjunkcija različitih literala. Dakle, formula se može opisati skupom klauza i, dalje, skupom skupova literala. Takva formula je zadovoljiva ako i samo ako postoji interpretacija u kojoj su sve njene klauze tačne. Klauza je zadovoljiva ako postoji interpretacija u kojoj je bar jedan literal iz te klauze tačan, pa se smatra da prazna klauza, u oznaci $\square$, nije zadovoljiva.

Može se osigurati da se logičke konstante $\top$ i $\bot$ ne pojavljuju u skupu klauza. Naime, klauza koja sadrži literal $\top$ je u svakoj valuaciji tačna, pa može biti eliminisana (jer ne utiče na zadovoljivost polaznog skupa klauza). Ako klauza C sadrži literal $\bot$, onda taj literal može biti obrisan, dajući novu klauzu C' (jer je u svakoj interpretaciji klauza C tačna ako i samo ako je tačna klauza C').

U slučaju iskazne logike, ako je literal l jednak iskaznom slovu p, onda sa $\bar{l}$ označavamo literal $\neg p$; ako je literal l jednak negaciji iskaznog slova p (tj. literalu $\neg p$), onda sa $\bar{l}$ označavamo literal p. Za literale l i $\bar{l}$ kažemo da su međusobno *komplementni*. U slučaju logike prvog reda, ako je literal l jednak $p(t_1, t_2, \ldots, t_n)$, onda sa $\bar{l}$ označavamo literal $\neg p(t_1, t_2, \ldots, t_n)$, a ako je literal l jednak $\neg p(t_1, t_2, \ldots, t_n)$, onda sa $\bar{l}$ označavamo literal $p(t_1, t_2, \ldots, t_n)$. Za literale l i $\bar{l}$ kažemo da su (međusobno) *komplementni*.

Metod rezolucije (i za iskaznu i za logiku prvog reda) proverava da li je dati skup klauza (ne)zadovoljiv. Ako je potrebno ispitati da li je formula Φ valjana, dovoljno je metodom rezolucije utvrditi da li je formula $\neg\Phi$ nezadovoljiva (ovaj vid dokazivanja da je formula Φ valjana zovemo *dokazivanje pobijanjem*), pri čemu je potrebno najpre formulu $\neg\Phi$ transformisati u skup klauza. Ako se izvede prazna klauza, onda to znači da je formula $\neg\Phi$ nezadovoljiva, pa je Φ valjana; ako u nekom koraku ne može da se izvede nijedna nova klauza, onda to znači da je formula $\neg\Phi$ zadovoljiva, pa Φ nije valjana. Za razliku od iskaznog slučaja, u logici prvog reda moguć je i ishod da nove klauze mogu da se izvode beskonačno, a da se pritom ne izvede prazna klauza.

Metod rezolucije za iskaznu logiku. U metodu rezolucije za iskaznu logiku primenjuje se pravilo rezolucije sledećeg oblika:

$$\frac{C' \vee l \quad C'' \vee \bar{l}}{C' \vee C''}$$

Klauzu $C' \vee C''$ zovemo *rezolventom* klauza $C' \vee l$ i $C'' \vee \bar{l}$, a klauze $C' \vee l$ i $C'' \vee \bar{l}$ *roditeljima rezolvente*. Kažemo da klauze $C' \vee l$ i $C'' \vee \bar{l}$ *rezolviramo* pravilom rezolucije.

Pravilo rezolucije može se zapisati i na sledeći, intuitivniji način:

$$\frac{\neg C' \Rightarrow l \quad l \Rightarrow C''}{\neg C' \Rightarrow C''}$$

Ilustrujmo taj oblik pravila jednostavnim primerom nad formulama iskazanim prirodnim jezikom:

$$\frac{\neg\text{„vedro je“} \Rightarrow \text{„pada kiša“} \quad \text{„pada kiša“} \Rightarrow \text{„meč se otkazuje“}}{\neg\text{„vedro je“} \Rightarrow \text{„meč se otkazuje“}}$$

U polaznoj formi pravila rezulucije, navedeno zaključivanje izgleda ovako:

$$\frac{\text{„vedro je“} \vee \text{„pada kiša“} \quad \neg\text{„pada kiša“} \vee \text{„meč se otkazuje“}}{\text{„vedro je“} \vee \text{„meč se otkazuje“}}$$

Metod rezolucije je postupak za ispitivanje zadovoljivosti skupa klauza koji se sastoji od uzastopnog primenjivanja pravila rezolucije (slika 9.3).

U primeni metoda rezolucije, niz klauza (polaznih i izvedenih) označavaćemo obično sa $C_1, C_2, C_3, \ldots$ Iza izvedene klauze zapisivaćemo oznake klauza iz kojih je ona izvedena, kao i redne brojeve literala nad kojim je primenjeno pravilo rezolucije. Literale u klauzama razdvajaćemo obično simbolom ',' (umesto simbolom '$\vee$').

Algoritam: Metod rezolucije

Ulaz: Skup klauza S

Izlaz: Odgovor *zadovoljiv/nezadovoljiv*

```
1: ponavljaj
2:     ako u tekućem skupu klauza postoji prazna klauza (□) onda
3:         vrati odgovor da je skup klauza S nezadovoljiv;
4:     ako se pravilom rezolucije može izvesti neka nova klauza onda
5:         dodaj odgovarajuću rezolventu u tekući skup klauza (i u skupu zadrži roditelje te rezolvente);
6:     inače
7:         vrati odgovor da je skup klauza S zadovoljiv.
```

Slika 9.3: Algoritam Metod rezolucije.

Primer 9.43. *Metodom rezolucije se iz skupa* $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}, \{\neg r\}\}$ *može izvesti prazna klauza:*

C_1 : $\neg p, \neg q, r$
C_2 : $\neg p, q$
C_3 : p
C_4 : $\neg r$

C_5 : $\neg p, r$ $(C_1, 2; C_2, 2)$
C_6 : $\neg p$ $(C_4, 1; C_5, 2)$
C_7 : $\square$ $(C_3, 1; C_6, 1)$

Skup klauza $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}, \{\neg r\}\}$ *je, dakle, nezadovoljiv.*

Primer 9.44. *Metodom rezolucije se iz skupa* $\{\{\neg p, \neg q, r\}, \{\neg p, q\}, \{p\}\}$ *ne može izvesti prazna klauza. Ovaj skup klauza je, dakle, zadovoljiv.*

U iskaznom slučaju, nad konačnim skupom promenljivih koje se pojavljuju u zadatom skupu klauza postoji konačno mnogo klauza, pa samim tim i mogućih rezolventi tokom primene metoda rezolucije. Zbog toga se metod rezolucije zaustavlja za svaku ulaznu formulu (tj. za svaki ulazni skup klauza). Može se dokazati i više – da je metod rezolucije *procedura odlučivanja* za zadovoljivost skupova klauza iskazne logike:

Teorema 9.7 (Teorema o algoritmu Metod rezolucije)**.** *Metod rezolucije zaustavlja se za svaku iskaznu formulu i u završnom skupu klauza postoji prazna klauza ako i samo ako je polazna formula nezadovoljiva.*

Metod rezolucije može se modifikovati na razne načine radi efikasnosti.

Metod rezolucije za logiku prvog reda. U iskaznoj logici pravilo rezolucije može da se primeni na sledeće načine:

$$\frac{\text{„Sokrat je filozof“} \Rightarrow \text{„Sokrat je čovek“} \quad \text{„Sokrat je čovek“} \Rightarrow \text{„Sokrat je smrtan“}}{\text{„Sokrat je filozof“} \Rightarrow \text{„Sokrat je smrtan“}}$$

$$\frac{\text{„Platon je filozof“} \Rightarrow \text{„Platon je čovek“} \quad \text{„Platon je čovek“} \Rightarrow \text{„Platon je smrtan“}}{\text{„Platon je filozof“} \Rightarrow \text{„Platon je smrtan“}}$$

U logici prvog reda, pravilo rezolucije (koje će tek biti uvedeno precizno) može da se primeni na sledeći način:

$$\frac{\text{„}x \text{ je filozof“} \Rightarrow \text{„}x \text{ je čovek“} \quad \text{„}x \text{ je čovek“} \Rightarrow \text{„}x \text{ je smrtan“}}{\text{„}x \text{ je filozof“} \Rightarrow \text{„}x \text{ je smrtan“}}$$

Ovakvo zaključivanje je opštije (pa u tom smislu i bolje) jer iz jednog zaključka „x je filozof“ $\Rightarrow$ „x je smrtan“ primenom supstitucija $[x \mapsto \text{Sokrat}]$ i $[x \mapsto \text{Platon}]$, možemo da dobijemo specifična tvrđenja:

$$\text{„Sokrat je filozof“} \Rightarrow \text{„Sokrat je smrtan“ ,}$$

$$\text{„Platon je filozof“} \Rightarrow \text{„Platon je smrtan“}$$

pa, primenom supstitucije $[x \mapsto \text{Cezar}]$, i

$$\text{„Cezar je filozof“} \Rightarrow \text{„Cezar je smrtan“}$$

(iako Cezar nije filozof) i mnoga druga slična. Ovakvo zaključivanje, koje uključuje promenljive logike prvog reda, ne može se opravdati u terminima iskazne logike i za njega je potrebno pravilo rezolucije za logiku prvog reda, motivisano prethodnim primerima. Njegov specijalni slučaj je:

$$\frac{\neg\Gamma' \Rightarrow \mathcal{A} \quad \mathcal{A} \Rightarrow \Gamma''}{\neg\Gamma' \Rightarrow \Gamma''}$$

U navedenom pravilu, veznik negacije koji postoji u formuli $\neg\Gamma' \Rightarrow \mathcal{A}$ tu je samo radi pogodnijeg zapisa pravila do kojeg ćemo doći kasnije. I formule bez te negacije mogu da se rezolviraju analogno.

Napravimo korak dalje. U navedenom pravilu specijalnog oblika, ista formula $\mathcal{A}$ pojavljuje se i u formuli $\neg\Gamma' \Rightarrow \mathcal{A}$ i u formuli $\mathcal{A} \Rightarrow \Gamma''$. Ona može da bude instanca formula $\mathcal{A}'$ i $\mathcal{A}''$ koje nisu jednake ali su unifikabilne, tj. formula $\mathcal{A}$ može da bude jednaka formulama $\mathcal{A}'\sigma$ i $\mathcal{A}''\sigma$. To ilustruje naredni primer.

Primer 9.45. *Formule* $\neg p(a,y) \Rightarrow q(a,y)$ *i* $q(x,b) \Rightarrow r(x,b)$ *mogu da se rezolviraju tek ako se na njih primeni supstitucija* σ *takva da važi* $q(a,y)\sigma = q(x,b)\sigma$*, na primer,* $\sigma = [x \mapsto a, y \mapsto b]$*. Formule* $\neg p(a,b) \Rightarrow q(a,b)$ *i* $q(a,b) \Rightarrow r(a,b)$ *se mogu rezolvirati i daju zaključak* $\neg p(a,b) \Rightarrow r(a,b)$*.*

Kao što je nagovešteno, i formule bez veznika negacije koji postoji u formuli $\neg\Gamma' \Rightarrow \mathcal{A}$ mogu da se rezolviraju analogno i to ilustruje naredni primer.[3]

Primer 9.46. *Pretpostavimo da su date formule*
$otac(Aleksa, y) \Rightarrow roditelj(Aleksa, y)$ *i*
$roditelj(x, Branko) \Rightarrow predak(x, Branko)$*.*
One mogu da se rezolviraju iako formule $roditelj(Aleksa, y)$ *i* $roditelj(x, Branko)$ *nisu jednake. Naime, one su unifikabilne i jedan unifikator je* $\sigma = [x \mapsto Aleksa, y \mapsto Branko]$*. Formule*
$otac(Aleksa, Branko) \Rightarrow roditelj(Aleksa, Branko)$ *i*
$roditelj(Aleksa, Branko) \Rightarrow predak(Aleksa, Branko)$
se mogu rezolvirati i daju zaključak
$otac(Aleksa, Branko) \Rightarrow predak(Aleksa, Branko)$*.*
Primetimo da se na osnovu datih formula ne može izvesti formula
$otac(Aleksa, y) \Rightarrow predak(Aleksa, y)$*.*

Pokazano je kako se rezolviraju formule $\neg\Gamma' \Rightarrow \mathcal{A}'$ i $\mathcal{A}'' \Rightarrow \Gamma''$, kada su formule $\mathcal{A}'$ i $\mathcal{A}''$ unifikabilne. Zapravo se rezolviraju formule $(\neg\Gamma' \Rightarrow \mathcal{A}')\sigma$ i $(\mathcal{A}'' \Rightarrow \Gamma'')\sigma$ tj. formule $\neg\Gamma'\sigma \Rightarrow \mathcal{A}'\sigma$ i $\mathcal{A}''\sigma \Rightarrow \Gamma''\sigma$, pri čemu je σ neki unifikator za formule $\mathcal{A}'$ i $\mathcal{A}''$, tj. važi $\mathcal{A}'\sigma = \mathcal{A}''\sigma$:

$$\frac{\neg\Gamma'\sigma \Rightarrow \mathcal{A}'\sigma \quad \mathcal{A}''\sigma \Rightarrow \Gamma''\sigma}{\neg\Gamma'\sigma \Rightarrow \Gamma''\sigma}$$

Navedeno pravilo može se zapisati i na sledeći način:

$$\frac{(\Gamma' \vee \mathcal{A}')\sigma \quad (\Gamma'' \vee \neg\mathcal{A}'')\sigma}{(\Gamma' \vee \Gamma'')\sigma}$$

ili, kao što je uobičajeno:

$$\frac{\Gamma' \vee \mathcal{A}' \quad \Gamma'' \vee \neg\mathcal{A}''}{(\Gamma' \vee \Gamma'')\sigma}$$

uz uslov da je σ unifikator za $\mathcal{A}'$ i $\mathcal{A}''$.

Primer 9.47. *Klauze*
$\neg otac(x,y) \vee roditelj(x,y)$ *i*
$\neg roditelj(x,y) \vee predak(x,y)$
rezolviraju se na sledeći način:

$$\frac{\neg otac(x,y) \vee roditelj(x,y) \quad \neg roditelj(x,y) \vee predak(x,y)}{\neg otac(x,y) \vee predak(x,y)}$$

Unifikator $\sigma = [x \mapsto Aleksa, y \mapsto Branko]$ *daje rezolventu* $\neg otac(Aleksa, Branko) \vee predak(Aleksa, Branko)$*. Ipak, mnogo opštiji zaključak je rezolventa* $\neg otac(x,y) \vee predak(x,y)$*, koja se dobija za unifikator* $\sigma = [\,]$*.*

Kao što pokazuje i prethodni primer, bolje je da su zaključci što opštiji, te se u pravilu zahteva da je σ *najopštiji* unifikator za $\mathcal{A}'$ i $\mathcal{A}''$. Zato *pravilo rezolucije* za logiku prvog reda u nešto opštijem, ali i dalje specijalnom obliku (tzv. binarna rezolucija) glasi ovako:

$$\frac{\Gamma' \vee \mathcal{A}' \quad \Gamma'' \vee \neg\mathcal{A}''}{(\Gamma' \vee \Gamma'')\sigma}$$

[3] Primetimo da se u ovom primeru *otac* koristi kao predikatski simbol, za razliku od primera 9.4 gde se koristi kao funkcijski simbol. I jedan i drugi pristup mogu imati smisla, u zavisnosti od problema koji se modeluje. Ukoliko se podrazumeva da za svaku osobu postoji jedinstven otac, onda je pogodno koristiti drugi pristup i on će dati kompaktnije formule. Ukoliko je, međutim, potrebno razmatrati i tvrđenja poput tvrđenja da za svaku osobu postoji otac, onda je upotrebljiv samo prvi način i tada to tvrđenje može da zapiše na sledeći način: $\forall x \exists y\, otac(y,x)$.

gde su Γ' i Γ'' klauze, a σ je najopštiji unifikator za $\mathcal{A}'$ i $\mathcal{A}''$. Opšte pravilo rezolucije omogućava rezolviranje više literala odjednom. Ono može biti opisano na sledeći način:

$$\frac{\Gamma' \vee \mathcal{A}'_1 \vee \mathcal{A}'_2 \vee \ldots \vee \mathcal{A}'_m \quad \Gamma'' \vee \neg\mathcal{A}''_1 \vee \neg\mathcal{A}''_2 \vee \ldots \vee \neg\mathcal{A}''_n}{(\Gamma' \vee \Gamma'')\sigma}$$

gde je σ najopštiji unifikator za formule (sve istovremeno) $\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m, \mathcal{A}''_1, \mathcal{A}''_2, \ldots, \mathcal{A}''_n$.

Primer 9.48. *Primenjeno nad klauzama $r(g(y)) \vee p(x) \vee p(f(y))$ i $q(x) \vee \neg p(f(g(a)))$, uz unifikator $[y \mapsto g(a), x \mapsto f(g(a))]$, opšte pravilo rezolucije daje rezolventu $r(g(g(a))) \vee q(f(g(a)))$.*

Obe klauze na koje se primenjuje pravilo rezolucije su (implicitno) univerzalno kvantifikovane. Stoga se svaka od njihovih varijabli može preimenovati (jer su formule $\forall x\mathcal{A}(x)$ i $\forall x'\mathcal{A}(x')$ logički ekvivalentne). Štaviše, to je neophodno uraditi za sve deljene varijable, jer bi, inače, neke primene pravila rezolucije bile (pogrešno) onemogućene (jer odgovarajući literali ne bi bili unifikabilni).

Primer 9.49. *Nad klauzama $\neg p(x, y) \vee \neg p(z, y) \vee p(x, z)$ i $\neg p(b, a)$ može se primeniti pravilo rezolucije, jer su literali $p(x, z)$ i $p(b, a)$ unifikabilni (uz najopštiji unifikator $\sigma = [x \mapsto b, z \mapsto a]$). Rezolventa ove dve klauze je klauza $\neg p(b, y) \vee \neg p(a, y)$.*

Ako se pravilo rezolucije primenjuje dalje, onda u dobijenoj klauzi sve promenljive treba da budu preimenovane (treba da dobiju imena koja do tada nisu korišćena): $\neg p(b, y') \vee \neg p(a, y')$.

Metod rezolucije za logiku prvog reda ima isti opšti oblik kao metod rezolucije za iskaznu logiku (slika 9.3), s tim što se koristi opšte pravilo rezolucije za logiku prvog reda.

Primer 9.50. *Dokažimo da je formula $p(a) \Rightarrow (\exists x)p(x)$ valjana. Negacija date formule je logički ekvivalentna formuli $p(a) \wedge (\forall x)\neg p(x)$. Metod rezolucije primenjuje se na skup klauza $\{p(a), \neg p(x)\}$. Pravilo rezolucije moguće je primeniti samo na jedan način — literali $p(a)$ i $\neg p(x)$ se unifikuju supstitucijom $[x \mapsto a]$ i njime se dobija prazna klauza. Odatle sledi da je formula $p(a) \Rightarrow (\exists x)p(x)$ valjana.*

Primer 9.51. *Dokažimo metodom rezolucije za logiku prvog reda da formula $(\forall x)(\exists y)p(x, y) \Rightarrow (\exists y)(\forall x)p(x, y)$ nije valjana. Negacija date formule je logički ekvivalentna sa formulom $(\forall x)(\exists y)(p(x, y) \wedge (\forall y)(\exists x)\ \neg p(x, y))$ i sa formulom $(\forall x)(\exists y)(\forall u)(\exists v)(p(x, y) \wedge \neg p(v, u))$. Od nje se skolemizacijom dobija skup od dve klauze: $\{p(x, f(x)), \neg p(g(x, u), u)\}$. Pravilo rezolucije nije moguće primeniti na ove dve klauze, odakle sledi da je formula $(\forall x)(\exists y)\ (p(x, y) \wedge (\forall y)(\exists x)\neg p(x, y))$ zadovoljiva, tj. polazna formula nije valjana.*

U primerima izvršavanja metoda rezolucije, niz klauza (polaznih i izvedenih) označavaćemo često sa C_i $(i = 1, 2, \ldots)$. Iza izvedene klauze zapisivaćemo oznake klauza iz kojih je ona izvedena, redne brojeve literala u tim klauzama, iskorišćeni najopštiji unifikator, kao i supstituciju pomoću koje se preimenuju promenljive.

Primer 9.52. *Dokažimo da je formula $(\forall x)(\exists y)q(x, y)$ logička posledica skupa formula $\{(\forall x)(\exists y)p(x, y), (\forall x)(\forall y)(p(x, y) \Rightarrow q(x, y))\}$.*

Dovoljno je dokazati da je formula

$$\mathcal{A} = ((\forall x)(\exists y)p(x, y) \wedge (\forall x)(\forall y)(p(x, y) \Rightarrow q(x, y))) \Rightarrow (\forall x)(\exists y)q(x, y)$$

valjana. Preneks normalna forma negacije ove formule je

$$(\exists w)(\forall x)(\exists y)(\forall u)(\forall v)(\forall z)(p(x, y) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(w, z)) \ .$$

Nakon skolemizacije, ova formula dobija oblik:

$$(\forall x)(\forall u)(\forall v)(\forall z)(p(x, g(x)) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(c, z)) \ ,$$

pri čemu je c nova Skolemova konstanta, a g nova Skolemova funkcija. Konjunktivna normalna forma formule

$$p(x, g(x)) \wedge (\neg p(u, v) \vee q(u, v)) \wedge \neg q(c, z)$$

je ista ta formula, pa početni skup čine sledeće klauze:

C_1 : $p(x, g(x))$ *(prvi deo hipoteze)*
C_2 : $\neg p(u, v), q(u, v)$ *(drugi deo hipoteze)*
C_3 : $\neg q(c, z)$ *(zaključak)*
Prazna klauza izvodi se na sledeći način:
C_4 : $q(x', g(x'))$ $(C_1, 1; C_2, 1)$, $[v \mapsto g(x), u \mapsto x]$; *preimenovanje:* $[x \mapsto x']$
C_5 : $\square$ $(C_3, 1; C_4, 1)$, $[x' \mapsto c, z \mapsto g(c)]$

Primer 9.53. *Neka je* $\Gamma = \{\ \forall x \forall y \forall z\ (predak(x, y) \wedge predak(y, z) \Rightarrow predak(x, z))\,,\ \forall x\ predak(otac(x), x)\ \}$ *i dokažimo:*
$\Gamma \models predak(otac(otac(Ana)), Ana)$. *Odgovarajuća klauzalna forma je*
$\forall x \forall y \forall z \forall u\ (\neg predak(x, y) \vee \neg predak(y, z) \vee predak(x, z)) \wedge\ predak(otac(u), u)\ \wedge \neg predak(otac(otac(Ana)), Ana)$, *a skup klauza:*

C_1 : $\neg predak(x, y) \vee \neg predak(y, z) \vee predak(x, z)$ *(prvi deo hipoteze)*
C_2 : $predak(otac(u), u)$ *(drugi deo hipoteze)*
C_3 : $\neg predak(otac(otac(Ana)), Ana)$ *(tvrđenje)*

Prazna klauza se izvodi na sledeći način:

C_4 : $\neg predak(otac(otac(Ana)), y') \vee \neg predak(y', Ana)$ $(C_1, 3; C_3, 1)$, $[x \mapsto otac(otac(Ana)), z \mapsto Ana]$; *preimenovanje* $[y \mapsto y']$
C_5 : $predak(otac(otac(Ana)), Ana)$ $(C_4, 1; C_2, 1)$, $[u \mapsto otac(Ana), y' \mapsto otac(Ana)]$;
C_6 : $\square$ $(C_5, 1; C_3, 1)$

Primer 9.54. *Dokažimo da je formula*

$$(\forall x)(\forall y)(\forall z)(above(y, x) \wedge below(z, x) \Rightarrow above(y, z))$$

logička posledica skupa formula
$\{\ (\forall x)(\forall y)(above(x, y) \Rightarrow \neg above(y, x))$,
$(\forall x)(\forall y)(above(x, y) \Leftrightarrow below(y, x))$,
$(\forall x)(\forall y)(\forall z)(above(x, y) \wedge above(y, z) \Rightarrow above(x, z))\ \}$
(videti primere 9.1 i 9.25). Dovoljno je dokazati da je formula
$(\forall x)(\forall y)(above(x, y) \Rightarrow \neg above(y, x))) \wedge$
$(\forall x)(\forall y)(above(x, y) \Leftrightarrow below(y, x))) \wedge$
$(\forall x)(\forall y)(\forall z)(above(x, y) \wedge above(y, z) \Rightarrow above(x, z)))$
$\Rightarrow$
$(\forall x)(\forall y)(\forall z)(above(y, x) \wedge below(z, x) \Rightarrow above(y, z))$
valjana. Odgovarajući skup klauza je:

C_1 : $\neg above(x_1, y_1) \vee \neg above(y_1, x_1)$ *(prvi deo hipoteze)*
C_2 : $\neg above(x_2, y_2) \vee below(y_2, x_2)$ *(drugi deo hipoteze)*
C_3 : $\neg below(x_3, y_3) \vee above(y_3, x_3)$ *(drugi deo hipoteze)*
C_4 : $\neg above(x_4, y_4) \vee \neg above(y_4, z_4) \vee above(x_4, z_4)$ *(treći deo hipoteze)*
C_5 : $above(c_y, c_x)$ *(prvi deo zaključka)*
C_6 : $below(c_z, c_x)$ *(drugi deo zaključka)*
C_7 : $\neg above(c_y, c_z)$ *(treći deo zaključka)*

Prazna klauza se izvodi na sledeći način:
C_8 : $\neg above(c_y, y_5) \vee \neg above(y_5, c_z)$ $(C_7, 1; C_4, 3)$, $[x_4 \mapsto c_y, z_4 \mapsto c_z]$; *preimenovanje:* $[y_4 \mapsto y_5]$
C_9 : $\neg above(c_x, c_z)$ $(C_8, 1; C_5, 1)$, $[y_5 \mapsto c_x]$;
C_{10} : $\neg below(c_z, c_x)$ $(C_3, 2; C_9, 1)$, $[y_3 \mapsto c_x, x_3 \mapsto c_z]$;
C_{11} : $\square$ $(C_6, 1; C_{10}, 1)$, $[\,]$

Primer 9.55. *Formula* $\forall x \forall y\ (p(x, y) \Rightarrow p(y, x))$ *je logička posledica formula* $\forall x\ p(x, x)$ *i* $\forall u \forall v \forall w\ (p(u, v) \wedge$

$p(w,v) \Rightarrow p(u,w))$, *jer je formula*

$$\mathcal{A} = (\forall x\ p(x,x)) \wedge (\forall u \forall v \forall w\ (p(u,v) \wedge p(w,v) \Rightarrow p(u,w))) \Rightarrow$$

$$(\forall x \forall y\ (p(x,y) \Rightarrow p(y,x)))$$

valjana:

C_1 :	$p(x,x)$	
C_2 :	$\neg p(u,v), \neg p(w,v), p(u,w)$	
C_3 :	$p(a,b)$	
C_4 :	$\neg p(b,a)$	
C_5 :	$\neg p(u',b), p(u',a)$	$(C_2,2;C_3,1)$ $[w \mapsto a, v \mapsto b]$; *preimenovanje:* $[u \mapsto u']$
C_6 :	$\neg p(b,b)$	$(C_4,1;C_5,2)$ $[u' \mapsto b]$
C_7 :	$\square$	$(C_1,1;C_6,1)$ $[x \mapsto b]$

Kao što je ranije rečeno, da bismo dokazali da važi $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n \models \mathcal{B}$, dovoljno je dokazati da je formula $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \ldots \wedge \mathcal{A}_n \Rightarrow \mathcal{B}$ valjana, tj. da je formula $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \ldots \wedge \mathcal{A}_n \wedge \neg\mathcal{B}$ nezadovoljiva. Ako formule $\mathcal{A}_1$, $\mathcal{A}_2, \ldots, \mathcal{A}_n$, $\mathcal{B}$ nemaju slobodne promenljive, onda ne moramo da u klauzalnu formu transformišemo navedenu konjunkciju, dovoljno je da u klauzalne forme transformišemo formule $\mathcal{A}_1$, $\mathcal{A}_2, \ldots, \mathcal{A}_n$, $\neg\mathcal{B}$ pojedinačno. Time će se dobiti isti skup klauza, ali na jednostavniji način, primenljiv i na prethodne primere.

Teorema 9.8 (Saglasnost i potpunost metoda rezolucije)**.** *Metod rezolucije je saglasan: ako je primenom metoda dobijena prazna klauza, onda je i polazni skup klauza nezadovoljiv (ili, drugim rečima, iz zadovoljivog skupa klauza može se dobiti samo zadovoljiv skup klauza).*

Metod rezolucije je potpun za pobijanje: iz svakog nezadovoljivog skupa klauza moguće je izvesti praznu klauzu.

Navedena teorema odnosi se na opšte pravilo rezolucije. Navedena svojstva važe i za binarno pravilo ako se pored njega koristi i dodatno pravilo — *faktorizacija* (ili *grupisanje*) koje grupiše i unifikuje unifikabilne literale u okviru jedne klauze.

Problem ispitivanja valjanosti u logici prvog reda nije odlučiv, ali metod rezolucije pruža najviše što se može dobiti: njime se iz svakog nezadovoljivog skupa klauza može izvesti prazna klauza (čime je dokazano da je nezadovoljiv), ali ne može se za svaki zadovoljiv skup klauza dokazati da je zadovoljiv (naime, moguće je izvesti beskonačno mnogo rezolventi). Dualno, pobijanjem se metodom rezolucije može za svaku valjanu formulu dokazati da je valjana, ali ne može se za svaku formulu koja nije valjana utvrditi da nije valjana.

Primetimo da u opisu metoda rezolucije nije zadat način na koji se biraju klauze nad kojim se primenjuje pravilo rezolucije. Takođe, teorema 9.8 tvrdi da se iz svakog nezadovoljivog skupa klauza *može* izvesti prazna klauza, a ne tvrdi da se iz svakog nezadovoljivog skupa klauza *mora* izvesti prazna klauza bez obzira na izbor klauza koje će biti rezolvirane. Naime, u zavisnosti od izbora klauza na koje se primenjuje pravilo rezolucije moguće je da se i za nezadovoljiv skup klauza metod rezolucije ne zaustavlja. Dakle, primena metoda rezolucije može se smatrati i problemom pretrage. Način na koji se biraju klauze na koje se primenjuje pravilo rezolucije čini *strategiju za navođenje* metoda rezolucije.

Jedna od mogućnosti za obezbeđivanje potpunosti metoda rezolucije u strožijem smislu (da postoji strategija za navođenje metoda rezolucije takva da se iz svakog nezadovoljivog skupa klauza nužno izvodi prazna klauza u konačno mnogo koraka) je sistematsko izvođenje svih rezolventi iz skupa klauza koji se širi tokom primene metoda. *Sistematski metod rezolucije* može se definisati na sledeći način: metod se primenjuje u iteracijama; prvu iteraciju čini kreiranje početnog skupa klauza; neka pre i-te iteracije tekući skup klauza čine klauze $C_1, C_2, \ldots, C_n$, i-ta iteracija sastoji se od izvođenja (i dodavanja tekućem skupu klauza) svih mogućih rezolventi iz po svake dve klauze iz skupa C_1, $C_2, \ldots, C_n$ (broj tih klauza je konačan); metod se zaustavlja ako se u nekom koraku izvede prazna klauza ili ako se u nekoj iteraciji ne može izvesti nijedna nova klauza. Ako je razmatrani skup klauza Γ nezadovoljiv, onda se, na osnovu teoreme 9.8, iz njega metodom rezolucije može izvesti prazna klauza, tj. postoji niz rezolventi R_1, $R_2, \ldots, R_n$ (koje se izvode iz početnih i izvedenih klauza) od kojih je poslednja u nizu prazna klauza. Ako se na skup klauza Γ primeni sistematski metod rezolucije, u nekoj iteraciji biće (ako već pre toga nije izvedena prazna klauza) izvedene sve klauze iz skupa R_1, $R_2, \ldots, R_n$, pa i prazna klauza. Iz navedenog razmatranja sledi naredna teorema.

Teorema 9.9 (Potpunost sistematskog metoda rezolucije)**.** *Ako je Γ nezadovoljiv skup klauza, onda se iz njega sistematskim metodom rezolucije mora izvesti prazna klauza.*

Očigledno je da je sistematski metod rezolucije izuzetno neefikasan. Postoji više strategija koje obezbeđuju nužno izvođenje prazne klauze iz nezadovoljivog skupa klauza (tj. sprečavaju beskonačne petlje), ali na efikasniji način. Smanjivanje izvođenja nepotrebnih klauza jedan je od najvažnijih ciljeva u dizajniranju strategija za metod rezolucije.

9.3.5 Prirodna dedukcija

Pojam valjanosti je semantičke prirode, a koncept dokazivanja i sistema za dedukciju vodi do pojma teoreme koji je sintaksičko-deduktivne prirode. Pojam *teoreme* je deduktivni pandan semantičkog pojma *valjane formule*. Između ova dva pojma postoji veza i deduktivni sistemi obično imaju svojstvo potpunosti i saglasnosti: ako je neka formula valjana, onda ona može biti dokazana u okviru deduktivnog sistema, a ako za neku formulu postoji dokaz u okviru deduktivnog sistema, onda je ona sigurno valjana.

Sistemi za dedukciju su čisto sintaksičke prirode — primenjuju se kroz kombinovanje simbola, ne razmatrajući semantiku formula. Sisteme za dedukciju zovemo i *računima* – (*iskazni račun* u slučaju iskazne logike i *predikatski račun* u slučaju logike prvog reda). Postoji više različitih deduktivnih sistema. U nastavku će biti opisan samo jedan — *prirodna dedukcija* (eng. *natural deduction*).

Sistem prirodne dedukcije (račun prirodne dedukcije) uveo je, 1935. godine, Gerhard Gencen (Gerhard Gentzen) sa namerom da prirodnije opiše uobičajeno zaključivanje matematičara.

U prirodnoj dedukciji koriste se logički veznici $\neg$, $\wedge$, $\vee$, $\Rightarrow$, kao i logička konstanta $\bot$. Formula $\mathcal{A} \Leftrightarrow \mathcal{B}$ je kraći zapis za $(\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{B} \Rightarrow \mathcal{A})$, a formula $\top$ kraći zapis za $\mathcal{A} \Rightarrow \mathcal{A}$. Skup formula definiše se na uobičajeni način.

Pravila izvođenja sistema prirodne dedukcije data su na slici 9.4. Primetimo da za svaki logički veznik i svaki kvantifikator postoje pravila koja ga uvode (pravila I-tipa) i pravila koja ga eliminišu (pravila E-tipa). Pravilo efq (*Ex falso quodlibet*) je jedino pravilo koje ne uvodi niti eliminiše neki logički veznik. Skup pravila sistema prirodne dedukcije za iskaznu logiku čine sva pravila sa slike 9.4 izuzev onih koja uključuju kvantifikatore.

$$\frac{\begin{array}{c}[\mathcal{A}]^u\\ \vdots\\ \bot\end{array}}{\neg\mathcal{A}}\ \neg I, u \qquad \frac{\mathcal{A} \quad \neg\mathcal{A}}{\bot}\ \neg E$$

$$\frac{\mathcal{A} \quad \mathcal{B}}{\mathcal{A}\wedge\mathcal{B}}\ \wedge I \qquad \frac{\mathcal{A}\wedge\mathcal{B}}{\mathcal{A}}\ \wedge E \qquad \frac{\mathcal{A}\wedge\mathcal{B}}{\mathcal{B}}\ \wedge E$$

$$\frac{\mathcal{A}}{\mathcal{A}\vee\mathcal{B}}\ \vee I \qquad \frac{\mathcal{B}}{\mathcal{A}\vee\mathcal{B}}\ \vee I \qquad \frac{\mathcal{A}\vee\mathcal{B} \quad \begin{array}{c}[\mathcal{A}]^u\\ \vdots\\ C\end{array} \quad \begin{array}{c}[\mathcal{B}]^v\\ \vdots\\ C\end{array}}{C}\ \vee E, u, v$$

$$\frac{\begin{array}{c}[\mathcal{A}]^u\\ \vdots\\ \mathcal{B}\end{array}}{\mathcal{A}\Rightarrow\mathcal{B}}\ \Rightarrow I, u \qquad \frac{\mathcal{A} \quad \mathcal{A}\Rightarrow\mathcal{B}}{\mathcal{B}}\ \Rightarrow E$$

$$\frac{\mathcal{A}[x\mapsto y]}{(\forall x)\mathcal{A}}\ \forall I \qquad \frac{(\forall x)\mathcal{A}}{\mathcal{A}[x\mapsto t]}\ \forall E$$

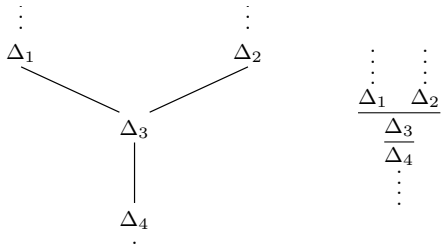
uz dodatni uslov

$$\frac{\mathcal{A}[x\mapsto t]}{(\exists x)\mathcal{A}}\ \exists I \qquad \frac{(\exists x)\mathcal{A} \quad \begin{array}{c}[\mathcal{A}[x\mapsto y]]^u\\ \vdots\\ \mathcal{B}\end{array}}{\mathcal{B}}\ \exists E, u$$

uz dodatni uslov

$$\frac{\bot}{D}\ efq$$

Slika 9.4: Pravila izvođenja sistema prirodne dedukcije.



Slika 9.5: Deo dokaza i njegov pojednostavljeni prikaz.

U pravilima izvođenja prikazanim na slici 9.4, simbol t označava proizvoljan term. Dodatni uslov za pravilo $\forall I$ je da važi da je $x = y$ ili da promenljiva y nije slobodna u $\mathcal{A}$, kao i da važi da y nije slobodna ni u jednoj neoslobođenoj pretpostavci u izvođenju formule $\mathcal{A}[x \mapsto y]$. Dodatni uslov za pravilo $\exists E$ je da važi da je $x = y$ ili da promenljiva y nije slobodna u $\mathcal{A}$, kao i da važi da y nije slobodna u $\mathcal{B}$ niti u bilo kojoj neoslobođenoj pretpostavci u izvođenju formule $\mathcal{B}$ osim, eventualno, u formuli $\mathcal{A}[x \mapsto y]$.

Postoji sistem prirodne dedukcije za klasičnu logiku, sistem NK, i sistem prirodne dedukcije za intuicionističku logiku, sistem NJ. U sistemu NK postoji jedna aksiomska shema – $\mathcal{A} \vee \neg\mathcal{A}$ (*tertium non datur*), a sistem NJ nema aksioma.

Tokom izvođenja dokaza u sistemu prirodne dedukcije mogu se koristiti (nedokazane) pretpostavke, ali one moraju biti eliminisane („oslobođene“) pre kraja izvođenja. U zapisu pravila, $[\mathcal{F}]$ označava da se nekoliko (možda i nula) pojavljivanja pretpostavke $\mathcal{F}$ oslobađa (kao nedokazane, neraspoložive pretpostavke) neposredno nakon primene pravila. Pritom, ta primena pravila može ostaviti i nekoliko neoslobođenih pojavljivanja pretpostavke $\mathcal{F}$. Pretpostavkama su pridružene oznake (obično prirodni brojevi), koje se zapisuju i u okviru zapisa primenjenog pravila (kako bi se znalo koja pretpostavka je oslodođena u kojem koraku).

U sistemu prirodne dedukcije, *dokaz* je stablo čijem je svakom čvoru pridružena formula, a svakom listu ili pretpostavka ili aksioma. Formula $\mathcal{A}$ je *teorema* prirodne dedukcije ako postoji dokaz u čijem je korenu $\mathcal{A}$ i koji nema neoslobođenih pretpostavki i tada pišemo $\vdash \mathcal{A}$ i kažemo da je formula $\mathcal{A}$ *dokaziva* u sistemu prirodne dedukcije. Ako postoji dokaz u čijem je korenu formula $\mathcal{A}$ i koji ima neoslobođene pretpostavke koje pripadaju nekom skupu Γ, onda kažemo da je formula $\mathcal{A}$ *deduktivna posledica* skupa Γ i tada pišemo $\Gamma \vdash \mathcal{A}$. Elemente skupa Γ tada zovemo i *premisama* ili *hipotezama* dokaza. Ako je skup Γ jednak $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$, onda pišemo i $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n \vdash \mathcal{A}$.

Dokaz u sistemu prirodne dedukcije obično se prikazuje u vidu stabla čiji su listovi na vrhu, a koren na dnu. To stablo se prikazuje pojednostavljeno, stilizovano (videti sliku 9.5).

U narednim primerima prikazani su dokazi nekoliko teorema. To su teoreme i klasične i intuicionističke logike jer se u dokazima ne koriste aksiome *tertium non datur*, tj. $\mathcal{A} \vee \neg\mathcal{A}$. Ako se neka formula može dokazati samo uz korišćenje takvih aksioma, onda je ona teorema klasične, ali ne i intuicionističke logike.

Primer 9.56. *Formula* $(\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})$ *je teorema sistema prirodne dedukcije, tj. važi* $\vdash (\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})$*:*

$$\dfrac{\dfrac{[\mathcal{A} \vee \mathcal{B}]^1 \quad \dfrac{[\mathcal{A}]^2}{\mathcal{B} \vee \mathcal{A}} \vee I \quad \dfrac{[\mathcal{B}]^3}{\mathcal{B} \vee \mathcal{A}} \vee I}{\mathcal{B} \vee \mathcal{A}} \vee E, 2, 3}{(\mathcal{A} \vee \mathcal{B}) \Rightarrow (\mathcal{B} \vee \mathcal{A})} \Rightarrow I, 1$$

Primer 9.57. *U sistemu prirodne dedukcije važi:* $\mathcal{A} \Rightarrow \mathcal{B}, \mathcal{B} \Rightarrow \mathcal{C} \vdash \mathcal{A} \Rightarrow \mathcal{C}$*:*

$$\dfrac{\dfrac{[\mathcal{A}]^1 \quad \mathcal{A} \Rightarrow \mathcal{B}}{\mathcal{B}} \Rightarrow E \quad \mathcal{B} \Rightarrow \mathcal{C}}{\dfrac{\mathcal{C}}{\mathcal{A} \Rightarrow \mathcal{C}} \Rightarrow I, 1} \Rightarrow E$$

Primer 9.58. *U sistemu prirodne dedukcije važi* $\vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$*:*

$$\dfrac{\dfrac{\dfrac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{B}} \vee I \quad \dfrac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{C}} \vee I}{(\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \wedge I}{\mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \Rightarrow I, 1$$

U prethodnom dokazu, primenom pravila $\Rightarrow I$ *nisu morala da budu oslobođena sva pojavljivanja pretpostavke* $\mathcal{A}$*. Na primer:*

$$\dfrac{\dfrac{\dfrac{[\mathcal{A}]^1}{\mathcal{A} \vee \mathcal{B}} \vee I \quad \dfrac{\mathcal{A}}{\mathcal{A} \vee \mathcal{C}} \vee I}{(\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \wedge I}{\mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})} \Rightarrow I, 1$$

Ovaj dokaz je dokaz tvrđenja $\mathcal{A} \vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$ *(što je slabije tvrđenje od tvrđenja* $\vdash \mathcal{A} \Rightarrow (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$*).*

Primer 9.59. *U sistemu prirodne dedukcije važi* $\forall x \mathcal{A}, \forall x(\mathcal{A} \Rightarrow \mathcal{B}) \vdash \forall x \mathcal{B}$*:*

$$\dfrac{\dfrac{\dfrac{\forall x \mathcal{A}}{\mathcal{A}} \forall E \quad \dfrac{\forall x(\mathcal{A} \Rightarrow \mathcal{B})}{\mathcal{A} \Rightarrow \mathcal{B}} \forall E}{\mathcal{B}} \Rightarrow E}{\forall x \mathcal{B}} \forall I$$

Primer 9.60. *Formula* $\neg(\exists x)p(x) \Rightarrow (\forall y)\neg p(y)$ *je teorema sistema prirodne dedukcije:*

$$\dfrac{\dfrac{\dfrac{\dfrac{[p(z)]^1}{(\exists x)p(x)} \exists I \quad [\neg(\exists x)p(x)]^2}{\bot} \neg E}{\neg p(z)} \neg I, 1}{(\forall y)\neg p(y)} \forall I}{\neg(\exists x)p(x) \Rightarrow (\forall y)\neg p(y)} \Rightarrow I, 2$$

Primer 9.61. *Formula* $(\exists x)(\forall y)p(x, y) \Rightarrow (\forall y)(\exists x)p(x, y)$ *je teorema sistema prirodne dedukcije (i za klasičnu i za intuicionističku logiku). Neki matematičar bi ovu formulu (neformalno) dokazao na sledeći način:*

1. *Pretpostavimo da važi* $(\exists x)(\forall y)p(x, y)$*.*
2. *Pretpostavimo da važi* $(\forall y)p(x', y)$ *za neko* x'*.*
3. *Neka je* y' *proizvoljni objekat. Tada važi* $p(x', y')$*.*
4. *Iz* $p(x', y')$ *sledi da važi* $(\exists x)p(x, y')$*.*
5. *Objekat* y' *je proizvoljan, pa važi* $(\forall y)(\exists x)p(x, y)$*.*
6. *Iz* $(\exists x)(\forall y)p(x, y)$ *i iz toga što pretpostavka* $(\forall y)p(x', y)$ *ima za posledicu* $(\forall y)(\exists x)p(x, y)$ *(a* $(\forall y)p(x, y)$ *ne zavisi od* x'*), sledi formula* $(\forall y)(\exists x)p(x, y)$*.*
7. *Kako je dokazano da iz formule* $(\exists x)(\forall y)p(x, y)$ *sledi* $(\forall y)(\exists x)p(x, y)$*, time je dokazana formula* $(\exists x)(\forall y)p(x, y) \Rightarrow (\forall y)(\exists x)p(x, y)$*.*

Ovaj dokaz može se precizno opisati u vidu dokaza u sistemu prirodne dedukcije:

$$\dfrac{\dfrac{[(\forall y)p(x',y)]^1}{p(x',y')}\,\forall E}{\dfrac{(\exists x)p(x,y')}{(\forall y)(\exists x)p(x,y)}\,\exists I}\,\forall I$$
$$\dfrac{[(\exists x)(\forall y)p(x,y)]^2 \qquad (\forall y)(\exists x)p(x,y)}{(\forall y)(\exists x)p(x,y)}\,\exists E,1$$
$$\dfrac{}{(\exists x)(\forall y)p(x,y)\Rightarrow(\forall y)(\exists x)p(x,y)}\,\Rightarrow I,2$$

Naredna teorema povezuje semantička i deduktivna svojstva klasične logike (ona važi i za iskaznu i za predikatsku logiku).

Teorema 9.10. *Formula je teorema sistema prirodne dedukcije za klasičnu logiku ako i samo ako je valjana.*

9.4 Rešavanje problema svođenjem na problem valjanosti

U ovom poglavlju biće prikazano nekoliko primera rešavanja problema svođenjem na problem valjanosti, na načine koji su nagovešteni u poglavlju 9.3.1.

9.4.1 Porodični odnosi

U bazama podataka državne uprave postoje informacije o svim građanima koji uključuju informacije o oba roditelja. Državnoj upravi je ponekad potrebno da proveri da li su neke dve osobe u nekom porodičnom odnosu, ali bilo bi iracionalno da se svi mogući rodbinski odnosi čuvaju eksplicitno u bazama podataka. Umesto toga, moguće je proveriti neke konkretne porodične odnose i neka opšta tvrđenja korišćenjem logičkog rasuđivanja.

Porodični odnosi (kao što su „biti otac“, „biti majka“, „biti brat“, „biti tetka“ i slično) mogu se opisati formulama logike prvog reda. Na primer,

$\mathcal{A} = \forall x\forall y(brat(x,y) \Leftrightarrow musko(x) \wedge x \neq y \wedge \exists z(roditelj(z,x) \wedge roditelj(z,y)))$

$\mathcal{B} = \forall x\forall y(brat(x,y) \wedge musko(y) \Leftrightarrow brat(y,x) \wedge musko(x))$

Navedene formule mogu se interpretirati na različite načine. Ali ako se interpretiraju nad nekim konkretnim skupom osoba a predikatski simboli se interpretiraju na prirodan način (na primer, relacijski simbol *roditelj* preslikava se u uobičajenu relaciju „biti roditelj“ nad skupom osoba), njihovo značenje se poklapa sa intuitivnim i može se razmatrati njihova valjanost na osnovu znanja o porodičnim odnosima. Na primer, formula $\mathcal{B}$ valjana je za svaki skup osoba i prirodno interpretirane predikatske simbole. To se može utvrditi nekakvim „rešavačem za porodične odnose“ koje većina ljudi ima u svojim glavama iako obično ne u nekom eksplicitnom obliku. O valjanosti formule $\mathcal{B}$ i sličnih možemo da rasuđujemo korišćenjem takvog rešavača (ako ga imamo) ili korišćenjem „univerzalnog rešavača“, tj. dokazivača za logiku prvog reda, kao što je opisano u nastavku.

Formula $\mathcal{B}$ valjana je za svaki skup osoba i prirodno interpretirane predikatske simbole, ali nije valjana (nije valjana u svakoj $\mathcal{L}$ strukturi). Ona je, međutim, logička posledica formule $\mathcal{A}$. Dokazivač za logiku prvog reda ćemo, dakle, moći da upotrebimo da proverimo da li je formula $\mathcal{B}$ logička posledica formule $\mathcal{A}$, tj. da proverimo da li je važi $\mathcal{A} \models \mathcal{B}$, tj. da proverimo da li je formula $\mathcal{A} \Rightarrow \mathcal{B}$ valjana. Na opisani način proveravamo tvrđenja oblika $\Gamma \models \mathcal{B}$. Ovim pristupom razmatra se „opšta“ valjanost ali skup Γ ima ulogu da efektivno ograniči skup interpretacija koje se razmatraju: ovakvim razmatranjem pokrivene su samo interpretacije u kojima su formule skupa Γ valjane. Formule Γ imaju ulogu svojevrsnih aksioma i matematičke teorije se grade u istom duhu. Da bi važilo $\Gamma \models \mathcal{B}$ za širok skup formula $\mathcal{B}$, u skup Γ dodaćemo više jednostavnih svojstava porodičnih odnosa, kao što su:

$\forall x\forall y\ (sestra(x,y) \Leftrightarrow zensko(x) \wedge x \neq y \wedge \exists z(roditelj(z,x) \wedge roditelj(z,y)))$

$\forall x\forall y\ (majka(x,y) \Leftrightarrow zensko(x) \wedge roditelj(x,y))$

$\forall x\forall y\ (otac(x,y) \Leftrightarrow musko(x) \wedge roditelj(x,y))$

$\forall x\forall y\ (tetka(x,y) \Leftrightarrow \exists z(sestra(x,z) \wedge roditelj(z,y)))$

Ako u skup Γ dodamo, na primer, i formule:

$musko(MarkoJankovic)$

$musko(LazarJankovic)$

$otac(JovanJankovic, MarkoJankovic)$

$otac(JovanJankovic, LazarJankovic)$

onda može da se pokaže da važi i $\Gamma \models brat(MarkoJankovic, LazarJankovic)$. Dakle, opisanim mehanizmom može se rasuđivati o porodičnim odnosima među konkretnim osobama.

9.4.2 Verifikacija softvera

U poglavlju 8.4.5 bilo je reči o verifikaciji softvera i razmatrano je da li je narednu funkciju `f` napisanu na programskom jeziku C:

```
int f(int y)
{
   int x;
   x = x ^ y;
   y = x ^ y;
   x = x ^ y;
   return x * x;
}
```

moguće zameniti narednom funkcijom `g` (to je uvek tako, bez obzira na to što promenljiva `x` nije inicijalizovana u funkciji `f` i njena inicijalna vrednost mogla bi da bude bilo šta):

```
int g(int n)
{
   return n * n;
}
```

Pokazano je da u ovom konkretnom slučaju može da se razmatra dejstvo bitovskih operatora nad pojedinačnim bitovima i sredstvima iskazne logike pokazano je da je svaki bit programske promenljive `x` pre naredbe `return x * x;` u funkciji `f` jednak odgovarajućem bitu programske promenljive `n` pre naredbe `return n * n;` u funkciji `g`, čime je dokazana ekvivalentnost dve funkcije.

Sredstvima logike prvog reda to tvrđenje može se dokazati na drugačiji način, primenljiv u širem skupu situacija. Dokaz će važiti za bilo koju širinu tipa `int`. Kao u iskaznom slučaju, pogodno je u razmatranje uvesti i promenljive za međurezultate:

`x1 = x ^ y;`
`y1 = x1 ^ y;`
`x2 = x1 ^ y1;`

Da bi se dokazalo da su funkcije `f` i `g` ekvivalentne, dovoljno je dokazati da je formula

$\forall x\ \forall x1\ \forall x2\ \forall y\ \forall y1 \quad x1 = (x \hat{}\ y) \ \wedge\ y1 = (x1 \hat{}\ y) \ \wedge\ x2 = (x1 \hat{}\ y1) \ \Rightarrow\ x2 = y$

valjana u strukturi koju čine moguće vrednosti tipa `int`. Ukoliko nemamo rešavač specijalizovan za tu strukturu, moramo da se okrenemo opštijem pristupu, tj. dokazivaču za logiku prvog reda. Ali tada moramo da u igru uvedemo uslove Γ koji opisuju strukturu vrednosti tipa `int` sa operatorom `^` i koji imaju ulogu aksioma. Može se dokazati (što nećemo ovde uraditi) da za tu strukturu i operator `^` važe sledeća tvrđenja koje ćemo koristiti kao aksiome:

$\forall u\ \forall v \quad (u \hat{}\ v) = (v \hat{}\ u)$
$\forall u\ \forall v \quad ((u \hat{}\ v) \hat{}\ v) = u$

Primenom metoda rezolucije može se pokazati da je navedena formula logička posledica navedenih aksioma i aksioma jednakosti (videti poglavlje 9.4.5).

9.4.3 Rezanje ploča

Nameštaj se često pravi od pločastih materijala i tada je od velike važnosti raspoređivanje komponenti nameštaja po pojedinačnim pločama. Što manje upotrebljenih ploča – veća isplativost. Razmotrimo jednostavnu instancu ovog problema: da li je moguće od ploče dimenzija 100×60 napraviti komade dimenzija 80×30 i 50×40? Pretpostavimo, jednostavnosti radi, da duže stranice moraju da budu paralelne dužim stranicama osnovne ploče (to, zbog teksture pločastog materijala, može da bude realan zahtev). Pretpostavimo i da je rezove moguće vršiti samo na celobrojnim rastojanjima. Dokazaćemo da traženo rezanje nije moguće.

Smestimo sve elemente u koordinatni sistem: neka je donji levi ugao ploče u tački $(0, 0)$ i neka su (x_1, y_1) i (x_2, y_2) donji levi uglovi dva željena pločasta dela. Da bi delovi bili na ploči mora da važi formula $\mathcal{A}$:

$0 \leq x_1 \ \wedge\ x_1 + 80 \leq 100 \ \wedge$
$0 \leq y_1 \ \wedge\ y_1 + 30 \leq 60 \ \wedge$
$0 \leq x_2 \ \wedge\ x_2 + 50 \leq 100 \ \wedge$
$0 \leq y_2 \ \wedge\ y_2 + 40 \leq 60$

Dva elementa nemaju presek ako i samo ako postoji horizontalna ili vertikalna prava koja ih razdvaja, tj. ako važi formula $\mathcal{B}$:

$x_1 + 80 \leq x_2 \ \vee \ x_2 + 50 \leq x_1 \ \vee \ y_2 + 40 \leq y_1 \ \vee \ y_1 + 30 \leq y_2$

Da bismo dokazali da traženo rezanje nije moguće, dovoljno je da dokažemo da važi $\mathcal{A} \models \neg\mathcal{B}$, tj. da dokažemo da je univerzalno zatvorenje formule $\mathcal{A} \Rightarrow \neg\mathcal{B}$ valjana formula. Ta formula, međutim, nije valjana. No, ona je valjana u strukturi celih brojeva sa prirodno interpretiranim simbolima + i <, a to je jedina interpretacija koja nas zanima. Valjanost u toj strukturi može da dokaže SMT rešavač za linearnu aritmetiku nad celim brojevima. Valjanost univerzalnog zatvorenja formule $\mathcal{A} \Rightarrow \neg\mathcal{B}$ SMT rešavač dokazuje tako što dokazuje da je formula $\mathcal{A} \wedge \mathcal{B}$ nezadovoljiva (poglavlje 9.4.5).

9.4.4 PROLOG

Jezik PROLOG najznačajniji je predstavnik paradigme logičkog programiranja, u kojoj se koristi logika kao deklarativni jezik za opisivanje problema, a dokazivač teorema kao mehanizam za rešavanje. Ime PROLOG dolazi od engleskih reči „PROgramming in LOGic“. Jezik PROLOG i prvi interpretator za njega razvijeni su na Univerzitetu u Marseju 1972. godine. Vreme najveće popularnosti PROLOG-a je prošlo, ali se on i dalje široko koristi, uglavnom za probleme iz oblasti veštačke inteligencije: od medicinskih sistema pa do istraživanja podataka. Pogodan je za brz razvoj prototipova jer se obrada ulaza i izlaza, parsiranje i druge slične operacije implementiraju jednostavno, a mehanizam za netrivijalno rasuđivanje je jednostavno raspoloživ. U nastavku će biti ukratko opisana samo neka svojstva jezika PROLOG.

Mehanizam izvođenja zaključaka u PROLOG-u zasniva se na metodu rezolucije i na korišćenju Hornovih klauza — klauza u kojima postoji najviše jedan literal koji nije pod negacijom. Za ispitivanje zadovoljivosti skupova klauza, zahvaljujući njihovoj specifičnoj formi, koristi se algoritam koji je polinomske složenosti. Četiri vrste Hornovih klauza i odgovarajuće formule logike prvog reda prikazani su u narednoj tabeli (formule $\mathcal{A}_i$ su atomičke). Može se dokazati da svaki nezadovoljiv skup Hornovih klauza mora da sadrži bar jednu činjenicu i bar jednu ciljnu klauzu.

Vrsta	logika prvog reda	PROLOG
implikaciona klauza	$\neg\mathcal{A}_1 \vee \ldots \vee \neg\mathcal{A}_n \vee \mathcal{A}$	$\mathcal{A} :- \mathcal{A}_1, \ldots, \mathcal{A}_n.$
ciljna klauza	$\neg\mathcal{A}_1 \vee \ldots \vee \neg\mathcal{A}_n$	$?- \mathcal{A}_1, \ldots, \mathcal{A}_n.$
činjenica	$\mathcal{A}$	$\mathcal{A}.$
prazna klauza	$\square$	`false`

PROLOG sistemi obično sadrže interaktivni interpretator a neki sistemi omogućavaju i kompiliranje kôda. Komunikacija sa PROLOG interpretatorom odvija se kroz komandni prozor, a prompt interpretatora obično izgleda ovako: `?-`. PROLOG konvencija je da se konstante zapisuju malim početnim slovom, a promenljive velikim početnim slovom.

Primer 9.62. *Pretpostavimo da je zadata činjenica*

```
man(sokrat).
```

(nova činjenica može se učitati iz datoteke, kao deo programa a može se zadati i interaktivno, na sledeći način: `?- assert(man(sokrat)).`*) Nakon ovoga, upit*

```
?- man(sokrat).
```

uspeva, tj. daje rezultat `Yes`*. Da bi zadati upit bio zadovoljen, činjenica* `man(sokrat)` *je bila rezolvirana sa klauzom dobijenom iz upita (to je klauza* ¬ `man(sokrat)`*) i to je dalo praznu klauzu, kao što je i trebalo. Time je, praktično, dokazano da je* `man(sokrat)` ⇒ `man(sokrat)` *valjana formula. Pretpostavimo da je zadato i sledeće pravilo (na primer, sa* `?- assert(mortal(X) :- man(X)).`*):*

```
mortal(X) :- man(X).
```

U ovom pravilu, predikat `mortal(X)` *je glava pravila a (jednočlani) niz predikata* `man(X)` *je rep pravila. Upit:*

```
?- mortal(sokrat).
```

uspeva (daje odgovor `Yes`*). Da bi ovaj upit bio zadovoljen, klauza* ¬`man(X)` ∨ `mortal(X)` *(dobijena iz zadatog pravila) rezolvira se sa klauzom dobijenom iz upita* ¬`mortal(sokrat)` *i daje rezolventu, tj. novi cilj* ¬`man(sokrat)`*. On uspeva, jer sa klauzom* `man(sokrat)` *(dobijenom iz zadate činjenice) daje praznu klauzu. Time je, praktično, dokazano da je*

`(man(sokrat)` ∧ ∀`x (man(x)` ⇒ `mortal(x)))` ⇒ `mortal(sokrat)`

valjana formula.

Ako se zada upit:

```
? - mortal(X).
```

onda se metodom rezolucije pokušava dokazivanje nezadovoljivosti skupa klauza:

`man(sokrat)`
$\neg$`man(X)` $\vee$ `mortal(X)`
$\neg$`mortal(Y)`

Primetimo da je u trećoj klauzi promenljiva preimenovana u `Y`*, da ne bi došlo do poklapanja imena promenljive u dve klauze. Ciljna (treća) klauza može da se rezolvira sa drugom klauzom, korišćenjem unifikatora* [`Y` $\mapsto$ `X`] *dajući novi cilj*

$\neg$`man(X)`

i nakon preimenovanja promenljive `X`*:*

$\neg$`man(X')`

Rezolviranjem ove klauze sa prvom klauzom iz početnog skupa, korišćenjem unifikatora [`X'` $\mapsto$ `sokrat`] *dobija se prazna klauza, pa je dokazana nezadovoljivost datog skupa klauza i* PROLOG *vraća rezultat:*

```
Yes
```

i daje odgovor:

```
X = sokrat
```

To je jedino moguće rešenje i ako ukucamo simbol `;` *(čime tražimo druge moguće unifikatore) dobićemo odgovor* `No`*. Naravno, upiti*

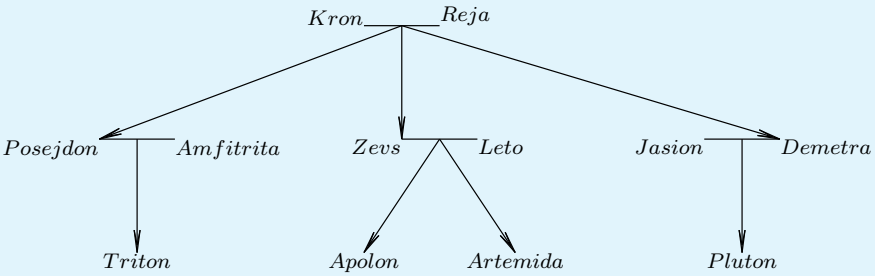
```
?- man(platon).
```

i

```
? - mortal(platon).
```

ne uspevaju i daju odgovor `No` *(sem ako nije zadata i činjenica* `man(platon)`*).*

Primer 9.63. *Definisanje odnosa u* PROLOG*-u može se ilustrovati na primeru porodičnih odnosa (kao što su otac, majka, brat, tetka i slično) (videti poglavlje 9.4.1) i to nad skupom starogrčkih božanstava, čije su veze prikazane na narednoj slici (u vidu porodičnog stabla):*



```
musko(kron).
musko(posejdon).
musko(zevs).
musko(jasion).
musko(triton).
musko(apolon).
musko(pluton).

zensko(reja).
zensko(amfitrita).
zensko(leto).
zensko(demetra).
zensko(artemida).

roditelj(kron,posejdon).
roditelj(reja,posejdon).
roditelj(kron,zevs).
roditelj(reja,zevs).
roditelj(kron,demetra).
roditelj(reja,demetra).
roditelj(posejdon,triton).
roditelj(amfitrita,triton).
roditelj(zevs,apolon).
roditelj(leto,apolon).
roditelj(zevs,artemida).
roditelj(leto,artemida).
roditelj(jasion,pluton).
roditelj(demetra,pluton).

predak(X,Y) :- roditelj(X,Y).
predak(X,Y) :- roditelj(X,Z), predak(Z,Y).
```

```
majka(X,Y) :- zensko(X), roditelj(X,Y).
otac(X,Y) :- musko(X), roditelj(X,Y).
brat(X,Y) :- musko(X), roditelj(Z,X),
             roditelj(Z,Y), X\==Y.
sestra(X,Y) :- zensko(X), roditelj(Z,X),
               roditelj(Z,Y), X\==Y.
tetka(X,Y) :- sestra(X,Z), roditelj(Z,Y).
stric(X,Y) :- brat(X,Z), otac(Z,Y).
ujak(X,Y) :- brat(X,Z), majka(Z,Y).
bratodstrica(X,Y) :- musko(X), otac(Z,X), stric(Z,Y).
sestraodstrica(X,Y) :- zensko(X), otac(Z,X), stric(Z,Y).
bratodujaka(X,Y) :- musko(X), otac(Z,X), ujak(Z,Y).
sestraodujaka(X,Y) :- zensko(X), otac(Z,X), ujak(Z,Y).
bratodtetke(X,Y) :- musko(X), majka(Z,X), tetka(Z,Y).
sestraodtetke(X,Y) :- zensko(X), majka(Z,X), tetka(Z,Y).
```

U relacijama `brat` *i* `sestra`*, predikat* `X\==Y` *ima vrednost* tačno *ako je* `X` *različito od* `Y`*. U suprotnom, ima vrednost* netačno*. U nastavku je dato nekoliko primera upita i rezultata koje sistem daje.*

```
?- stric(posejdon,apolon).

Yes
```

```
?- ujak(X,Y).

X = zevs,
Y = pluton
X = zevs,
Y = pluton
X = posejdon,
Y = pluton
X = posejdon,
Y = pluton
```

```
?- sestraodstrica(X,Y).

X=artemida,
Y=triton
X=artemida,
Y=triton
```

U navedenim primerima, mogu se primetiti ponavljanja istih rešenja. U slučaju upita `ujak(X,Y)`*, razlog za to je što zadovoljavanje ovog cilja zavisi od zadovoljavanja podcilja* `brat(X,Z)`*, koji zavisi od zadovoljavanja podciljeva* `roditelj(W,X)` *i* `roditelj(W,Z)`*. Ako važi* `X=zevs`*,* `Y=pluton` *i* `Z=demetra`*, onda postoje dve mogućnosti za* `W`*, što su* `kron` *i* `reja`*. Kako sistem dva puta nalazi način da zadovolji sve potciljeve u kojima važi* `X=zevs` *i* `Y=pluton`*, dva puta navodi tu kombinaciju kao rešenje. Slučaj upita* `sestraodstrica(X,Y)` *je analogan.*

Primetimo da pravilu

```
otac(X,Y) :- musko(X), roditelj(X,Y).
```

odgovara formula $\forall x \forall y\ (musko(x) \wedge roditelj(x,y) \Rightarrow otac(x,y))$*, koja je slabija od formule (iz poglavlja 9.4.1):*

$\forall x \forall y\ (otac(x,y) \Leftrightarrow musko(x) \wedge roditelj(x,y))$

PROLOG *iz navedenih činjenica može da pokaže da važi* `otac(kron,zevs)`*. No, da nije data činjenica* `roditelj(kron,zevs)`*,* PROLOG *ne bi mogao da je izvede čak i da ima na raspolaganju činjenicu* `otac(kron,zevs)`*. Naime, navedeni* PROLOG *program dizajniran je tako da iz informacija o roditeljstvu izvodi sve druge porodične odnose. Ukoliko je potrebno i obratno, onda u program treba dodati dodatna pravila.*

Da bi PROLOG mogao da bude upotrebljiv kao programski jezik, u njegovoj semantici postoje odstupanja od semantike logike prvog reda (na primer, umesto negacije koja je kao u logici prvog reda, u PROLOG-u se koristi takozvana „negacija kao neuspeh“).

9.4.5 FOL dokazivači i SMT rešavači i njihovi ulazni formati

Programe koji rešavaju instance problema valjanosti ili zadovoljivosti u logici prvog reda zovemo obično *dokazivači za logiku prvog reda* ili FOL *dokazivači* (eng. FOL-*provers*, od *first-order logic*). Većina savremenih FOL dokazivača zasnovana je na metodi rezolucije, obogaćenoj mnogim dodatnim tehnikama i heuristikama. Neki od danas popularnih FOL dokazivača su Vampire, E i Spass. FOL dokazivači obično očekuju ulaz u nekom od TPTP[4] formata, kao što je FOF format. U ovom formatu, formule se navode jedna po jedna sa oznakama o tome da li se radi o aksiomi ili o tvrđenju koje dokazivač treba da dokaže. U slučaju primera 9.54, problem može da se zapiše na sledeći način.

```
fof(a1, axiom, (![X,Y] : (above(X,Y) => ~above(Y,X)))).
fof(a2, axiom, (![X,Y] : (above(X,Y) <=> below(Y,X)))).
fof(a3, axiom, (![X,Y,Z] : ((above(X,Y) & above(Y,Z)) =>
                  above(X,Z)))).
fof(cn, conjecture, (![X,Y,Z] : ((above(Y,X) & below(Z,X)) =>
                      above(Y,Z)))).
```

[4] TPTP (Thousands of Problems for Theorem Provers) je biblioteka problema za automatske dokazivače teorema. U okviru nje definisano je i nekoliko formata za zapis formula logike prvog reda.

Za navedeni opis problema (u vidu datoteke `above.tptp`), na primer, dokazivač Vampire vratiće odgovor da je odgovarajući skup klauza nezadovoljiv, tj. da je navedena formula logička posledica navedenih aksioma:

```
% Refutation found.
% SZS status Theorem for above
% SZS output start Proof for above
2. ! [X0,X1] : (above(X0,X1) <=> below(X1,X0)) [input]
3. ! [X0,X1,X2] : ((above(X1,X2) & above(X0,X1)) => above(X0,X2)) [input]
4. ! [X0,X1,X2] : ((below(X2,X0) & above(X1,X0)) => above(X1,X2)) [input]
5. ~! [X0,X1,X2] : ((below(X2,X0) & above(X1,X0)) => above(X1,X2))
   [negated conjecture 4]
7. ! [X0,X1,X2] : (above(X0,X2) | (~above(X1,X2) | ~above(X0,X1)))
   [ennf transformation 3]
8. ! [X0,X1,X2] : (above(X0,X2) | ~above(X1,X2) | ~above(X0,X1))
   [flattening 7]
9. ? [X0,X1,X2] : (~above(X1,X2) & (below(X2,X0) & above(X1,X0)))
   [ennf transformation 5]
10. ? [X0,X1,X2] : (~above(X1,X2) & below(X2,X0) & above(X1,X0))
   [flattening 9]
11. ! [X0,X1] : ((above(X0,X1) | ~below(X1,X0)) & (below(X1,X0) |
   ~above(X0,X1))) [nnf transformation 2]
12. ? [X0,X1,X2] : (~above(X1,X2) & below(X2,X0) & above(X1,X0)) =>
   (~above(sK1,sK2) & below(sK2,sK0) & above(sK1,sK0)) [choice axiom]
13. ~above(sK1,sK2) & below(sK2,sK0) & above(sK1,sK0) [skolemisation 10,12]
16. ~below(X1,X0) | above(X0,X1) [cnf transformation 11]
17. ~above(X1,X2) | above(X0,X2) | ~above(X0,X1) [cnf transformation 8]
18. above(sK1,sK0) [cnf transformation 13]
19. below(sK2,sK0) [cnf transformation 13]
20. ~above(sK1,sK2) [cnf transformation 13]
23. above(sK0,sK2) [resolution 16,19]
28. ~above(X1,sK0) | above(X1,sK2) [resolution 17,23]
29. above(sK1,sK2) [resolution 28,18]
30. $false [subsumption resolution 29,20]
% SZS output end Proof for above
% ------------------------------
% Version: Vampire 4.2.2 (commit e1949dd on 2017-12-14 18:39:21 +0000)
% Termination reason: Refutation

% Memory used [KB]: 4733
% Time elapsed: 0.103 s
```

Tvrđenje iz poglavlja 9.4.2 može se opisati na sledeći način:

```
fof(a1, axiom, (![U,V] : ( xor(xor(U,V),V)=U ))).
fof(a1, axiom, (![U,V] : ( xor(U,V)=xor(V,U)))).
fof(cn, conjecture, (![X,X1,X2,Y,Y1] :
          ((X1=xor(X,Y) & Y1=xor(X1,Y) & X2=xor(X1,Y1)) =>
             X2=Y))).
```

Dokazivač Vampire vratiće odgovor da je odgovarajući skup klauza nezadovoljiv, tj. da je navedena formula logička posledica navedenih aksioma i aksioma jednakosti (nije potrebno eksplicitno zadavati aksiome jednakosti).

Za mnoge praktične probleme, umesto FOL dokazivača (tj. „univerzalnog rešavača“), pogodno je koristiti specijalizovane rešavače za neke konkretne teorije ili konkretne strukture.

Na primer, za dokazivanje nezadovoljivosti formule iz primera sa rezanjem ploča (poglavlje 9.4.3) pogodno je koristiti specijalizovani SMT rešavač za linearnu aritmetiku nad celim brojevima i to za fragment bez kvantifikatora (jer su sve promenljive implicitno egzistencijalno kvantifikovane) – ta teorija obično se označava sa QF_LIA. Potrebno je dokazati da je formula $\mathcal{A} \wedge \mathcal{B}$ nezadovoljiva i taj zadatak može se opisati u vidu datoteke `ploce.smt` sa narednim sadržajem u formatu `smt-lib`:

```
(set-logic QF_LIA)
(declare-const x1 Int)
(declare-const y1 Int)
(declare-const x2 Int)
(declare-const y2 Int)

(assert (<= 0 x1))
```

```
(assert (<= 0 (+ x1 80) 100))
(assert (<= 0 y1))
(assert (<= 0 (+ y1 30) 60))
(assert (<= 0 x2))
(assert (<= 0 (+ x2 50) 100))
(assert (<= 0 y2))
(assert (<= 0 (+ y2 40) 60))
(assert (or (<= (+ x1 80) x2) (<= (+ x2 50) x1)
            (<= (+ y1 30) y2) (<= (+ y2 40) y1)))
(check-sat)
(get-model)
```

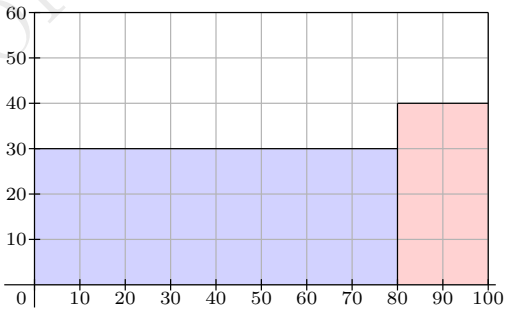
Za navedeni opis problema (u vidu datoteke `ploce.smt`), na primer, rešavač `z3` vratiće odgovor `unsat` (što znači da je polazni skup formula nezadovoljiv u strukturi celih brojeva):

```
unsat
```

Ukoliko dimenzije drugog komada nisu 50×40 nego, na primer, 20×40, onda za odgovarajuću `smt` datoteku (samo je vrednost 50 zamenjena sa 20), rešavač `z3` daje odgovor koji sadrži i jedno rešenje kako treba iseći tražene komade ploče:

```
sat
(model
  (define-fun x2 () Int
    80)
  (define-fun y1 () Int
    0)
  (define-fun x1 () Int
    0)
  (define-fun y2 () Int
    0)
)
```

Ovo rešenje, ovaj model zadate formule govori da postoji (barem jedno) željeno rezanje i u njemu je donji levi ugao prvog komada u tački $(x_1, y_1) = (0, 0)$, a donji levi ugao drugog komada u tački $(x_2, y_2) = (80, 0)$, kao što je ilustrovano na narednoj slici:



Deo III

Mašinsko učenje

Glava 10

Rešavanje problema korišćenjem mašinskog učenja

Rešavanje problema korišćenjem automatskog rasuđivanja počiva na formalnom i obično potpunom opisu problema koji je potrebno rešiti. Na primer, da bismo rešili problem n dama, potrebno je precizno definisati sva pravila koja se tiču raspoređivanja figura i njihovog napadanja. Ako je potrebno donositi zaključke o geometrijskim problemima, potrebno je potpuno precizno zadati aksiome geometrije i tvrđenje koje je potrebno dokazati. Međutim, u velikom broju praktičnih problema nije moguće potpuno precizno opisati pravila sveta u kojem se rešava problem, pa ni svojstva rešenja. Na primer, u slučaju prepoznavanja lica nije lako precizno definisati šta je lice. Prirodan pokušaj da se lice definiše u terminima njegovih delova dalje povlači pitanje definicija tih delova, kao i odnosa među njima. Ipak, određena pravilnost nesporno postoji. Sličan je i problem prevođenja rečenica sa jednog jezika na drugi. Iako je poznato da jezik ima jaku strukturu, ona je previše komplikovana i prepuna izuzetaka koji su dovoljni da učine formalno preciziranje pravila previše teškim. Samim tim još je veći problem opisati kako se struktura jednog jezika preslikava u strukturu drugog jezika na koji je potrebno izvršiti prevođenje.

Da bi neki problem bio rešen potrebno je da postoji njegova specifikacija. U prethodnim primerima teško je dati eksplicitnu i preciznu specifikaciju. Specifikacija može biti i implicitna, data velikom količinom primera onoga što jeste rešenje i onoga što nije rešenje problema, odnosno kroz konkretne podatke. Ovakva specifikacija nužno je slabija od formalne, ali nekada je najbolje što možemo uraditi u praksi. Mašinsko učenje bavi se upravo rešavanjem problema za koje je teško dati formalnu specifikaciju, ali je dostupna velika količina podataka koji na neki način ilustruju rešenja pojedinačnih instanci problema. Metode mašinskog učenja u pruženim podacima uočavaju relevantne zakonitosti i formulišu ih u vidu nekog matematičkog modela. Ključni cilj metoda mašinskog učenja je dobra generalizacija, tj. dobro ponašanje dobijenog matematičkog modela u situacijama koje nisu bile opisane podacima na osnovu kojih je model dobijen. Drugim rečima, ključni cilj je dobijanje modela koji će biti koristan za donošenje zaključaka o budućim instancama problema, tj. u njihovom rešavanju.

Greške u modelovanju podataka su nepoželjne ali moguće. Greške modela mogu da postoje u odnosu na podatke iz kojih je model dobijen ili u odnosu na podatke u kasnijoj primeni. Važnije su ove druge, jer je ključni cilj mašinskog učenja, kao što je rečeno, kreiranje modela koji će biti korisni u rešavanju budućih instanci problema.

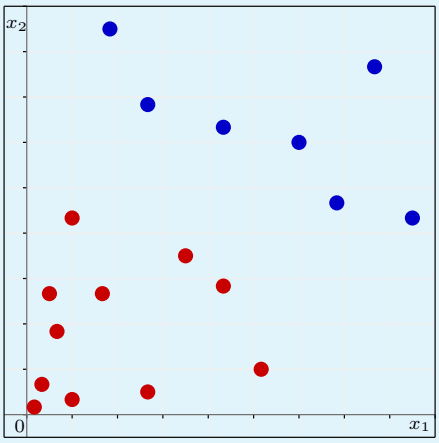
Raspoloživi podaci nekada mogu sadržati greške, pa i protivrečnosti. Takođe, mogu davati nepotpunu sliku problema koji se rešava. Otud je jasno da logika nije pogodan formalni okvir za mašinsko učenje, te metode mašinskog učenja nisu deduktivne, već induktivne metode zaključivanja, obično zasnovane na verovatnoći i statistici. Verovatnosni okvir pruža im robusnost u odnosu na greške u podacima, a često omogućava i kvantifikaciju pouzdanosti ponuđenog rešenja. Algoritmi učenja i algoritmi koji budu naučeni obično nemaju svojstvo potpunosti: nema garancija da će biti naučen traženi algoritam, niti da će algoritam koji je naučen da uspešno rešava svaku instancu problema. Uprkos ovome i imanentnom postojanju grešaka, mašinsko učenje u mnogim poljima daje sjajne rezultate koji daleko nadmašuju druge pristupe.

Metode mašinskog učenja tipično vrše odlučivanje veoma brzo (znatno brže nego, na primer, metode automatskog rasuđivanja koje svoju egzaktnost plaćaju visokom računskom zahtevnošću prilikom odlučivanja). Otud je njihova primena preferirana i u nekim slučajevima kada jeste moguće u potpunosti precizirati problem, ali je prostor pretrage ogroman. Takav je primer igranja igre go u kojoj je metodama mašinskog učenja pobeđen ljudski svetski šampion. Ipak, u rešenjima zasnovanim na mašinskom učenju, greške se uvek mogu očekivati te taj pristup nije primeren domenima u kojima greška nije dozvoljena (na primer, u automatskoj kontroli metroa).

U nastavku teksta, osnovni koncepti mašinskog učenja biće ilustrovani kroz naredni primer.

Primer 10.1. *Pretpostavimo da je potrebno napraviti specijalizovani pretraživač interneta koji omogućuje korisnicima da pretražuju samo računarske članke. Pored mnoštva elemenata koje ovaj sistem mora da ima, poput onih vezanih za komunikaciju na internetu, skladištenje informacija, interfejs prema korisniku i tako dalje, potrebno je da ima i specijalizovan modul koji se bavi razvrstavanjem, odnosno* klasifikacijom *članaka na članke iz oblasti računarstva i na članke iz svih ostalih oblasti. Razmotrimo kako bi elementarna varijanta ovakvog modula mogla biti osmišljena.*

Apstraktni aspekti teksta, poput semantike, stila i sličnih, iako vrlo relevantni za navedeni problem, ne dopuštaju laku analizu od strane mašine. Otud bi se elementarna varijanta sistema za razvrstavanje članaka morala zasnivati na neposredno dostupnim aspektima teksta. To su pre svega reči od kojih se tekst sastoji. Na sreću, reči mogu biti vrlo indikativne po pitanju oblasti kojoj tekst pripada, pošto mogu predstavljati stručne termine. Otud je zamislivo da se pomenuti problem može u zadovoljavajućoj meri rešiti razmatranjem frekvencija pojavljivanja pojedinih, pogodno odabranih reči (frekvencija neke reči u članku se računa tako što se broj pojavljivanja te reči podeli ukupnim brojem pojavljivanja svih reči u članku). U pojednostavljenom pristupu, moguće je osloniti se na frekvenciju stručnih reči „računar“ i „datoteka“, čime se svaki tekst predstavlja kao tačka u dvodimenzionoj ravni: x_1 koordinata predstavlja frekvenciju reči „računar“, a koordinata x_2 predstavlja frekvenciju reči „datoteka“. Očekuje se da će računarski tekstovi u tom slučaju biti predstavljeni tačkama koje su daleko od koordinatnog početka, a da će ostali tekstovi biti blizu koordinatnog početka, kao što je prikazano na sledećoj slici. Plavi krugovi predstavljaju računarske članke, a crveni ostale (prikazani podaci su izmišljeni, a za stvarne podatke bi bilo očekivano da veliki broj tačaka bude na osama koordinatnog sistema).



U idealnoj situaciji, te dve grupe tačaka su razdvojene i između njih postoji prava koja ih razdvaja. U tom slučaju, kasnije je moguće jednostavno odgovoriti na pitanje da li je članak računarski samo na osnovu toga sa koje strane te prave se nalazi tačka koja odgovara tom članku. Ukoliko ne postoji prava koja razdvaja dve grupe tačaka, potrebno je naći neku pravu koja većinu tačaka razdvaja ispravno. Važno pitanje je kako se i u takvom slučaju može odrediti prava koja najbolje razdvaja dve grupe članaka. Takvih algoritama ima više, a intuitivno, osnovna ideja je da se krene od proizvoljne prave i da se ona postepeno pomera dok se ne pozicionira najbolje moguće (po nekom preciznom kriterijumu) između dve grupe tačaka, tj. između dve klase. Detalji jednog takvog algoritma biće dati kasnije.

Primeri primena mašinskog učenja. Mašinsko učenje uspešno se primenjuje u mnoštvu praktičnih problema. Jedan od najstarijih, a još uvek zanimljivih praktičnih rezultata postignut je od strane sistema ALVINN zasnovanog na neuronskoj mreži, krajem osamdesetih godina dvadesetog veka, koji je naučen da bez ljudske pomoći vozi automobil auto-putem u prisustvu drugih vozila i brzinom od oko 110km/h. Sistem je uspešno vozio na putu dužine oko 140km. Sa razvojem dubokih neuronskih mreža, sredinom prve decenije ovog veka, projekat razvoja autonomnih vozila dobio je novi zamah. Mnoge kompanije trenutno razvijaju vozila koja treba da budu u stanju da samostalno učestvuju u gradskoj vožnji, koja je značajno komplikovanija od vožnje na auto-putu. Izazovi za tehnike mašinskog učenja u ovom problemu uključuju kako prepoznavanje puta i učesnika u saobraćaju, tako i donošenje odluka. Slične metode koriste se i za učenje upravljanja kvadrotorima (malim letilicama sa četiri propelera) u cilju prenošenja predmeta. Kompanija Amazon razmatra mogućnost ovakvog načina dostavljanja svojih pošiljki u gradskim sredinama.

Jedan od najpoznatijih ranih primera primene mašinskog učenja je i sistem TD-Gammon za igranje igre *Backgammon* konstruisan devedesetih godina prošlog veka. Igrajući protiv sebe više od milion partija i nasta-

vljajući da uči u igri sa ljudskim igračima, sistem je dostigao nivo igre u rangu svetskog šampiona. Na sličnim principima, ali koristeći modernije algoritme učenja konstruisan je sistem AlphaGo koji je 2015. i 2016. ubedljivo pobedio evropskog, a zatim i svetskog šampiona u igri go. Ova igra poznata je kao jedan od, do sada, najozbiljnijih izazova veštačkoj inteligenciji u domenu igranja igara, pošto po broju mogućih stanja daleko prevazilazi i šah, što drastično otežava primenu tradicionalnih tehnika veštačke inteligencije poput algoritma minimaks sa alfa-beta odsecanjem.

Kompanije poput Amazona, koje se bave internet prodajom, odavno koriste sisteme koji na osnovu primera kupovnih transakcija korisnika uče kako da budućim korisnicima preporučuju proizvode koji bi ih mogli interesovati. Ovakvi sistemi i odgovarajući algoritmi učenja nazivaju se sistemima za preporučivanje (eng. *recommender systems*).

Sistemi za prepoznavanje govora takođe koriste mašinsko učenje u nekoj formi. Sistem Sphinx, takođe sa kraja osamdesetih, bio je u stanju da prepozna izgovorene reči uz prilagođavanje izgovoru različitih ljudi, različitim karakteristikama mikrofona, pozadinskoj buci i slično. U međuvremenu su takvi sistemi značajno napredovali, ušli u široku upotrebu na mobilnim telefonima, a u stanju su i da vode dijaloge, daju preporuke relevantne korisniku i slično.

Neke od najvećih uspeha mašinsko učenje postiglo je u obradi slika i video zapisa. Tipični problemi su prepoznavanje i lociranje objekata na slikama i video zapisima, praćenje objekata u video zapisima, automatsko opisivanje slika prirodnim jezikom i slično. Impresivni uspesi postignuti su u generisanju realističnih slika na osnovu tekstualne specifikacije. Moderni sistemi ovog tipa u stanju su da generišu i slike kakve nikada nisu videli i da pritom zadovolje zahteve specifikacije po pitanju sadržaja, umetničkog stila, prirode osvetljenja, itd.

Verovatno najveći uspesi mašinskog učenja postignuti su obradi teksta na prirodnom jeziku. Rani uspesi postignuti su u prepoznavanju sentimenta koji tekst izražava (na primer, zadovoljstvo ili nezadovoljstvo), prepoznavanju vrsta reči i ekstrakciji informacija iz teksta koje odgovaraju nekom upitu. Sa razvojem dubokog učenja postalo je moguće trenirati sisteme koji su u stanju da generišu tekst – na primer ostatak teksta započetog pisma. Daljim razvojem ovakvi sistemi (na primer, *ChatGPT*) osposobljeni su za vođenje smislenih razgovora sa čovekom, generisanje programskog koda, parsiranje podataka iz teksta, stilsko uobličavanje teksta i razne druge poslove. Ovakvi sistemi mnoge poslove ne izvode savršeno. Na primer, mogu davati netačne informacije i često greše u rešavanju matematičkim problemima, a neretko traže nezanemarljivu veštinu korisnika u rešavanju drugih složenijih problema. Ipak, ovi sistemi se već koriste u poslovnoj praksi i brzo se dalje unapređuju. Ovi modeli, kao i modeli koji generišu video zapise ili slike (često na osnovu tekstualnog ulaza), nazivaju se *generativnim modelima*.

Sveprisutnost društvenih mreža dala je veliki impuls razvoju metoda mašinskog učenja nad grafovima. Društvena mreža može se smatrati grafom čiji čvorovi predstavljaju učesnike mreže, a grane postoje između učesnika koji su povezani u mreži (poput prijateljstva na mreži *Facebook*). Metode mašinskog učenja se u ovom kontekstu koriste za predviđanje budućih veza među učesnicima, recimo prilikom preporučivanja učesnicima mreže sa kime se mogu povezati. Razvijene su i metode za otkrivanje postojećih, ali neopaženih veza u društvenim mrežama (koje ne moraju biti samo mreže na internetu, već i u realnom životu). Jedna od motivacija za razvoj ovih metoda je otkrivanje povezanosti u terorističkim i kriminalnim grupama.

Veliki proboj u bioinformatici napravljen je metodama mašinskog učenja kada je rešen problem savijanja proteina. Proteini su u središtu većine bioloških procesa, pa je razumevanje njihove funkcije u živim organizmima od velikog značaja i može voditi razvoju novih lekova i novih metoda lečenja raznih bolesti. Funkcija proteina tesno je povezana sa njegovim sastavom – nizom aminokiselina i njegovom trodimenzionalnom strukturom, koja se može opisati koordinatama pomenutih aminokiselina. Problem savijanja proteina sastoji se u određivanju trodimenzionalne strukture na osnovu sekvence amino kiselina. Utvrđivanje sâme sekvence nije težak problem, ali je rešavanje problema određivanja načina savijanja na osnovu te sekvence trajalo oko pedeset godina, do 2020. godine, kada je sistem AlphaFold dao odlične rezultate na referentnom skupu instanci.

Faze rešavanja problema mašinskim učenjem. Tôk rešavanja problema pomoću mašinskog učenja obično obuhvata sledeće osnovne korake:

- Modelovanje problema;
- Rešavanje problema, odnosno treniranje modela;
- Evaluacija dobijenog rešenja, odnosno modela;
- Eksploatacija modela

Kod pretrage i automatskog rasuđivanja, treći korak može biti važan, ali je nekada i trivijalan ili nepostojeći. Mašinsko učenje uključuje evaluaciju dobijenog rešenja kao suštinski važan korak zbog pomenute mogućnosti

grešaka. Ukoliko je proces rešavanja dao nezadovoljavajuće rezultate, kroz prva dva navedena koraka prolazi se iznova, primenjujući druge moguće pristupe.

U ostatku teksta, biće dosledno korišćena naredna notacija. Običnim malim slovima poput x označavaju se skalari. Podebljanim malim slovima poput $\mathbf{x}$ označavaju se vektori, a podebljanim velikim poput $\mathbf{X}$ označavaju se matrice. U kontekstu u kojem se javljaju podebljano slovo i obično slovo sa indeksima, podrazumeva se da obično slovo predstavlja koordinatu vektora ili element matrice označene odgovarajućim podebljanim slovom. Na primer, x_i će označavati i-tu koordinatu vektora $\mathbf{x}$, a x_{ij} odgovarajući element matrice $\mathbf{X}$. S druge strane, $\mathbf{x}_i$ označava i-ti vektor $\mathbf{x}$, a ne i-tu koordinatu vektora $\mathbf{x}$ i slično za matrice.

10.1 Modelovanje problema

Modelovanje problema polazni je deo procesa mašinskog učenja i on često zahteva najviše znanja i inventivnosti. Konkretni koraci variraju od problema do problema, ali postoje neki koraci zajednički za većinu slučajeva i to su:

- Uočavanje opšteg okvira učenja adekvatnog za dati problem, koji je pre svega uslovljen vrstom informacije date u primerima iz kojih se uči;
- Izbor reprezentacije podataka iz kojih se uči;
- Izbor forme modela zakonitosti u podacima, odnosno skupa dopustivih modela;
- Izbor funkcije greške koja meri koliko model odgovara podacima na kojima se vrši trening.

Ovi koraci biće objašnjeni u nastavku.

10.1.1 Nadgledano, nenadgledano i učenje potkrepljivanjem

Koliko god primene mašinskog učenja bile raznovrsne, postoje određene zajedničke karakteristike zadataka i procesa učenja koje se često sreću. Postoje tri glavna okvira problema učenja: *nadgledano učenje* (eng. *supervised learning*), *nenadgledano učenje* (eng. *unsupervised learning*) i *učenje potkrepljivanjem* (eng. *reinforcement learning*).

Nadgledano učenje odnosi se na situacije u kojima se algoritmu, zajedno sa podacima iz kojih uči, daju i željeni izlazi, to jest vrednosti takozvane *ciljne promenljive*. Algoritam treba da nauči da za date podatke pruži odgovarajuće izlaze. Očekuje se da izlazi dati za podatke na kojima nije vršeno učenje takođe budu dobri. Neki primeri problema nadgledanog učenja su predviđanje da li je članak računarski ili nije, da li određeno elektronsko pismo predstavlja neželjenu poštu (eng. *spam*) ili ne i predviđanje cene nekih akcija u zavisnosti od kretanja cene tih akcija u prošlosti i kretanja cena drugih akcija.

U nenadgledanom učenju algoritmu koji uči pružaju se samo podaci bez izlaza. Od algoritma koji uči očekuje se da sâm uoči neke zakonitosti u datim podacima. Primer nenadgledanog učenja je takozvano klasterovanje – uočavanje grupa na neki način sličnih objekata kada ne postoji pouzdano znanje o tome koliko grupa postoji ili koje su njihove karakteristike. Jedan primer primene klasterovanja je smanjenje skupa boja slike. Pikseli slike se mogu grupisati klasterovanjem po njihovoj blizini u RGB prostoru boja, a potom se iz svakog klastera može izabrati po jedna boja koja bi ga predstavljala i kojom bi bili obojeni svi pikseli koji pripadaju tom klasteru. Drugi primer primene klasterovanja je grupisanje (prirodnih) jezika u grupe jezika srodnog porekla.

Učenje potkrepljivanjem odnosi se na situacije u kojima je nizom akcija potrebno postići neki cilj pri čemu u podacima nije poznato koja akcija je bila dobra u kojoj situaciji, već samo finalni ishod čitavog niza akcija. Cilj je naučiti koje su akcije dobre u kojim stanjima. Tipičan primer problema pogodnih za ovu vrstu učenja je igranje igara. Naime, u igrama često nije uvek jasno koji potez je bio dobar (ili ključan), a koji nije, ali je poznato da li je partija na kraju dobijena ili izgubljena.

Primer 10.2. *Problem prepoznavanja računarskih članaka očito spada u okvir nadgledanog učenja. Naime, postoji konkretna ciljna promenljiva — da li je članak računarski ili nije i nju je potrebno predvideti, i moguće je zajedno sa podacima iz kojih se uči (člancima) pružiti i vrednosti ove ciljne promenljive. Ovaj problem ne uklapa se u druga dva opisana okvira. Ne može se očekivati da bi se nenadgledanim pristupom mogli izdiferencirati računarski i ostali članci. Iako nenadgledano učenje često omogućava grupisanje, teško je obezbediti da grupisanje bude izvršeno baš u skladu sa jednim aspektom teme članka – njenom pripadnošću oblasti računarstva. Sasvim je moguće da bi neki postojeći algoritam grupisao članke po nekom drugom kriterijumu, koji najverovatnije ne bi ni bio lako razumljiv čoveku ili dovoljno precizan za potrebe ovog problema. Na kraju, ovaj problem se ne uklapa ni u okvir učenja potkrepljivanjem, pošto rešenje predstavlja*

jednostavan odgovor da li je članak računarski, a ne niz akcija čijim se zajedničkim efektom postiže neki cilj (kao u igranju igara).

Kako je prepoznato da se radi o problemu nadgledanog učenja i da podaci pripadaju dvema klasama, za uspešno rešavanje problema potrebno je obezbediti određeni skup primera koji sadrži kako računarske, tako i članke iz drugih oblasti.

10.1.2 Reprezentacija podataka

Kao što je rečeno na početku, podacima se opisuju konkretni primerci, tj. *instance* nekog problema. Primera radi, iako možemo govoriti o opštem problemu prepoznavanja računarskih članaka, u praksi se uvek susrećemo sa njegovim konkretnim primercima – uvek je za neke konkretne članke potrebno ustanoviti da li su računarski ili nisu. Otuda se, kada se govori o podacima, često govori o instancama podataka pri čemu se podrazumeva da jedna instanca podataka opisuje jedan konkretan primerak problema koji treba rešiti.

Instance koje čine podatke treba da budu zapisane u obliku koji je pogodan za primenu algoritama učenja. Najpogodniji i najčešće korišćeni način koji se koristi u algoritmima mašinskog učenja je predstavljanje instanci nekim njihovim relevantnim *svojstavima*, tj. *atributima*, *odlikama* ili *obeležjima* (eng. *feature, attribute*). Svojstva predstavljaju karakteristike instanci kao što su boja, veličina, težina i slično. Svako od izabranih svojstava može imati vrednost koja pripada nekom unapred zadatom skupu. Te vrednosti su nekad numeričke, kao u slučaju težine, koja je skalarna veličina i koja se najbolje opisuje brojem. Primer numeričke vrednosti može biti i frekvencija reči u nekom članku, kao što je to bio slučaj u primeru sa klasifikacijom članaka. Svojstva takođe mogu biti i kategorička — mogu predstavljati imena nekih kategorija kojima se ne mogu dodeliti smislene numeričke vrednosti ili pridružiti uređenje. Primer kategoričkog svojstva može biti grad u kome osoba živi, pol, nacionalnost i slično. Skup svojstava koji će se koristiti u zapisu instance generalno nije unapred zadat, već ga je potrebno odabrati u skladu sa time koje su karakteristike instanci bitne za dati problem učenja. Na primer, za predviđanje ocene na ispitu nije bitna boja očiju studenta. Kada su izabrana svojstva pomoću kojih se instance opisuju, svaka instanca može se predstaviti vektorom svojih, konkretnih vrednosti svojstava. Ciljne promenljive obično se reprezentuju analogno reprezentovanju svojstava.

Umesto preko vrednosti izabranih atributa, podaci se nekada mogu dati algoritmu učenja i u sirovoj formi, poput matrice piksela koji čine sliku, bez opisivanja slike nekim specifičnim svojstvima. Nisu svi algoritmi učenja dizajnirani da dobro rade sa takvim reprezentacijama.

Primer 10.3. *U slučaju prepoznavanja računarskih članaka, instanca je jedan članak. Instance će biti u računaru predstavljene pomoću nekih podataka koji ih opisuju. Ciljna promenljiva y koja predstavlja klasu može biti 1 za računarske članke i -1 za ostale. Iako je predstavljena brojem, ovo je kategorička vrednost, pošto se radi o dve kategorije za koje su ovi brojevi proizvoljno izabrani.*

Da bi bilo sprovedeno učenje, potrebno je raspoložive članke predstaviti u nekom obliku koji je pogodan za algoritam učenja i koji bi mogao da na neki način sažme osnovne karakteristike na osnovu kojih se članci iz ove dve kategorije mogu razlikovati. Kao što je ranije zapaženo, očekivano je da će u člancima iz računarstva biti češće pominjani računarski pojmovi nego u ostalim člancima. To svojstvo bi se moglo iskoristiti za razlikovanje članaka. U skladu sa ovim, mogu se nabrojati sve reči iz nekog rečnika računarske terminologije. Svaki članak može biti predstavljen vektorom frekvencija ovih reči. Ako je $\mathbf{x}$ vektor koji odgovara nekom članku, onda će x_i označavati frekvencije izabranih pojedinačnih reči.

Opisani vektori frekvencija predstavljaju tačke u euklidskom prostoru. Kao što je diskutovano ranije, ako se u pojednostavljenom rečniku nalaze samo dve reči — „računar“ i „datoteka“, ove tačke mogu izgledati kao na slici u primeru 10.1. Ukoliko su u člancima iz jedne kategorije ovi računarski termini visokofrekventni, a u drugim niskofrekventni, tačke koje odgovaraju računarskim člancima će se grupisati dalje od koordinatnog početka, dok će se ostale grupisati bliže njemu.

10.1.3 Skup dopustivih modela

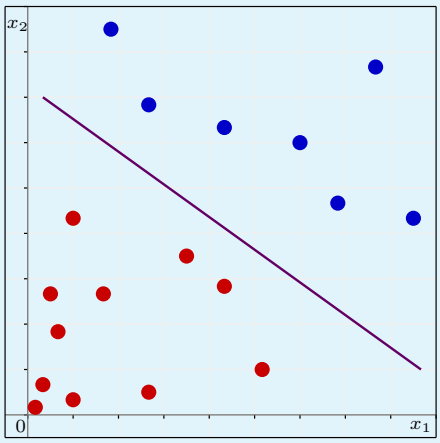
Proces učenja može se razumeti kao proces pronalaženja zakonitosti u podacima ili, preciznije, zavisnosti među promenljivim koje ih opisuju, a koje čine svojstva i ciljna promenljiva. Ako je raspoloživ dobar opis zavisnosti izmedju svojstava i ciljnih promenljivih za postojeće podatke, onda ga možemo upotrebiti za predvidjanje ciljnih vrednosti za nove, neviđene instance. Kako bi se učenje moglo automatizovati, potrebno je da opšta forma tih zavisnosti bude matematički definisana. Matematičke reprezentacije zavisnosti među promenljivim nazivamo *modelima*. Ovaj pojam vrlo je blizak pojmu modela u empirijskim naukama, koji takođe ustanovljava zavisnosti između veličina koje su relevantne za proučavani fenomen (na primer, zavisnosti između brzine, puta i vremena).

Obično modeli koji se razmatraju imaju unapred određenu opštu formu i moguće je uočiti *skup dopu-*

stivih modela. Forme modela mogu biti raznovrsne: modeli mogu biti pravila oblika `IF...THEN` (na primer, `IF x1>3 and x2<2 THEN return 1`), linearne funkcije svojstava (na primer, $w_1x_1 + w_2x_2$), nelinearne funkcije svojstava (na primer, $\log(w_1x_1 + w_2x_2)$) i tako dalje.

Primer 10.4. *U slučaju prepoznavanja računarskih članaka, pretpostavimo da se članci prepoznaju na osnovu frekvencija (x_1 i x_2) dve izabrane reči. Kao skup dopustivih modela može da se koristi skup linearnih modela $\{w_1x_1 + w_2x_2 + b \mid w_1, w_2, b \in \mathbb{R}\}$, odnosno njima odgovarajućih pravih.*

Između dve grupe tačaka možda postoji prava (koja odgovara nekom dopustivom modelu) koja ih razdvaja, kao na sledećoj slici.



Ako je ova prava poznata, onda neki nov, nepoznati članak može biti prepoznat kao članak iz oblasti računarstva ukoliko se tačka koja mu odgovara nalazi sa iste strane prave kao i tačke računarskih članaka koji su nam poznati. U suprotnom, smatra se da članak nije iz oblasti računarstva.

Izbor skupa dopustivih modela od fundamentalnog je značaja za kvalitet učenja. Ukoliko ovaj skup nije dovoljno bogat, onda učenje, jasno, ne može biti dovoljno dobro. Naizgled paradoksalno, i preterano bogatstvo skupa dopustivih modela može da dovede (i često dovodi) do loših rezultata. Ovaj fenomen biće diskutovan u delu 11.3.

10.1.4 Funkcija greške

U mašinskom učenju greška je nužno prisutna, ali je i nepoželjna. Stoga je potrebno meriti kolike su greške koje model pravi. Tome služi funkcija greške (eng. *loss*) koja se može razlikovati od problema do problema.

U slučaju nadgledanog učenja, ova funkcija treba da kvantifikuje odstupanje predviđene od stvarne vrednosti ciljne promenljive za neku instancu. U slučaju nenadgledanog učenja, ako je potrebno, na primer, izvršiti identifikaciju grupa u podacima, greška treba da kvantifikuje raznolikost podataka unutar grupa, pošto je poželjno da podaci unutar jedne grupe budu što sličniji. U slučaju učenja potkrepljivanjem, obično se govori o nagradi umesto o grešci, a nagrada je utoliko veća ukoliko je posao bolje obavljen nizom akcija koje se preduzimaju. Funkcija greške definiše se za jednu instancu, ali se ta definicija proširuje i do ukupne greške za veći broj instanci, obično kao suma ili prosek grešaka na pojedinačnim instancama. Uloga funkcije greške je sledeća: ukupna greška nad trening podacima se (u fazi treniranja) minimizuje i iz skupa dopustivih modela bira se onaj koji daje najmanju grešku. Ukoliko taj model daje najmanju grešku nad trening podacima, očekuje se da će davati male greške i nad budućim, neviđenim podacima. Funkcija greške obično se bira tako da pored smislenosti u odnosu na problem ima i neka poželjna svojstva (koja olakšavaju postupak minimizacije) poput neprekidnosti, diferencijabilnosti, konveksnosti i slično.

Primer 10.5. *Mogući su različiti izbori funkcije greške. U primeru prepoznavanja računarskih članaka, jedan izbor bi mogao biti 1 ako je instanca loše klasifikovana, a 0 ako je dobro klasifikovana, ali ta funkcija nema dobra tehnička svojstva i otud se ne koristi. Drugi prirodan izbor bi se mogao voditi idejom da je greška utoliko veća ukoliko je instanca dalje od prave koja razdvaja klase, a sa njene pogrešne strane. Ukoliko je tačka sa ispravne strane ove prave, znak funkcije*

$$w_1x_1 + w_2x_2 + b$$

se poklapa sa znakom ciljne promenljive y (1 ili -1) i proizvod $y \cdot (w_1x_1 + w_2x_2 + b)$ je pozitivan. Ukoliko je tačka sa pogrešne strane, taj proizvod je negativan i njegova apsolutna vrednost utoliko je veća što je tačka dalje od prave, na pogrešnoj strani. Otud se za grešku, kada postoji, može uzeti izraz

$$-y \cdot (w_1x_1 + w_2x_2 + b)$$

Kako bi se izbegla negativna greška u slučaju tačne klasifikacije, ovaj izraz može se zameniti sledećim

$$\max(0, -y \cdot (w_1x_1 + w_2x_2 + b))$$

tako da je u slučaju tačne klasifikacije greška jednaka nuli. Onda prava koja ispravno razdvaja sve članke pravi grešku 0 na svim člancima. Dodatno, ova funkcija je neprekidna, konveksna i diferencijabilna svugde osim u jednoj tački.

Ponovimo da su oznake klasa 1 i -1 zapravo kategoričke vrednosti. Pogodni izbori ovih vrednosti mogu voditi jednostavnijoj formulaciji funkcije greške, kao u ovom primeru.

10.2 Rešavanje problema

Rešavanje problema u mašinskom učenju svodi se na pronalaženje modela koji pravi najmanju grešku (prema izabranom kriterijumu) na datim podacima. To rešavanje se onda može razumeti i kao pretraga skupa dopustivih modela vođena podacima, a koju realizuje algoritam učenja. Taj algoritam često se svodi na neki optimizacioni metod poput gradijentnog spusta nad sumom grešaka modela na instancama trening skupa. Proces pronalaženja adekvatnog modela naziva se *treningom*.

Algoritam: Algoritam za klasifikaciju članaka

Ulaz: Trening skup T, brzina učenja α i preciznost ε

Izlaz: Parametri modela $\mathbf{w} = (w_1, w_2, b)$

postavi (w_1, w_2, b) na $(0, 0, 0)$;
ponavljaj
    postavi (w_1', w_2', b') na (w_1, w_2, b);
    za svaku instancu $(x_1, x_2, y) \in T$ **radi**
        ako $y(w_1x_1 + w_2x_2 + b) \leq 0$ **onda**
            na w_1 dodaj $\alpha y x_1$;
            na w_2 dodaj $\alpha y x_2$;
            na b dodaj αy;
dok nije ispunjen uslov $\|(w_1, w_2, b) - (w_1', w_2', b')\| \leq \varepsilon$;
vrati (w_1, w_2, b) kao rešenje.

Slika 10.1: Algoritam za klasifikaciju članaka.

Primer 10.6. *U primeru prepoznavanja računarskih članaka, pronalaženje željene prave može se izvesti „pomeranjem“ neke polazne prave dok ona ne bude pozicionirana između tačaka koje treba da razdvaja, odnosno dok greška ne postane jednaka nuli ili, ako je to nemoguće, što manja moguća. Ovo pomeranje vrši se postepenim modifikacijama parametara w_1, w_2 i b. Veličinu tog pomeraja određuje vrednost koju nazivamo* brzinom učenja *(eng.* learning rate*). Jedan takav algoritam, poznat kao perceptron, dat je na slici 10.1.*

Važno pitanje je da li ažuriranje parametara u predloženom algoritmu vodi poboljšanju naučene funkcije (tj. tekućeg modela). Pre svega, algoritam ne menja vrednosti parametara w u slučaju da je instanca tačno klasifikovana, već samo ako nije, što deluje dobro. Dalje, vrednost α, koja kontroliše veličinu korekcije $(\alpha y x_1, \alpha y x_2, \alpha y)$, mora biti mala kako bi korekcije bile male i postepene. Vrednosti x_1 i x_2 su uvek nenegativne i stoga znak korekcije zavisi samo od znaka vrednosti y. Ukoliko je vrednost y negativna, parametri se smanjuju čime se i vrednost $w_1x_1 + w_2x_2 + b$ smanjuje. Ako pritom ta vrednost postane negativna, greška na toj instanci je eliminisana. Analogno u slučaju da je vrednost y pozitivna. Stoga, naslućujemo da se ovim algoritmom vrednosti $w_1x_1 + w_2x_2 + b$ po znaku približavaju vrednostima y. Korekcije su proporcionalne

vrednostima x_1 i x_2, odnosno veće su za parametre čija promena može više doprineti promeni vrednosti $w_1x_1 + w_2x_2 + b$ za dati primer. Ovaj postupak liči na gradijentni spust kojim se minimizuje suma, po svim instancama trening skupa, ranije definisane funkcije greške:

$$\max(0, -y \cdot (w_1x_1 + w_2x_2 + b)).$$

10.3 Evaluacija rešenja

Učenje uvek polazi od nekih podataka. Podaci na osnovu kojih se vrši generalizacija, nazivaju se *podacima za trening*, a njihov skup *trening skup*. Evaluacija naučenog znanja na podacima na osnovu kojih je učeno obično dovodi do značajno boljih rezultata od onih koji se mogu kasnije dobiti u primenama, što ne bi trebalo da iznenađuje. Naime, i u slučaju ljudskog učenja, lakše je rešavati probleme koji su već upoznati u procesu učenja, nego potpuno nove probleme. Stoga je pre upotrebe potrebno proceniti kvalitet naučenog znanja na novim, nepoznatim podacima. To se obično radi tako što se razmatra koliko je naučeno znanje kvalitetno u odnosu na neke unapred date *podatke za testiranje* čija je uloga da u evaluaciji simuliraju (nepoznate) podatke na kojima će model biti primenjivan u budućnosti. Podaci za testiranje čine *test skup*. Test skup treba da bude disjunktan sa trening skupom. Kvalitet modela na test skupu može se meriti računanjem različitih statistika čija je uloga da što bolje opišu poklapanje predviđanja modela sa stvarnim podacima. U praksi, trening i test skup se dobijaju tako što se raspoloživi podaci podele na dva dela. Naravno, kako različite podele na trening i test skup mogu uroditi različitim rezultatima, slučajno deljenje nije najbolji način formiranja trening i test skupa, posebno ako količina raspoloživih podataka nije veoma velika. U praksi se koriste nešto komplikovanije sheme evaluacije, o čemu će biti reči kasnije.

Primer 10.7. *U primeru prepoznavanja računarskih članaka, pre treninga mogli smo odvojiti, na primer, trećinu ukupnih raspoloživih podataka za testiranje, a trening sprovesti nad preostale dve trećine. Kad su članci klasifikovani, kao mera kvaliteta učenja može se na test skupu izračunati udeo dobro klasifikovanih članaka u ukupnom broju članaka. Poželjno je da on bude što veći.*

10.4 Eksploatacija modela

Nakon što je model treniran i nakon što je evaluacijom ustanovljeno da je dovoljno dobar za praktičnu primenu, potrebno ga je uposliti u rešavanju instanci problema za koji je namenjen. Razvijeni model, sa izabranim i fiksiranim vrednostima parametara, primenjuje se u praksi za resavanje novih instanci. One se, kao i u ranijim fazama, reprezentuju atributima i daju se modelu kao ulaz. Model daje izlaz koji se prosleđuje ostatku sistema koji donosi dalje odluke na osnovu tog izlaza. Početak eksploatacije modela ne predstavlja kraj brige o modelu, naprotiv. U toku eksploatacije, može doći do promena u svojstvima podataka. Otud je potrebno povremeno proveravati kvalitet modela i u fazi eksploatacije i ponovo ga trenirati kada se ukaže potreba.

Primer 10.8. *U primeru prepoznavanja računarskih članaka, model bi bio ugrađen u pretraživač interneta čiji je cilj da prepoznaje takve članke. Svi članci bili bi predstavljeni frekvencijama reči i model bi bio primenjivan na njih. Članci koji bi bili prepoznati kao računarski, prikazali bi se korisniku, dok bi ostali bili preskočeni.*

Usled rastućeg prisustva računarske tehnologije u svakodnevnom životu, frekvencija računarskih reči u neračunarskim člancima može se povećati. Otud model vremenom može korisnicima prikazivati sve više nerelevantnih članaka.

Postoji dosta specifičnih problema vezanih za eksploataciju modela. Primera radi, neki modeli treba da rade na računarima ograničene brzine i memorije. Otud ih je potrebno adektavno postprocesirati kako bi se brže izvršavali. Primer ovakvog postprocesiranja je predstavljanje parametara modela u brojevima u pokretnom zarezu sa manjom preciznošću. Neki modeli treba da budu u stanju da odgovaraju na veliki broj zahteva u kratkom vremenskom periodu. To se može postići paralelizacijom na adekvatnoj serverskoj arhitekturi. Neki modeli mogu biti izloženi zlonamernim napadima čiji je cilj da zbune model kako bi doneo pogrešnu odluku. Prevencija takvih napada je i dalje aktivna tema naučnog istraživanja. Putem modela od treninga do njegove isporuke i eksploatacije i praćenjem njegovih performansi u primeni bavi se disciplina koja se naziva MLOps (eng. machine learning operations). Njeni detalji su van okvira ove knjige.

Glava 11

Nadgledano mašinsko učenje

Nadgledano mašinsko učenje karakteriše se time da su za sve raspoložive podatke poznate vrednosti ciljne promenljive. U nastavku su opisani osnovni problemi nadgledanog učenja – klasifikacija i regresija, zatim pojmovi zajednički za većinu algoritama nadgledanog mašinskog učenja, potom konkretni modeli nadgledanog učenja i algoritmi za njihovo treniranje i, na kraju, primeri praktičnih primena.

11.1 Klasifikacija i regresija

Problemi nadgledanog mašinskog učenja najčešće se mogu svrstati u jednu od dve kategorije – u probleme klasifikacije ili u probleme regresije.

Klasifikacija se sastoji u razvrstavanju nepoznate instance u jednu od unapred ponuđenih kategorija, tj. klasa. Neki od primera klasifikacije su razvrstavanje bankovnih transakcija u rizične, koje mogu predstavljati prevaru ili u nerizične, koje predstavljaju uobičajene transakcije, određivanje autorstva tekstova pri čemu se tekstu nepoznatog autora pridružuje jedan od nekoliko unapred ponuđenih autora, razvrstavanje elektronske pošte u željenu i neželjenu i slično. U navedenim primerima, svakoj instanci (bankovna transakcija, tekst, elektronska poruka) odgovara vektor vrednosti nekih izabranih svojstava. Svakoj instanci takođe odgovara i oznaka klase kojoj ta instanca pripada. Problem klasifikacije sastoji se u određivanju vrednosti klase na osnovu svojstava instance. Ključno zapažanje je da ciljna promenljiva u ovom problemu ima konačan skup mogućih vrednosti i da se, u opštem slučaju, oznakama klasa ne mogu smisleno dodeliti numeričke vrednosti, niti uređenje. Dakle, klasa, čiju je vrednost potrebno odrediti, kategoričko je svojstvo.

Problem regresije predstavlja problem u kojem pored vrednosti svojstava svakoj instanci odgovara i neka neprekidna numerička vrednost koju je potrebno predvideti. Primene regresije su mnogobrojne. One uključuju procenu budućih cena akcija na berzi, određivanje pozicija objekata na slikama, predviđanje vremena vožnje u saobraćaju do odredišta, predviđanje koncentracija zagađivača u životnoj sredini, predviđanje svojstava hemijskih jedinjenja, itd.

11.2 Minimizacija greške modela

Mašinsko učenje obično pronalazi veze između nekih veličina. Te veze zovemo modelima i predstavljamo ih matematičkim funkcijama.

Definicija 11.1 (Model)**.** *Model je funkcija $f_{\mathbf{w}}(\mathbf{x})$ koja izražava zavisnost između atributa $\mathbf{x}$ i ciljne vrednosti, gde je $\mathbf{w}$ vektor* parametara modela *koji ga određuju.*

Zakonitosti koje opisuje model koriste se za donošenje zaključaka o nepoznatim instancama problema.

Primer 11.1. *U primeru prepoznavanja računarskih članaka, model je linearna funkcija*

$$w_1 x_1 + w_2 x_2 + b$$

a vektor parametara modela je $\mathbf{w} = (w_1, w_2, b)$. Ukolko su vrednosti ovih parametara $\mathbf{w} = (0.5, 1, 1)$ i ukoliko su frekvencije razmatranih reči $\mathbf{x} = (2, 3)$, onda će vrednost modela biti jednaka

$$0.5 \cdot 2 \ + \ 1 \cdot 3 \ + \ 1 \ = \ 5.$$

Pošto je dobijen pozitivan broj, smatra se da model predviđa da je članak računarski.

Odstupanja stvarnih vrednosti ciljne promenljive od predviđanja modela zovemo greškama. Greška se kvantifikuje *funkcijom greške* (eng. *loss*). Mnogi algoritmi klasifikacije i regresije zasnivaju se na minimizaciji greške, pri čemu se iste funkcije greške mogu koristiti u kombinaciji sa različitim algoritmima i modelima. Čak i kada greška ne figuriše eksplicitno u formulaciji problema učenja, često se može uočiti pažljivom analizom. Diskusija u nastavku pretpostavlja da će funkcija greške biti eksplicitno formulisana.

Primer 11.2. *Čest izbor funkcije greške predstavlja kvadrat razlike predviđene realne vrednosti $f_{\mathbf{w}}(\mathbf{x})$ i ciljne vrednosti y:*

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = (f_{\mathbf{w}}(\mathbf{x}) - y)^2$$

Naravno, za njeno izračunavanje potrebno je da su nad vrednostima ciljne promenljive definisane aritmetičke operacije, što ne važi za proizvoljnu ciljnu promenljivu (na primer, ako ciljna promenljiva ima vrednosti „crveno" i „plavo"). Ipak, tamo gde je primenljiva, ova funkcija ponaša se intuitivno — velike razlike između predviđene i stvarne vrednosti proizvode veliku vrednost greške, a takođe je i matematički pogodna zbog svoje diferencijabilnosti. Zbog toga predstavlja čest izbor funkcije greške, mada nije najbolji izbor ni uvek kada je primenljiva.

Funkcija greške primenjuje se na jednu instancu podataka, što nije dovoljno da bi se okarakterisao kvalitet predviđanja modela. Za to se koristi naredni pojam greške modela (koji se nekad naziva i *rizikom*).

Definicija 11.2 (Greška modela)**.** *Ako je L funkcija greške, onda se* greška modela $f_{\mathbf{w}}$ *definiše kao očekivanje funkcije greške u odnosu na raspodelu podataka koji se modeluju:*

$$E(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)} L(f_{\mathbf{w}}(\mathbf{x}), y)$$

gde je $\mathbb{E}_{(\mathbf{x},y)}$ matematičko očekivanje po svojstvima i ciljnoj promenljivoj.

Kada su definisani forma modela i greška modela, lako je formulisati kriterijume za izbor najboljeg modela — to je model koji pravi najmanju grešku.

Definicija 11.3 (Najbolji model)**.** Najbolji model *je model određen parametrima kojima se minimizuje greška modela, tj. model određen parametrima koji su rešenje sledećeg problema minimizacije:*

$$\min_{\mathbf{w}} E(\mathbf{w}) \ .$$

Za izračunavanje matematičkog očekivanja $\mathbb{E}_{(\mathbf{x},y)}$, potrebno je poznavati raspodelu promenljivih $\mathbf{x}$ i y (za sve parove, ne samo one u raspoloživim podacima), koja obično nije poznata. Međutim, ono što u praksi jeste poznato je uzorak podataka na osnovu kojeg se greška može empirijski aproksimirati.

Definicija 11.4 (Empirijska greška)**.** *Neka je dat trening skup $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$.* Empirijska greška modela *određenog parametrima $\mathbf{w}$ je uzoračka aproksimacija njegove greške:*

$$E(\mathbf{w}) \approx \frac{1}{N} \sum_{i=1}^{N} L(f_{\mathbf{w}}(\mathbf{x}_i), y_i)$$

Primer 11.3. *U primeru klasifikacije članaka, (empirijska) greška modela data je funkcijom (videti primer 10.6):*

$$E(\mathbf{w}) \approx \frac{1}{N} \sum_{i=1}^{N} \max(0, -y \cdot (w_1 x_1 + w_2 x_2 + b)).$$

Vrednosti parametara $\mathbf{w}$ u praksi se biraju minimizacijom vrednosti empirijske greške modela. Kako prosek i suma imaju isti minimum, obično se prilikom minimizacije ne vodi računa o tome da li je izvršeno deljenje brojem instanci. To će biti vidljivo i u minimizacionim problemima u nastavku.

Za minimizaciju greške modela koriste se metode matematičke optimizacije. Jedna od klasičnih metoda korišćenih u ovom kontekstu je gradijentni spust (poglavlje 4.1). Taj metod zahteva da je funkcija koja se

minimizuje diferencijabilna. Postoje i druge optimizacione metode koje mogu biti pogodnije za optimizacioni problem koji se rešava. Takve metode (kao i sam gradijentni spust) obično omogućavaju podešavanje dužine koraka (videti poglavlje 4.1.2). U kontekstu mašinskog učenja, dužina koraka se obično naziva *brzinom učenja* i predstavlja hiperparametar koji je važno dobro izabrati. Napomenimo da dve metode za minimizaciju greške ne daju nužno iste vrednosti parametara $\mathbf{w}$. Štaviše, ista metoda minimizacije može (zbog mogućih nasumičnih inicijalizacija) u različitim pokretanjima da da različite vrednosti parametara, koje odgovaraju različitim lokalnim minimumima empirijske greške.[1]

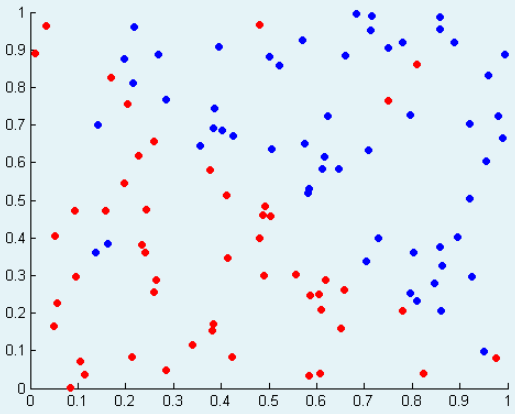
U praktičnim primenama, kada broj instanci za treniranje može da se meri i milionima, raspoloživi resursi često ne dopuštaju minimizaciju empirijske greške modela za sve trening podatke odjednom. Umesto toga, u iteracijama se minimizuje greška za nasumično kreirane podskupove trening skupa i taj postupak konvergira. Ovaj postupak minimizacije zaustavlja se kada je postignut neki zadati kriterijum (na primer, broj iteracija, vreme izvršavanja, mala razlika između tekućih vrednosti parametara). Ovaj pristup obično daje slično rešenje kao kada se minimizuje greška za sve podatke istovremeno.

11.3 Preprilagođavanje i regularizacija

Očigledno, što je greška modela manja, to je prilagođenost modela podacima za trening veća i obratno. Dakle, vrednost greške modela može da ima ulogu mere prilagođenosti modela podacima za trening.

Uslovi pod kojima se minimizacijom greške dobro aproksimira polazni problem nisu uvek ispunjeni, pa prethodno opisani postupak minimizacije i izbora modela ne vodi nužno dobrim rezultatima učenja. Naime, osnovna prepreka dobroj aproksimaciji optimalnih vrednosti parametara, a time i uspešnoj generalizaciji, je preterano bogatstvo skupa dopustivih modela. Ukoliko je taj skup toliko bogat da u njemu za svaki trening skup postoji model koji ga savršeno opisuje, onda ne postoje garancije za uspešno učenje. Dodatno, što je skup dopustivih modela bogatiji, to je potrebno više podataka za uspešno učenje. Ilustrovaćemo ovaj uvid kroz dva primera, od kojih se prvi odnosi na klasifikaciju, a drugi na regresiju.

Primer 11.4. *Neka je dat trening skup instanci koje predstavljaju članke, od kojih su neki računarski, a neki ne. Taj skup prikazan je na sledećoj slici.*

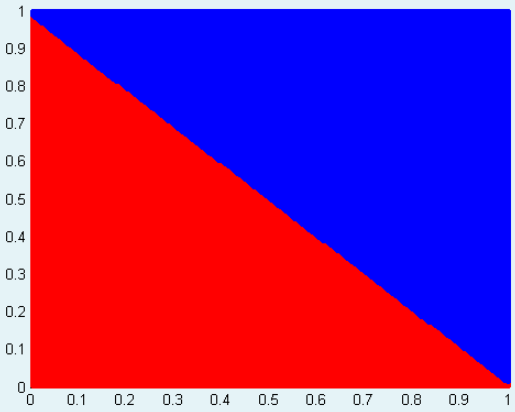


U tom skupu postoje i neki računarski članci sa niskom frekvencijom reči iz specifično računarske terminologije, ali i neki članci koji nisu računarski, a ipak imaju visoku frekvenciju računarskih termina. U praksi je česta situacija da, iz različitih razloga, određeni broj instanci odstupa od očekivanja.

Pretpostavimo da je forma modela linearna, kao i do sada — $f_{\mathbf{w}}(\mathbf{x}) = w_1x_1 + w_2x_2 + b$ i da su parametri $\mathbf{w}$ određeni minimizacijom greške pri čemu je za funkciju greške korišćen kvadrat razlike ciljne i predviđene vrednosti (u ovom konkretnom problemu postoje i pogodnije funkcije greške, ali će za potrebe ovog primera i kvadrat razlike biti dovoljno dobra funkcija). Naredna slika ilustruje rešenje u tom slučaju. Tačke u ravni

[1] Lokalni minimumi u rešavanju optimizacionih problema (ne samo onih vezanih za mašinsko učenje) mogu predstavljati veliki problem. Taj problem često se ublažava upotrebom metaheuristika ili pokretanjem optimizacionog procesa više puta sa različitim polaznim vrednostima. Dugo je smatrano da lokalni minimumi predstavljaju značajan problem u mašinskom učenju i to je decenijama služilo kao lenjo objašnjenje za svaki model koji nije dostizao očekivane performanse. Vremenom se došlo do empirijskih i (ograničenih) teorijskih rezultata koji sugerišu da lokalni minimumi ne predstavljaju suštinski problem u visokodimenzionalnim prostorima parametara i da su razlozi zbog kojih se ne pronalaze kvalitetni modeli obično neki drugi (na primer, gradijenti koji se izračunavaju u procesu optimizacije mogu imati previše malu ili previše veliku normu, podaci nisu kvalitetni, itd).

za koje model daje pozitivnu vrednost (tj. previđanje da je članak računarski) označene su plavo, a tačke za koje daje negativnu vrednost (tj. previđanje da je članak nije računarski), označene su crveno.

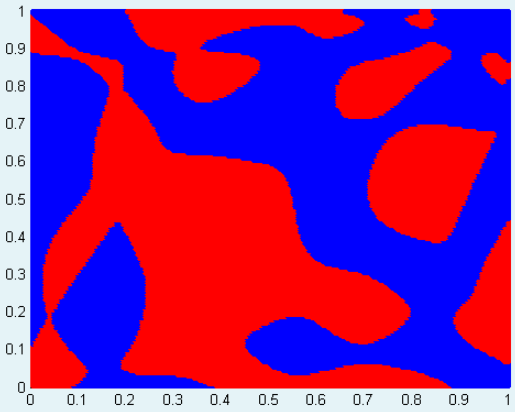


Može se primetiti da model nije saglasan sa svim instancama iz trening skupa, odnosno da postoje računarski članci koji nisu prepoznati i članci koji su prepoznati kao računarski, a to nisu. Međutim, to ne bi trebalo da bude zabrinjavajuće, pošto je za većinu članaka klasifikacija ispravna. Članci koji su pogrešno klasifikovani odstupaju od trenda učestalog korišćenja reči „računar" i „datoteka" u takvim člancima, ali ih nema dovoljno da bi sugerisali da i drugi članci u kojima se te reči retko koriste treba da budu klasifikovani kao računarski.

Naredna slika prikazuje klasifikaciju raspoloživih podataka korišćenjem modela iz skupa svih polinoma dve promenljive nekog fiksiranog stepena n, odnosno u slučaju da se za modele koristi sledeća forma:

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{n} \sum_{j=0}^{i} w_{ij} x_1^j x_2^{i-j}$$

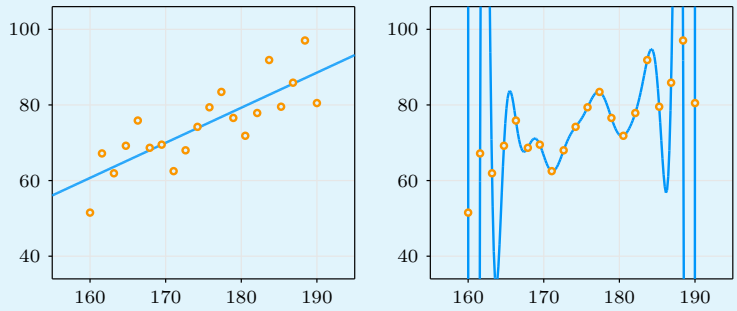
dok je funkcija greške ista.



Izabrani model saglasan je sa svim instancama iz trening skupa i zbog toga je greška jednaka nuli. Međutim, zakonitost koju ovaj model opisuje ne izgleda uverljivo. Naime, intuitivno je da su računarski članci koji ne sadrže računarske termine redak izuzetak, a ne da postoje velike oblasti prostora svojstava koje odgovaraju niskim frekvencijama računarskih termina, a koje se ipak odnose na računarske članke. Vredi primetiti i da se u oblasti za koju bi se očekivalo da je plava, nalazi veliki potprostor obojen crvenom bojom, a u kojem među trening instancama ne postoji nijedna crvena tačka. Ovakve „zakonitosti" čine korišćenje ovakvog modela u predikciji potpuno nepouzdanim i izvesno je da je stvarna greška daleko veća od nule.

Primer 11.5. *Pretpostavimo da je dat trening skup od 20 instanci koje se sastoje od jednog svojstva (visine) i vrednosti ciljne promenljive (telesne mase). Pretpostavimo da je forma razmatranih modela linearna — $f_{\mathbf{w}}(\mathbf{x}) = w_1 x + b$ i da su parametri $\mathbf{w}$ određeni minimizacijom greške, pri čemu je za funkciju greške korišćen*

kvadrat razlike ciljne i predviđene vrednosti. Naredna slika (levo) ilustruje takav pristup.



Može se primetiti da model nije u potpunosti saglasan ni sa jednom instancom iz trening skupa: za svaku instancu postoji manja ili veća greška u predviđanju. Dakle, jednostavan linearni model nije dovoljno fleksibilan da se može potpuno prilagoditi podacima za trening. S druge strane, očigledno je da on dobro opisuje opšti linearni trend koji u podacima postoji i, posebno važno, za očekivati je da greška na novim podacima iz iste raspodele bude približno jednaka grešci na poznatim, trening podacima.

Prethodna slika (desno) prikazuje aproksimaciju datih podataka korišćenjem modela iz skupa svih polinoma stepena 19*, odnosno u slučaju da se za modele koristi forma* $f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{19} w_i x^i$*. Izabrani model saglasan je sa svim instancama iz trening skupa i stoga je greška modela jednaka nuli. Međutim, posmatrajući globalni izgled izabranog modela, vidi se da on zapravo ne opisuje nikakvu zakonitost u podacima. Oscilacije koje pravi između tačaka čine njegovo korišćenje u budućem predviđanju potpuno nepouzdanim i izvesno je da je stvarna greška ovog modela daleko veća od nule.*

Problem koji se u prethodnim primerima javlja proističe upravo iz toga što skup svih polinoma čini previše bogat skup mogućih modela. Za svaki trening skup može se naći model koji ga savršeno opisuje. Međutim, prilagođavajući se trening podacima do krajnosti, gubi se svaka moć generalizacije. Takav zaključak važi i za druge previše bogate skupove dopustivih modela, a ne samo za polinome. Ilustrovani fenomen naziva se *preprilagođavanje* (eng. *overfitting*) i predstavlja jedan od važnih izazova u mašinskom učenju.

U svetlu prethodnog zaključka, teži se smanjivanju fleksibilnosti dopustivih modela. Fleksibilnost modela može se smanjiti ako se unapred izabere skup dopustivih modela tako da bude relativno siromašan. Na primer, linearna forma modela sa ograničenim brojem parametara može biti nefleksibilna. Alternativno, fleksibilnost modela može se smanjiti tako što se modifikuje funkcija greške koja se minimizuje i to tako da nepoželjni modeli ima visoku vrednost te funkcije. Često korišćen način da se to postigne je postupak *regularizacije*, koji može biti sproveden na različite načine a ovde će biti prikazan najstariji i najuobičajeniji. Umesto minimizacije greške, vrši se minimizacija regularizovane greške, odnosno, rešava se problem minimizacije

$$\min_{\mathbf{w}} \left(E(\mathbf{w}) + \lambda \Omega(\mathbf{w})\right)$$

gde je $\Omega(\mathbf{w})$ *regularizacioni izraz*, a λ *regularizacioni hiperparametar*,[2] pri čemu važi $\lambda \geq 0$. Regularizacioni izrazi obično su zasnovani na normama, pa su uobičajeni izbori poput

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^{n} w_i^2$$

ili

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^{n} |w_i|$$

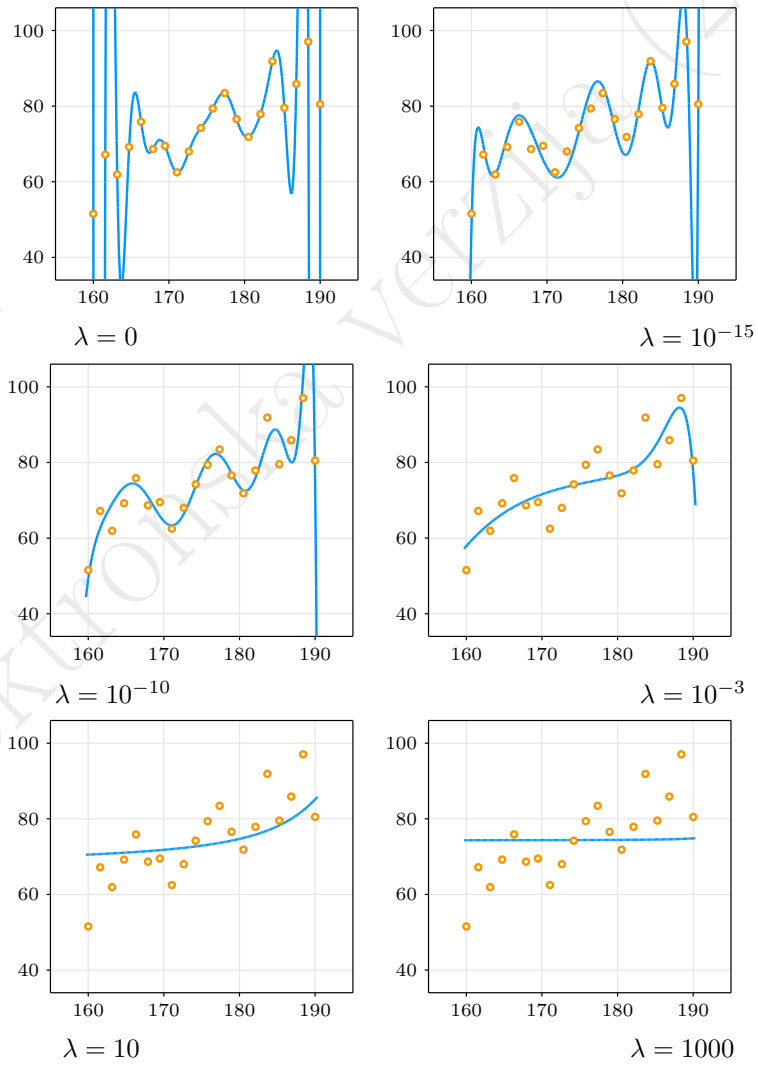
ali se koriste i drugi. Minimizacija greške, koja meri prilagođenost modela podacima, zahteva odstupanje nekih parametara w_i od nule. Međutim, dodavanjem regularizacionog izraza, takvo odstupanje kažnjava se utoliko više što je odstupanje veće. Time se otežava preveliko prilagođavanje modela podacima, tj. fleksibilnost modela se smanjuje. Mera u kojoj regularizacioni izraz utiče na model kontroliše se izborom hiperparametra λ (više o načinima izbora ovog hiperparametra biće reči kasnije). Lošim rezultatima vode i premale i prevelike vrednosti hiperparametra λ. Prve zbog preprilagođavanja, a druge jer se za njih dobijaju nefleksibilni modeli koji se ne mogu dovoljno prilagoditi podacima.

[2] Termin *hiperparametar* koristi se za veličine koje su zadate unapred i učestvuju u definiciji optimizacionog problema (poput vrednosti λ), čime se potcrtava razlika u odnosu na veličine koje su uče (poput vektora $\mathbf{w}$) i koje zovemo parametrima. Termin *hiperaparametar* koristi se u istom duhu i u nenadgledanom učenju i u učenju potkrepljivanjem.

Primer 11.6. *Neka se u primeru predviđanja telesne težine na osnovu visine koristi model predstavljen polinomom stepena* 19*, kao funkcija greške neka se koristi kvadrat razlike ciljne i predviđene vrednosti i neka se koristi regularizacija. Tada je potrebno rešiti sledeći problem minimizacije:*

$$\min_{\mathbf{w}} \left(\sum_{i=1}^{N} \left(\sum_{j=0}^{19} w_j x_i^j - y_i \right)^2 + \lambda \sum_{j=1}^{19} w_j^2 \right)$$

Za vrednosti regularizacionog hiperparametra $\lambda = 0,\ 10^{-15},\ 10^{-10},\ 10^{-3},\ 10,\ 1000$*, dobijaju se modeli prikazani na slici 11.1. Očigledno je da povećavanje regularizacionog hiperparametra* λ *smanjuje mogućnost preprilagođavanja modela, ali i da njegovo preterano povećavanje vodi njegovoj potpunoj neprilagodljivosti, usled čega je, za vrednost* $\lambda = 1000$*, model približno konstantan. Preciznije, približno je jednak pravoj* $y = w_0$*, pri čemu je* w_0 *približno jednako proseku vrednosti* y*. Primetimo da parametar* w_0 *nije uključen u regularizacini izraz, što je česta praksa, kako bi se izbeglo pomeranje modela ka koordinatnom početku.*



Slika 11.1: Polinomski modeli stepena 19 dobijeni za različite vrednosti regularizacionog hiperparametra.

Pored opisanog načina regularizacije postoje i drugi. Termin *regularizacija* često se koristi i slobodnije tako da obuhvata i druge modifikacije optimizacionog problema kojima se kontroliše fleksibilnost modela, čime se omogućava izbor modela koji nije preprilagođen i dobro generalizuje.

11.4 Linearni modeli

Linearni modeli polaze od pretpostavke da se veza između ciljne promenljive i svojstava može približno modelovati funkcijom linearnom po parametrima $\mathbf{w}$, uz eventualnu jednostavnu nelinearnu transformaciju povrh toga.

Definicija 11.5 (Linearni model)**.** Linearni model *je model oblika*

$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot h(\mathbf{x})) = g\left(\sum_{i=1}^{m} w_i h_i(\mathbf{x})\right)$$

za neke funkcije $h : \mathbb{R}^n \to \mathbb{R}^m$ *i* $g : \mathbb{R} \to \mathbb{R}$*, gde je* n *dimenzija vektora* $\mathbf{x}$*,* a m *dimenzija vektora* $\mathbf{w}$*.*

Ovako definisan model ne uklapa se u standardnu matematičku definiciju linearne funkcije (zbog transformacije g) i u nekim izvorima se pod linearnim modelima podrazumevaju samo modeli u kojima je funkcija g identička. U njima se modeli iz definicije 11.5 nazivaju se *uopštenim linearnim modelima*. Ipak, za potrebe diskusija u kontekstu mašinskog učenja, u nastavku ćemo za sve njih koristiti kraći naziv — *linearni modeli*.

Primer 11.7. *Primeri linearnih modela su:*

- $f_{\mathbf{w}}(\mathbf{x}) = w_1 x_1 + w_2 x_2$*, gde su funkcije* g *i* h *identičke funkcije.*
- $f_{\mathbf{w}}(\mathbf{x}) = w_1 + w_2 \cos(x_1) + w_3 x_2^2 + w_4 e^{x_3}$*, gde je* g *identička funkcija, a* h *je funkcija* $(x_1, x_2, x_3) \mapsto (1, cos(x_1), x_2^2, e^{x_3})$*.*
- $f_{\mathbf{w}}(\mathbf{x}) = w_1 + w_2 x_1 + w_3 x_1^2 + w_4 x_1^3 + w_5 x_1^4$*, gde je* g *identička funkcija, a* h *je funkcija* $(x_1) \mapsto (1, x_1, x_1^2, x_1^3, x_1^4)$*.*
- $f_{\mathbf{w}}(\mathbf{x}) = w_1 + w_2 x_1 + w_3 x_2 + w_4 x_1 x_2$*, gde je* g *identička funkcija, a* h *je funkcija* $(x_1, x_2) \mapsto (1, x_1, x_2, x_1 x_2)$*.*
- $f_{\mathbf{w}}(\mathbf{x}) = e^{w_1 + w_2 x_1 + w_3 x_2}$*, gde je* g *eksponencijalna funkcija, a* h *je funkcija* $(x_1, x_2) \mapsto (1, x_1, x_2)$*.*

Sledeći primeri ne predstavljaju linearne modele:

- $f_{\mathbf{w}}(\mathbf{x}) = \frac{w_1 x_1}{w_2 + x_2}$
- $f_{\mathbf{w}}(\mathbf{x}) = \frac{\cos(w_1 x_1) w_2 x_2}{e^{w_3 x_3}}$
- $f_{\mathbf{w}}(\mathbf{x}) = \frac{w_1}{\sqrt{w_1^2 + w_2^2}} x_1 + \frac{w_2}{\sqrt{w_1^2 + w_2^2}} x_2$

U nastavku diskutujemo osnovne varijante linearnih modela za regresiju i klasifikaciju.

11.4.1 Linearna regresija

Linearna regresija predstavlja metod regresije zasnovan na linearnom modelu. Pretpostavka je da je funkcija g iz definicije 11.5 identitet tj. da se predviđanje dobija kao linearna kombinacija vrednosti svojstava.

Definicija 11.6 (Model linearne regresije)**.** *Model linearne regresije je linearna kombinacija ulaznih svojstava:*

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i,$$

gde je n *dimenzija vektora* $\mathbf{x}$ *i vektora* $\mathbf{w}$*.*

Kako bi prethodni zapis uključivao i modele koji imaju slobodan član, obično se podaci proširuju novim svojstvom koje za sve instance ima vrednost 1. Razmotrimo značaj ove modifikacije kroz jednostavan primer. Ukoliko je potrebno pravom aproksimirati skup tačaka u ravni, model oblika wx bez slobodnog člana, omogućavao bi aproksimaciju isključivo pravama koje sadrže koordinatni početak. Ovakav uslov previše je restriktivan u slučaju mnogih praktičnih problema. Otud je korisno obuhvatiti i modele koji uključuju slobodan član.

Zadatak linearne regresije, kao i uopšte metoda nadgledanog učenja, je određivanje vrednosti parametara $\mathbf{w}$ koji najbolje odgovaraju trening podacima, tj. najbolje odgovaraju opažanjima iz iskustva.

Pored osnovnog zadatka pronalaženja najboljeg modela, linearna regresija korisna je i za ustanovljavanje jačine uticaja nekog svojstva na vrednost ciljne promenljive. Naime, veće apsolutne vrednosti parametra w_i označavaju jači uticaj svojstava x_i uz koji stoje. Znak parametra određuje smer uticaja svojstva. Može se meriti i statistička značajnost ovog uticaja ali se, jednostavnosti radi, u nastavku fokusiramo samo na osnovni problem određivanja optimalnih vrednosti parametara $\mathbf{w}$ i proveru kvaliteta naučenog modela.

Najjednostavniji slučaj linearne regresije je predviđanje vrednosti y na osnovu samo jednog svojstva x. Primera radi, možemo razmatrati predviđanje telesne mase na osnovu visine. Primetna je zakonitost da su visoki ljudi uglavnom teži od niskih ljudi. Potrebno je modelovati tu zavisnost. Od nje postoje i odstupanja, ali kao i inače, u mašinskom učenju prihvatamo činjenicu da će greške postojati. Dakle, u ovom slučaju razmatraćemo linearni model oblika

$$f_{\mathbf{w}}(x) = (b, w) \cdot (1, x) = b + wx$$

što je standardna jednačina linearne funkcije.

Primer 11.8. *Na već diskutovanoj slici iz primera 11.5 prikazano je 20 tačaka pri čemu svaka odgovara jednoj instanci, tj. jednom ispitaniku. Koordinata x predstavlja visinu, a y telesnu masu. Na slici se može primetiti opšti trend linearnog povećanja telesne mase u zavisnosti od visine, koji je prikazan pravom. Prikazana prava predstavlja linearni model datih podataka. Metod kojim se do njega dolazi biće prikazan u nastavku.*

Na slici je primetno i da jedan, vrlo mali broj tačaka značajno odstupa od linearnog modela. Ovakve tačke nazivamo odudarajućim podacima *(eng.* outliers*).*

Ako se koristi opšta jednačina linearnog modela (iz definicije 11.5), moguće je uključiti veći broj svojstava u predviđanje vrednosti ciljne promenljive. Naime, visina nije dovoljna da u potpunosti objasni variranje telesne težine. Dodatna svojstva koji bi vodila ka poboljšavanju predviđanja mogu da se odnose na način života pojedinaca — koliko vremena dnevno provode u sedećem položaju, koliko se bave sportom, koliko kalorija unose dnevno i slično. Umesto prave, u ovakvom slučaju regresioni model bi predstavljao jednu hiperravan.

Osnovni kriterijum izbora vrednosti parametara linearnog modela je smanjivanje odstupanja između vrednosti koje model predviđa i vrednosti koje ciljna promenljiva ima u podacima. Ovaj problem obično se formuliše kao problem minimizacije srednjekvadratne greške. Funkcija greške je $L(f_{\mathbf{w}}(\mathbf{x}), y) = (\mathbf{w} \cdot \mathbf{x} - y)^2$, pa je odgovarajući minimizacioni problem

$$\min_{\mathbf{w}} \left(\sum_{i=1}^{N} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 + \lambda \Omega(\mathbf{w}) \right)$$

pri čemu je N broj instanci u trening skupu, λ regularizacioni hiperparametar, a $\Omega(\mathbf{w})$ regularizacioni izraz (videti poglavlje 11.3). Alternativno, u matričnoj notaciji, isti problem može se zapisati kao

$$\min_{\mathbf{w}} \left(\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \Omega(\mathbf{w}) \right)$$

U slučaju da je $\Omega(\mathbf{w}) = \|\mathbf{w}\|_2^2$, ispostavlja se da za postavljeni problem postoji jednostavno (analitičko) rešenje koje ne zahteva korišćenje optimizacionih metoda:

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

pri čemu je $\mathbf{I}$ jedinična matrica i važi

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \ddots & \vdots & \\ x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Pritom, kao što je već naglašeno, prva kolona može biti kolona jedinica, čime se postiže postojanje slobodnog člana u modelu linearne regresije.

Linearna regresija ne mora uključivati regularizaciju. To se postiže postavljanjem hiperparametra λ na vrednost 0, kao što je slučaj u narednom primeru.

Primer 11.9. *Potrebno je modelovati vazdušno zagađenje izraženo u terminima indeksa kvaliteta vazduha (veće vrednosti su nepoželjnije), a u zavisnosti od populacije, maksimalne brzine vetra i indikatora da li je grad u kotlini ili ne. Raspoloživi su (stvarni) podaci za četiri dana – 5, 10, 15. i 20. januar 2020. godine*

u Beogradu, Sarajevu i Temišvaru. Želimo da dobijemo model kojim je moguće predviđati zagađenje, ali i koji je moguće analizirati. Naime, apsolutna vrednost parametara i njihov znak govore nam koliko je koje svojstvo bitno i u kom smeru njegova promena utiče na promenu ciljne promenljive.

Matrice $\mathbf{X}$, $\bar{\mathbf{X}}$ *i vektor* $\mathbf{y}$, *dati u nastavku sadrže podatke na osnovu kojih treba trenirati model linearne regresije. Matrica* $\bar{\mathbf{X}}$ *predstavlja matricu standardizovanih atributa, čije izračunavanje je objašnjeno u nastavku. Primetimo da je prva kolona matrica* $\mathbf{X}$ *i* $\bar{\mathbf{X}}$ *jednaka jedinici. Ovo omogućava postojanje slobodnog parametra u modelu linearne regresije (koji nije eksplicitno predviđen definicijom modela linearne regresije (11.6)). Takođe, svojstvo koje označava da li se grad nalazi u kotlini ili ne je kategoričke prirode. Pošto je to svojstvo binarno, njegove vrednosti moguće je kodirati brojevima* 0 *i* 1 *(u slučaju kategoričkih svojstava sa više vrednosti, bilo bi potrebno nešto komplikovanije kodiranje u koje ovde ne ulazimo). Dalje, promenljive koje koristimo u modelu izražavaju se na različitim skalama. Veličina parametara u modelu zavisi od skale na kojoj se mere vrednosti svojstava. Kako bi vrednosti parametara bile uporedive, sva svojstva treba da uzimaju vrednosti u sličnom rasponu. Jedan način da se to postigne je da se kolone matrice* $\mathbf{X}$ *(osim kolone jedinica) standardizuju – od svake vrednosti u koloni se oduzima njen prosek, a potom se sve vrednosti dele standardnom devijacijom kolone. Takve kolone sve imaju prosek* 0 *i standardnu devijaciju* 1. *Tako se od matrice* $\mathbf{X}$ *dobija matrica* $\bar{\mathbf{X}}$ *(uz zaokruživanje na dve decimale):*

$$\mathbf{X} = \begin{bmatrix} 1 & 1687132 & 35 & 0 \\ 1 & 1687132 & 13 & 0 \\ 1 & 1687132 & 10 & 0 \\ 1 & 1687132 & 13 & 0 \\ 1 & 555210 & 21 & 1 \\ 1 & 555210 & 5 & 1 \\ 1 & 555210 & 5 & 1 \\ 1 & 555210 & 13 & 1 \\ 1 & 468162 & 37 & 0 \\ 1 & 468162 & 19 & 0 \\ 1 & 468162 & 21 & 0 \\ 1 & 468162 & 16 & 0 \end{bmatrix}$$

$$\bar{\mathbf{X}} = \begin{bmatrix} 1 & 1.41 & 1.81 & -0.71 \\ 1 & 1.41 & -0.44 & -0.71 \\ 1 & 1.41 & -0.75 & -0.71 \\ 1 & 1.41 & -0.44 & -0.71 \\ 1 & -0.63 & 0.38 & 1.41 \\ 1 & -0.63 & -1.26 & 1.41 \\ 1 & -0.63 & -1.26 & 1.41 \\ 1 & -0.63 & -0.44 & 1.41 \\ 1 & -0.78 & 2.01 & -0.71 \\ 1 & -0.78 & 0.17 & -0.71 \\ 1 & -0.78 & 0.38 & -0.71 \\ 1 & -0.78 & -0.14 & -0.71 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 70 \\ 168 \\ 181 \\ 141 \\ 121 \\ 245 \\ 247 \\ 112 \\ 63 \\ 80 \\ 67 \\ 61 \end{bmatrix}$$

Primenom formule za rešavanje problema linearne regresije (za $\lambda = 0$*)*

$$\mathbf{w} = (\bar{\mathbf{X}}^T\bar{\mathbf{X}})^{-1}\bar{\mathbf{X}}^T\mathbf{y}$$

dobija se vektor parametara

$$\mathbf{w} = \begin{bmatrix} 129.67 \\ 24.19 \\ -46.11 \\ 29.83 \end{bmatrix}$$

Dobijene vrednosti parametara sugerišu određene zakonitosti – naseljenost grada je pozitivno korelisana sa zagađenjem, vetrovitost negativno, a zatvorenost u kotlinu, takođe pozitivno. Pritom, model sugeriše da je vetrovitost najvažniji od pomenuta tri svojstva. Ipak, koliko možemo verovati ovim zaključcima? Najbolje bi bilo primeniti model na podatke na kojima nije treniran, kako bi se videlo koliko precizno predviđa. Takođe, moguće je proveriti da li se model uopšte uspešno prilagodio podacima za trening. O evaluaciji modela će biti reči kasnije, ali možemo proveriti koren srednjekvadratne greške modela. On ima vrednost 29. *Poređenja radi, standardna devijacija vrednosti ciljne promenljive je* 65. *Standardna devijacija je koren*

srednjekvadratne greške u odnosu na model koji konstantno predviđa prosečnu vrednost promenljive y, te se može zaključiti da su predviđanja ovde opisanog modela značajno bolja od takvog jednostavnog modela. Ipak, ima još dosta prostora za njegovo unapređenje, što nije ni neobično imajući u vidu da model ne uzima u obzir sve veličine koji utiču na zagađenje.

Osnovni problem pri određivanju optimalnih vrednosti parametara $\mathbf{w}$ je potencijalna loša uslovljenost matrice $\mathbf{X}$ (za male promene elemenata polazne matrice, moguće su ogromne promene elemenata inverzne matrice). Naime, moguće je da su neka svojstva linearno zavisna ili da su jako korelirana. U tom slučaju, matrica $\mathbf{X}$, pa i matrica $\mathbf{X}^\top\mathbf{X}$ je neinvertibilna ili loše uslovljena, pa se, ako je $\lambda = 0$, optimalne vrednosti parametara $\mathbf{w}$ ne mogu izračunati ili su previše nestabilne. Baš zbog toga se preporučuje da se prilikom linearne regresije uvek koristi regularizacija (tj. treba da važi $\lambda \neq 0$), pri čemu se kao regularizacioni izraz najčešće koristi kvadrat euklidske norme (eng. *ridge regression*). Moguće je koristiti i druge norme kako u regularizacionom izrazu, tako i u funkciji greške, što dovodi do varijanti metode sa različitim osobinama.

U gore navedenom rešenju problema regularizovane linearne regresije

$$\mathbf{w} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

moguće je da je dimenzija matrice $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ velika i da je njeno invertovanje računski previše zahtevno. U takvim situacijama minimizacija se vrši metodama optimizacije poput gradijentnog spusta. Pošto je dokazano da je regularizovana greška u slučaju linearne regresije konveksna, gradijentni spust konvergira tački globalnog minimuma.

Osnovna i očigledna pretpostavka linearne regresije je da vezu između svojstava i ciljne promenljive izražava linearni model. Ipak, ne može se očekivati da vrednosti ciljne promenljive budu uvek jednake vrednostima modela zbog postojanja šuma, odnosno slučajne greške u podacima. Poreklo šuma može biti nesavršenost opreme kojom se vrši merenje, slučajna priroda samog fenomena ili to što izbor linearne zavisnosti predstavlja svesnu odluku da se inače kompleksna zavisnost donekle pojednostavi radi lakše analize. Stoga se često pretpostavlja da odnos svojstava i ciljne promenljive izražava sledeća veza:

$$y = \mathbf{w} \cdot \mathbf{x} + \varepsilon$$

gde je $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ normalno raspodeljena slučajna promenljiva koja označava šum, pri čemu je standardna devijacija σ konstantna. Neformalno, time se pretpostavlja da se greške „poništavaju“, odnosno da se prebacivanja i podbacivanja javljaju jednako često, da su pritom velike greške vrlo malo verovatne, kao i da veličina greške ne zavisi od vrednosti y (pošto je σ konstantno).

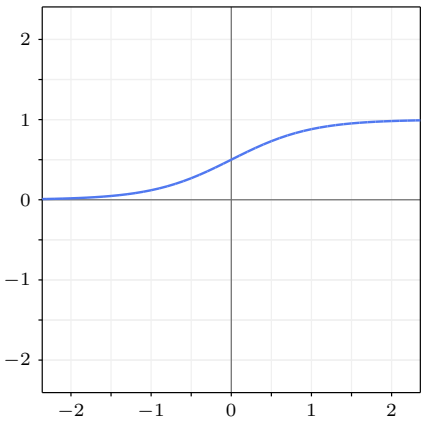
11.4.2 Logistička regresija

Logistička regresija predstavlja jednu od najkorišćenijih metoda klasifikacije. Glavni razlozi za to su jednostavnost, efikasno treniranje i postojanje verovatnosne interpretacije rezultata. Ograničenje ove metode je da je primenljiva samo na binarnu klasifikaciju — na slučaj kada svaka instanca može pripadati jednoj od dve klase koje se mogu označiti brojevima 0 i 1. Ovakav izbor brojeva samo je tehnička pogodnost i za njega ne postoji nikakav suštinski razlog, pošto su oznake klasa zapravo kategoričke vrednosti. Osnovna ideja logističke regresije je da se vrši predviđanje verovatnoće $P(y = 1|\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x})$ da instanca $\mathbf{x}$ pripada klasi 1. Očigledno, mora važiti $f_{\mathbf{w}}(\mathbf{x}) \in [0, 1]$. Tada je verovatnoća pripadnosti drugoj klasi $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - f_{\mathbf{w}}(\mathbf{x})$. Logistički model predviđa da $\mathbf{x}$ pripada klasi 1 ako je $f_{\mathbf{w}}(\mathbf{x}) > 0.5$ i da pripada klasi 0 inače (slučaj $f_{\mathbf{w}}(\mathbf{x}) = 0.5$ može da se pridruži jednoj ili drugoj klasi).

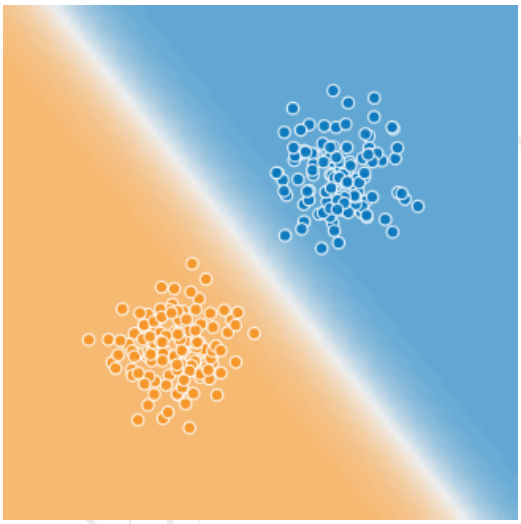
Postavlja se pitanje šta je pogodna forma za modele logističke regresije. Da bi se modelovala verovatnoća, potrebno je da model bude funkcija koja uzima sve vrednosti u intervalu $[0, 1]$. Da bi se to postiglo pomoću linearnog modela $\mathbf{w} \cdot \mathbf{x}$, potrebno je izabrati funkciju g (koja figuriše u formi modela iz definicije 11.5) tako da slika interval $[-\infty, \infty]$ u interval $[0, 1]$. Jedna takva funkcija je *sigmoidna funkcija*:[3] $\sigma(x) = 1/(1+e^{-x})$. Ovo nije jedina funkcija koja zadovoljava pomenuti zahtev, ali se tradicionalno ona koristi u mašinskom učenju. Grafik sigmoidne funkcije prikazan je na slici 11.2. Zahvaljujući ovim pojmovima moguće je definisati model logističke regresije.

Definicija 11.7 (Model logističke regresije)**.** Model logističke regresije *je kompozicija sigmoidne i linearne*

[3] Primetimo da se sigmoidna funkcija uobičajeno označava sa σ, isto kao i standarda devijacija kod normalne raspodele (kao u poglavlju 11.4.1). To je puka istorijska slučajnost: sigmoidna funkcija i standardna devijacija potpuno su različiti pojmovi.



Slika 11.2: Grafik sigmoidne funkcije.



Slika 11.3: Podaci koji se sastoje od dve klase i prava koja ih razdvaja. Boje ukazuju na regione prostora koji pripadaju različitim klasama, a bela prava u centralnom delu odgovara uslovu $P(y = 1|\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x}) = 0.5$ (slika je generisana pomoću alata A Neural Network Playground).

funkcije ulaznih svojstava i ima sledeću formu:

$$f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

Primetimo da važi $f_{\mathbf{w}}(\mathbf{x}) > 0.5$ ako važi $\mathbf{w} \cdot \mathbf{x} > 0$, a $f_{\mathbf{w}}(\mathbf{x}) < 0.5$ ako važi $\mathbf{w} \cdot \mathbf{x} < 0$. To znači da se logistička regresija može interpretirati kao metoda koja traži razdvajajuću hiperravan između instanci dve klase (pa je i kategorizacija članaka, opisana u primerima 10.1 i 11.3, tipičan primer za koji je prirodno primeniti logističku regresiju). Pritom, što je neka tačka dalja od razdvajajuće hiperravni, to je vrednost $\mathbf{w} \cdot \mathbf{x}$ veća po apsolutnoj vrednosti, a samim tim je i vrednost $\sigma(\mathbf{w} \cdot \mathbf{x})$ bliža vrednosti 0 ili 1 u zavisnosti od znaka vrednosti $\mathbf{w} \cdot \mathbf{x}$. Drugim rečima, što je instanca dublje u oblasti prostora koji pripada nekoj klasi, to model izražava veću sigurnost da ona pripada toj klasi. Ovo ponašanje potpuno je u skladu sa intuicijom. Na slici 11.3 prikazan je primer skupa podataka i prava dobijena logističkom regresijom koja ih klasifikuje.

Kao što je navedeno u motivaciji logističke regresije, verovatnoća $P_{\mathbf{w}}(y = 1|\mathbf{x})$ predviđa se formulom $P_{\mathbf{w}}(y = 1|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x})$. Takođe, važi $P_{\mathbf{w}}(y = 0|\mathbf{x}) = 1 - P_{\mathbf{w}}(y = 1|\mathbf{x}) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x})$. Odavde se može izvesti opšti zaključak:

$$P_{\mathbf{w}}(y|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x})^y (1 - \sigma(\mathbf{w} \cdot \mathbf{x}))^{1-y}.$$

Intuitivno je da vrednosti parametara treba izabrati tako da verovatnoća instanci koje čine trening skup bude maksimalna. Pod standardno korišćenom pretpostavkom da instance predstavljaju nezavisne uzorke, ta verovatnoća jednaka je proizvodu

$$\prod_{i=1}^{N} P_{\mathbf{w}}(y_i|\mathbf{x}_i)$$

koji se naziva *funkcijom verodostojnosti parametra* (eng. *likelihood function*). Korišćenje proizvoda je iz tehničkih razloga[4] nepreporučljivo, pa se umesto funkcije verodostojnosti koristi njen logaritam. Pošto je logaritam monotono rastuća funkcija, funkcija verodostojnosti i njen logaritam dostižu maksimum u istoj tački. Kako je logaritam broja koji je između 0 i 1 negativan, umesto maksimizacije logaritma verodostojnosti, može se minimizovati njegova negativna vrednost

$$\begin{aligned} -\log \prod_{i=1}^{N} P_{\mathbf{w}}(y_i|\mathbf{x}_i) &= -\sum_{i=1}^{N} \log P_{\mathbf{w}}(y_i|\mathbf{x}_i) \\ &= -\sum_{i=1}^{N} \log \left(\sigma(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}(1-\sigma(\mathbf{w}\cdot\mathbf{x}_i))^{1-y_i}\right) \\ &= -\sum_{i=1}^{N} \left(y_i \log \sigma(\mathbf{w}\cdot\mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}_i))\right) \end{aligned}$$

Po dodavanju regularizacije, minimizacioni problem koji se rešava postaje:

$$\min_{\mathbf{w}} -\left(\sum_{i=1}^{N} \left(y_i \log \sigma(\mathbf{w}\cdot\mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}_i))\right) + \lambda\Omega(\mathbf{w})\right)$$

Na prethodni način došli smo do izraza koji se minimizuje, a koji se i inače često koristi kao greška u problemima klasifikacije i koji zovemo *binarna unakrsna entropija*.[5]

Definicija 11.8 (Binarna unakrsna entropija)**.** Binarna unakrsna entropija *je sledeća funkcija greške:*

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = -y \log f_{\mathbf{w}}(\mathbf{x}) - (1-y)\log(1-f_{\mathbf{w}}(\mathbf{x}))$$

Minimizacija ukupne ovako definisane greške nema jednostavno analitičko rešenje kao u slučaju linearne regresije, već se mora sprovesti postupak optimizacije. U tu svrhu moguće je koristiti gradijentni spust, ali se često koriste i efikasnije metode poput Njutnove. Može se dokazati da je ova funkcija koja se minimizuje konveksna funkcija koja ima jedan globalni minimum. To čini značajnu pogodnost, jer ne postoji mogućnost da proces optimizacije završi u nekom neoptimalnom lokalnom minimumu (što je problem sa nekim drugim metodama učenja, poput neuronskih mreža).

Treba imati u vidu da za primenu logističke regresije nije neophodno da klase budu linearno razdvojive. Trening logističke regresije sigurno konvergira zahvaljujući tome što metode optimizacije koje se koriste u ovom problemu sigurno uspešno aproksimiraju minimum konveksne funkcije. Naravno, preciznost dobijenog modela ne može biti savršena ako se radi o linearno nerazdvojivom problemu. Razlike između linearno razdvojivog i linearno nerazdvojivog problema ilustrovane su sledećim primerom.

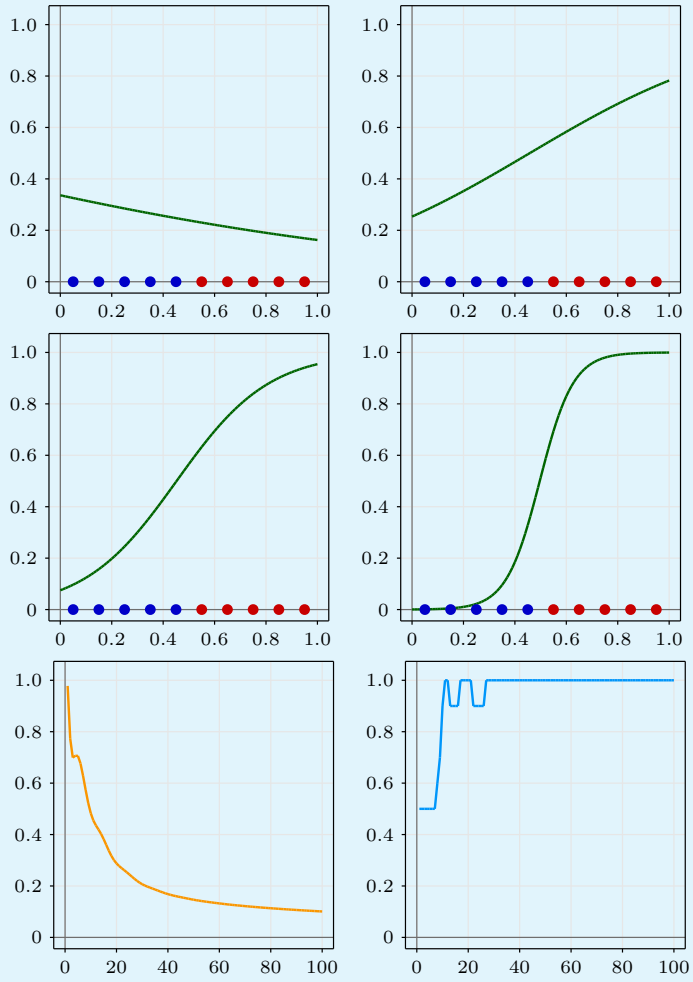
Primer 11.10. *Za model logističke regresije u slučaju jednodimenzionalnih podataka (proširenih jedinicom) važi* $f_{\mathbf{w}}((1,x)) > 0.5$ *ako važi* $w_1 + w_2 x > 0$*, a* $f_{\mathbf{w}}((1,x)) < 0.5$ *ako važi* $w_1 + w_2 x < 0$*. Dakle, u slučaju jednodimenzionalnih podataka logistička regresija pronalazi najbolje vrednosti za* w_1 *i* w_2*, te kao najbolju hiperravan daje tačku* $-w_1/w_2$*. U slučaju linearno razdvojivih podataka, takva tačka (za pogodne parametre) razdvaja dve klase.*

Niz od sledeće četiri slike prikazuje jednostavan primer treninga modela logističke regresije za linearno razdvojiv problem pri čemu se minimizacija greške vrši gradijentnim spustom. Dato je 10 *tačaka od kojih* 5 *plavih pripada klasi* 0*, a* 5 *crvenih klasi* 1*. Slike prikazuju model na početku treninga, nakon 10, nakon 20 i nakon 100 iteracija. Primećuje se kako model najpre iz opadajuće prelazi u rastuću krivu, a potom u okolini tačke* $x = 0.5$ *postaje sve strmiji. Trening gradijentnim spustom bi se mogao nastaviti, pri čemu bi kriva postajala sve strmija. Kako sigmoidna kriva ne dostiže vrednosti* 0 *i* 1*, broj koraka takvog treninga nije ograničen. Sve veća strmost modela i približavanje njegovih predviđanja vrednostima* 0 *i* 1 *odražava sve veću sigurnost modela u svoja predviđanja. Na petoj slici narandžasta kriva predstavlja grešku koja se optimizuje (za* $\lambda = 0$*), a na šestoj slici svetlo-plava kriva predstavlja tačnost modela (udeo tačnih predviđanja). Prime-*

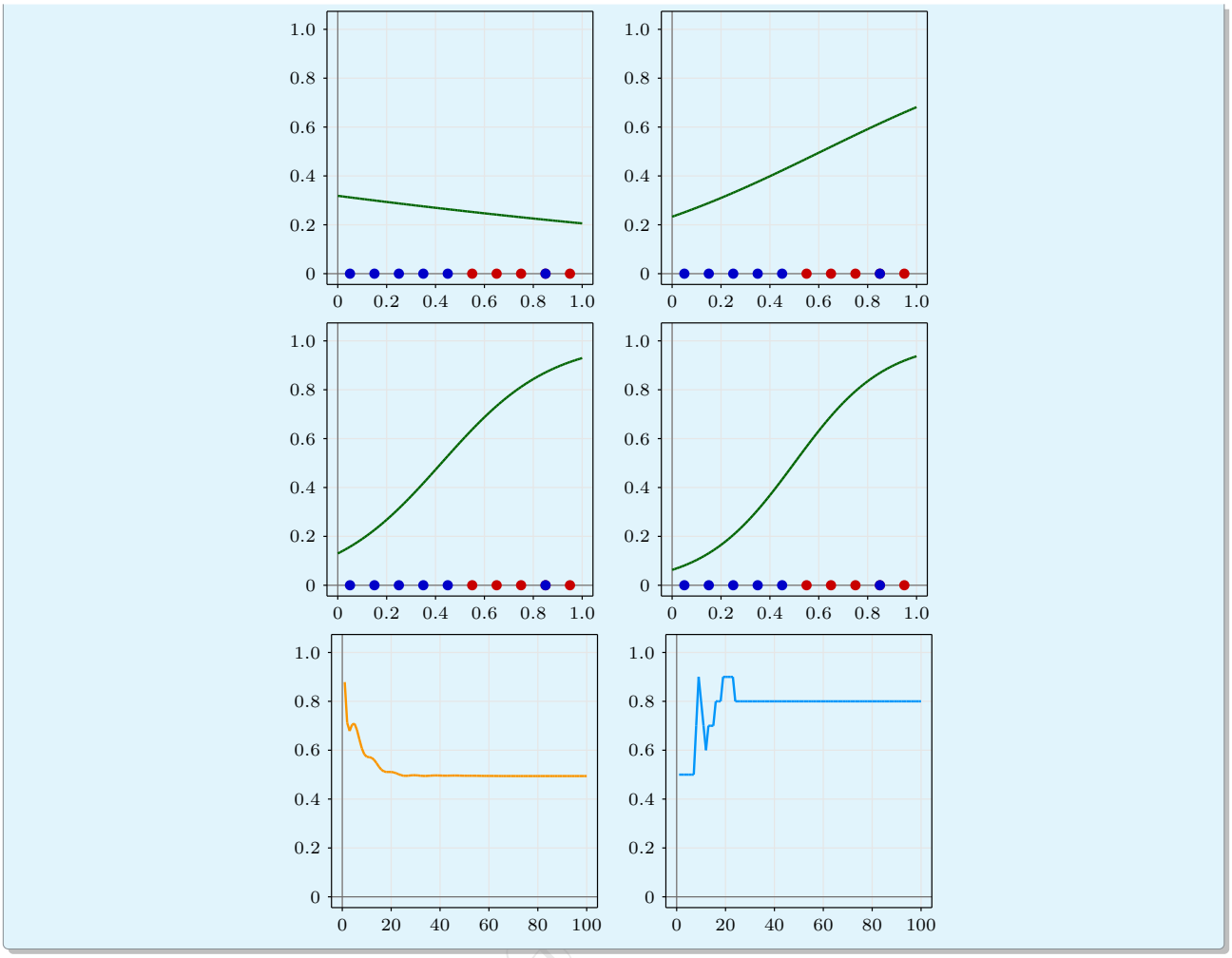
[4] Proizvod velikog broja vrednosti između 0 i 1 lako može postati 0 usled potkoračenja.

[5] Ova funkcija greške može se uopštiti i na klasifikacione probleme koji nisu binarni. Više o tome biće reči u kontekstu neuronskih mreža.

timo da tačnost nije monotona u intervalu u kom greška jeste. Ovaj fenomen će biti detaljnije analiziran u nastavku.



Sledeće slike prikazuju primer sličan prethodnom, ali za podatke koji nisu linearno razdvojivi. Iako tok treninga deluje slično prethodnom primeru, postoje bitne razlike. Pre svega zbog nerazdvojivosti klasa, model ne može neograničeno postajati strmiji, jer bi time sa previsokom sigurnošću pridruživao pogrešnu klasu plavoj tački sa desne strane, što se značajno odražava na povećanje funkcije greške. Zbog toga se model prikazan na četvrtoj slici daljim treningom ne bi značajno menjao. Druga zanimljiva razlika tiče se toga da je finalna tačnost modela 0.8*. Ovo je neobično zato što postoji tačka na x osi koja bi razdvojila klase sa tačnošću* 0.9 *(na primer, tačka $x = 0.5$). Štaviše, model prikazan na trećoj slici ima tačnost* 0.9*, ali ona u kasnijim iteracijama pada na vrednost* 0.8 *i na njoj ostaje do kraja treninga. Razlog za ovu neintuitivnu pojavu je činjenica da se pri treningu logističke regresije minimizuje ranije definisana funkcija greške, a ne neposredno broj pogrešnih predviđanja. Broj pogrešnih predviđanja se i ne može minimizovati gradijentnim metodama, jer je ta vrednost nediferencijabilna u nekim tačkama (tada gradijent nije definisan) i deo po deo konstantna (u takvim tačkama je gradijent jednak nuli). Optimizacijom funkcije greške koja ima pogodnija tehnička svojstva, kao što je slučaj u logističkoj regresiji, dobija se koristan model, ali ne optimalan u smislu tačnosti, tj. udela tačnih predviđanja. U datom primeru, na trećoj slici najlevlja crvena tačka je klasifikovana ispravno, dok na četvrtoj slici nije. Promena u vrednosti modela u toj tački nije velika i ne odražava se značajno na povećanje greške (iako je dovoljno velika da promeni klasifikaciju tačke). S druge strane, greška je smanjena za četiri plave tačke sa leve strane, čime je ukupna greška smanjena uprkos smanjenju tačnosti.*



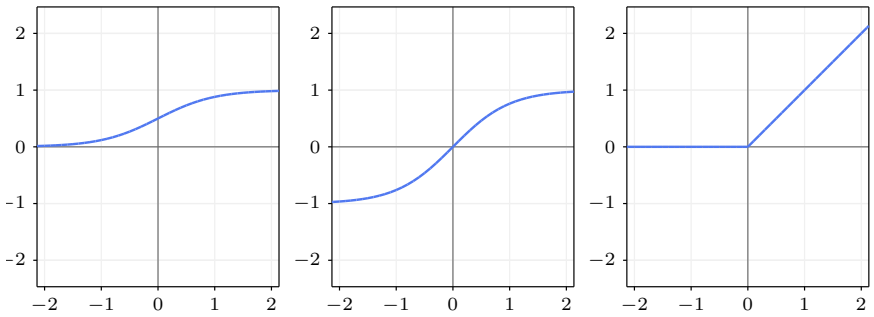
11.5 Neuronske mreže

Neuronske mreže predstavljaju jedan od najstarijih i najuspešnijih pristupa mašinskom učenju. Datiraju iz 50-ih godina dvadesetog veka i više puta su u svojoj istoriji bile u prvom planu istraživanja, a potom potiskivane od strane drugih metoda. Njihov tekući uspon počinje 2006. godine sa pojavom takozvanih *dubokih neuronskih mreža*, a novi zalet dobijaju 2012. godine kada duboke mreže po prvi put značajno nadilaze postojeće metode u problemima *računarskog vida* (eng. *computer vision*). Od tada su neuronske mreže postigle velike uspehe i u mnogim drugim oblastima, primarno u obradi prirodnog jezika, obradi govora, igranju igara, itd. U nekim problemima, prevazišle su i mogućnosti ljudskih eksperata. Trenutno predstavljaju metod mašinskog učenja sa najširim spektrom primena.

Postoje različite vrste neuronskih mreža, često specijalizovanih za konkretne primene. Ipak, sve su prepoznatljive po svojoj kompozitnoj strukturi — svaka mreža sastoji se od određenog broja elementarnih jedinica koje nazivamo neuronima. Neuroni, po gruboj analogiji sa biološkim neuronima u mozgu, jedni drugima prosleđuju signale i izračunavaju nove signale na osnovu onih koji su im prosleđeni. Tačna struktura povezanosti neurona i način na koji oni vrše izračunavanja čine takozvanu *arhitekturu mreže* i precizno određuju o kakvoj mreži se radi. Arhitekturu mreže definiše ekspert, imajući u vidu specifičnosti problema koji se rešava. U nastavku će najpre biti opisana struktura pojedinačnih neurona, a potom kako se oni organizuju u različite fundamentalne arhitekture. Pri kraju ove glave biće diskutovane smernice vezane za njihovu primenu i biće opisano kako se neuronske mreže primenjuju u različitim praktičnim problemima.

11.5.1 Neuron

Neuroni su osnovne jedinice izračunavanja čijim se povezivanjem grade neuronske mreže. Po uzoru na biološke neurone, oni kao ulaze primaju signale od drugih neurona i transformišu ih na neki način čime proizvode svoj izlazni signal. Svi signali predstavljeni su realnim brojevima, a njihova obrada vrši se nekom jednostavnom matematičkom funkcijom koja obično uključuje linearnu kombinaciju ulaznih signala, a potom nelinearnu transformaciju te linearne kombinacije.



Slika 11.4: Sigmoidna funkcija, hiperbolički tangens i ispravljena linearna jedinica.

Definicija 11.9 (Neuron)**.** Neuron *je funkcija* n *argumenata* $x_1, \ldots, x_n$ *koja ima opšti oblik:*

$$g\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

gde su $b, w_1, \ldots, w_n$ parametri *ili* težine *neurona, kojima se vrši linearno kombinovanje ulaza, a* $g : \mathbb{R} \to \mathbb{R}$ *je nelinearna* aktivaciona funkcija *kojom se ta linearna kombinacija transformiše.*

U definiciji se postavlja uslov da je aktivaciona funkcija nelinearna, jer se linearnom transformacijom ne bi mogla dobiti dodatna izražajnost u odnosu na ulaznu linearnu kombinaciju. Naime, linearna kombinacija različitih linearnih kombinacija ulaznih promenljivih je i dalje linearna kombinacija ulaznih promenljivih.

Aktivaciona funkcija nije jednoznačno određena i postoji više mogućih izbora pri definisanju neuronske mreže. Aktivacione funkcije najčešće su monotone, neprekidne i skoro svuda diferencijabilne. Neki od često korišćenih izbora su sigmoidna funkcija:

$$\sigma(x) = \frac{1}{1 + e^{-x}} ,$$

hiperbolički tangens:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} ,$$

ispravljena linearna jedinica (eng. *rectified linear unit* – ReLU):

$$\mathrm{ReLU}(x) = \max(0, x)$$

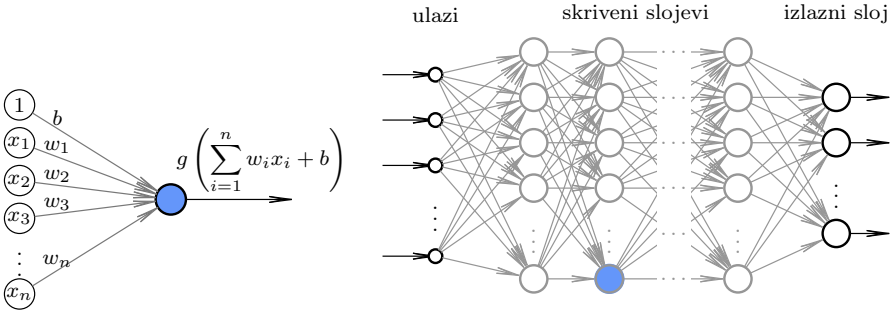
i njene varijacije. Grafici ovih funkcija prikazani su na slici 11.4. U skorije vreme najčešće se u svim slojevima mreže osim u poslednjem koristi neka varijanta ispravljene linearne jedinice zbog njenih poželjnih tehničkih svojstava.

Zbog monotonosti aktivacione funkcije, što je vrednost linearne kombinacije veća, to je izlazni signal jači. Očito, visoke težine nekih ulaza vode jakom uticaju tih ulaza na izlazni signal, a vrednosti bliske nuli umanjuju uticaj tih ulaza. Znak težine određuje da li neki ulaz potkrepljuje ili inhibira ispoljavanje signala.

Može se primetiti da, u slučaju da se za aktivacionu funkciju izabere sigmoidna funkcija, neuron predstavlja model logističke regresije, dok u slučaju da se za aktivacionu funkciju izabere identititet, neuron predstavlja model linearne regresije. Zato neuronske mreže predstavljaju uopštenja ovih modela, a takođe su sposobne da vrše klasifikaciju i regresiju.

11.5.2 Potpuno povezane neuronske mreže

Potpuno povezane neuronske mreže (eng. *fully connected neural networks*) predstavljaju najstariji od trenutno korišćenih modela neuronskih mreža. Naziv su dobile po strukturi povezanosti neurona – neuroni se slažu u slojeve, grupe neurona, i povezuju se tako što svi neuroni jednog sloja kao ulaze primaju vrednosti svih neurona prethodnog sloja, a svoje vrednosti prosleđuju svim neuronima narednog sloja. Arhitektura potpuno povezane neuronske mreže prikazana je na slici 11.5. Na njoj su prikazani ulazi, koji ne predstavljaju neurone, već numeričke vrednosti, potom slojevi neurona koji se nazivaju *skrivenim* jer se njihove vrednosti koriste isključivo unutar modela i jedan sloj koji se naziva *izlaznim*, pošto daje predviđanja koja model izračunava. Neuronske mreže obično imaju barem jedan skriveni sloj. U slučaju da neuronska mreža nema skrivenih slojeva, ona se



Slika 11.5: Struktura neurona i arhitektura potpuno povezane neuronske mreže.

svodi na linearni model (poput linearne i logističke regresije). Moderne neuronske mreže često imaju i desetine skrivenih slojeva i tada se često o njima govori kao o *dubokim neuronskim mrežama* (eng. *deep neural networks*). Taj izraz, međutim, nema preciznu definiciju, te poimanje koje su mreže duboke a koje nisu zavisi od konteksta, pa i od vremena u kojem se o mrežama govori.

Mreža sa L slojeva može se precizno definisati na sledeći način.[6]

Definicija 11.10 (Potpuno povezana mreža)**.** Potpuno povezana neuronska mreža *je funkcija* $f_{\mathbf{w}}$ *definisana na sledeći način:*

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{x} \\ \mathbf{h}_i &= g(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i), \quad i = 1, \ldots, L-1 \\ f_{\mathbf{w}}(\mathbf{x}) &= \tilde{g}(\mathbf{W}_L \mathbf{h}_{L-1} + \mathbf{b}_L) \end{aligned}$$

gde $\mathbf{x}$ *predstavlja ulazni vektor,* L *je broj slojeva,* $\mathbf{h}_i$ *za* $i = 1, \ldots, L-1$ *predstavlja vektor vrednosti* i*-tog sloja,* $\mathbf{W}_i$ *je matrica parametara čije vrste predstavljaju vektore parametara neurona* i*-tog sloja koji množe ulazne vrednosti,* $\mathbf{b}_i$ *je vektor slobodnih koeficijenata tih neurona, a* g *i* $\tilde{g}$ *su aktivacione funkcije. Matrice* $\mathbf{W}_i$ *i vektori* $\mathbf{b}_i$ *čine parametre* $\mathbf{w}$*.*

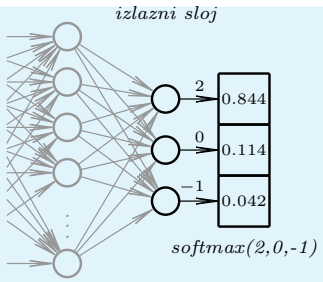
U navedenoj definiciji, pod primenom aktivacione funkcije g na vektor, podrazumeva se primena funkcije g na svaku koordinatu vektora pojedinačno. Očito, izračunavanje koje mreža vrši sastoji se od linearnih matričnih transformacija i primena aktivacionih funkcija. Funkcija $\tilde{g}$ može predstavljati drugačiju aktivacionu funkciju od funkcije g, što je i uobičajeno za poslednji sloj mreže. Izbor funkcije $\tilde{g}$ zavisi od toga u koju svrhu se mreža koristi. U slučaju regresije, obično se koristi identička funkcija, a poslednji sloj ima samo jedan neuron čija vrednost predstavlja ocenu ciljne promenljive. U ovom slučaju, mreža se može razumeti kao linearna regresija nad izlazima pretposlednjeg sloja mreže, a svi slojevi zaključno sa njim se mogu razumeti kao mehanizam konstrukcije novih svojstava iz ulaznih podataka. U slučaju klasifikacije, podrazumeva se da poslednji sloj mreže ima onoliko neurona koliko ima klasa i da se instanca klasifikuje u klasu koja odgovara neuronu koji ima najvišu vrednost. U tom slučaju, na poslednjem sloju se, kao funkcija $\tilde{g}$, koristi sledeća vektorska aktivaciona funkcija:

$$\text{softmax}(a_1, \ldots, a_m) = \left(\frac{e^{a_1}}{\sum_{i=1}^{m} e^{a_i}}, \ldots, \frac{e^{a_m}}{\sum_{i=1}^{m} e^{a_i}} \right)$$

Kako su koordinate ovog vektora nenegativne i imaju zbir 1, ovaj vektor se može interpretirati kao raspodela verovatnoće nad m različitih klasa.

Primer 11.11. *Neka je potrebno uraditi klasifikaciju neke instance u jednu od tri moguće klase. Tada neuronska mreža ima tri izlazna neurona od kojih svaki izračunava linearnu kombinaciju svojih ulaza i odgovarajući element funkcije softmax. Neka su vrednosti tih linearnih kombinacija recimo* $a_1 = 2$*,* $a_2 = 0$ *i* $a_3 = -1$*. Na njih se primenjuje funkcija softmax. Eksponencijalne vrednosti su onda* $e^{a_1} = 7.389$*,* $e^{a_2} = 1.000$ *i* $e^{a_3} = 0.368$*. Normiranjem ovih vrednosti njihovom sumom dobija se raspodela verovatnoće po klasama* $(0.844, 0.114, 0.042)$*. Na osnovu toga, najverovatnije je da data instanca pripada prvoj klasi, a izračunate verovatnoće daju i procenu pouzdanosti tog izbora.*

[6] Data definicija predstavlja najčešći slučaj. Ipak, može se na razne načine odstupiti od ove definicije ukoliko za to u konkretnom problemu postoji potreba. Na primer, aktivacione funkcije ne moraju biti iste za sve slojeve, struktura povezanosti ne mora uvek biti takva da svi neuroni jednog sloja prosleđuju svoje izlaze baš svim neuronima sledećeg sloja, itd.



Kako bi se neuronska mreža trenirala da obavlja neki zadatak, potrebno je definisati i funkciju greške, a potom obezbediti algoritam učenja. Funkcije greške mogu varirati od primene do primene, ali postoje neki uobičajeni izbori. U slučaju regresije, to je obično kvadratna greška:

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = (f_{\mathbf{w}}(\mathbf{x}) - y)^2$$

U slučaju klasifikacije u m klasa, potrebno je prvo definisati pogodnu reprezentaciju kategoričke promenljive. Najčešće korišćena reprezentacija pretpostavlja da je klasa definisana vektorom $\mathbf{y}$ dimenzije m, koji se sastoji od $m-1$ nula i jedne jedinice, pri čemu je jedinica na poziciji j ako i samo ako vektor $\mathbf{y}$ predstavlja klasu j. Pri toj reprezentaciji, svakoj instanci pridružen je jedan takav vektor $\mathbf{y}$ u skladu sa klasom kojoj instanca pripada.[7] Najčešće korišćena funkcija greške za kategoričke ciljne promenljive koja se oslanja na tu reprezentaciju je *unakrsna entropija*. Ona se definiše na sledeći način.

Definicija 11.11 (Unakrsna entropija)**.** Unakrsna entropija *je sledeća funkcija greške:*

$$L(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = -\sum_{i=1}^{m} y_i \log f_{\mathbf{w}}^{(i)}(\mathbf{x})$$

gde $f_{\mathbf{w}}^{(i)}$ *označava* i*-tu koordinatu vektorske funkcije* $f_{\mathbf{w}}$*, tj. procenjenu verovatnoću klase* i*.*

Ova funkcija može se izvesti iz statističkog principa maksimalne verodostojnosti, analogno izvođenju koje je dato u slučaju binarne unakrsne entropije za logističku regresiju. Štaviše, za pogodnu reprezentaciju klasa, binarna unakrsna entropija (definicija 11.8) je specijalan slučaj unakrsne entropije. Nije teško intuitivno razumeti ovu funkciju. Naime, ukoliko je verovatnoća stvarne klase i visoka, logaritam vrednosti $f_{\mathbf{w}}^{(i)}(\mathbf{x})$ je blizak nuli i doprinos sumi koji mu odgovara je mali. Ostali sabirci jednaki su nuli jer su sve vrednosti vektora $\mathbf{y}$ osim y_i jednake nuli (podrazumeva se da važi $0 \log 0 = 0$). Dakle, kada model ispravno proceni verovatnoće klasa, greška je mala. Ukoliko model ispravnoj klasi daje verovatnoću blisku nuli, negativna vrednost logaritma te verovatnoće je velika, množi se vrednošću y_i koja je za pravu klasu jednaka 1, dok su ostali sabirci kao i pre jednaki nuli, pa je ukupna suma velika. Ovo je upravo ponašanje koje se očekuje od dobre funkcije greške.

Kada je definisana greška modela, treba je minimizovati po parametrima modela. Ovo se u slučaju neuronskih mreža uvek radi nekom od gradijentnih metoda optimizacije koje obično predstavljaju modifikacije gradijentnog spusta. Algoritam za izračunavanje gradijenta u slučaju neuronskih mreža naziva se algoritmom *propagacije unazad* (eng. *backpropagation*).[8] Kako je mreža sastavljena od linearnih matričnih transformacija i primena aktivacionih funkcija, algoritam za izračunavanje gradijenta se sastoji prosto od množenja izvoda tih aktivacionih funkcija i odgovarajućih matrica u skladu sa pravilom izvoda složene funkcije. Zato neće biti reči o detaljima ovog algoritma. Iz prethodnog grubog opisa može se naslutiti jedan opšti problem dubokih neuronskih mreža, a to je da u slučaju njihove velike dubine dolazi do velikog broja pomenutih množenja. Zato koordinate gradijenta često bivaju bliske nuli ili ogromne po apsolutnoj vrednosti, što vodi ili previše sporoj konvergenciji gradijentog spusta ili velikoj nestabilnosti ovog postupka. Postoje načini da se takvo ponašanje ublaži, ali se time nećemo baviti.

Primetimo da je u slučaju binarne klasifikacije umesto dva izlaza i funkcije softmax dovoljno koristiti jedan izlaz i sigmoidnu funkciju. Takva varijanta arhitekture može se razumeti kao logistička regresija nad pretposlednjim slojem mreže. Ilustrujmo primerom da je neuronska mreža moćnija od logističke regresije.[9]

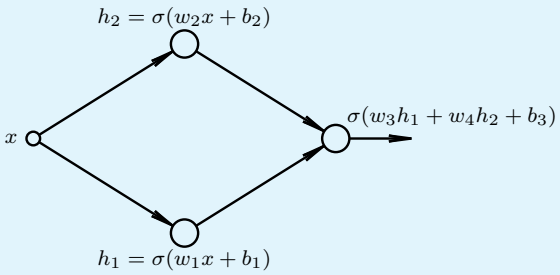
[7] Ovo je, kod neuronskih mreža, uobičajena reprezentacija kategoričkih promenljivih sa više od dve vrednosti (i svojstava i ciljnih promenljivih). Ukoliko promenljiva ima samo dve kategorije, onda se ona može predstaviti samo jednom promenljivom sa dve moguće vrednosti (na primer, 0 i 1).

[8] Treba imati u vidu da neuronske mreže najčešće predstavljaju diferencijabilne funkcije, jer najčešće nastaju komponovanjem diferencijabilnih funkcija. Nekada se koriste nediferencijabilne aktivacione funkcije poput ReLU. Ipak, u takvim slučajevima skup tačaka nediferencijabilnosti je mere nula, pa ne predstavlja problem u praksi.

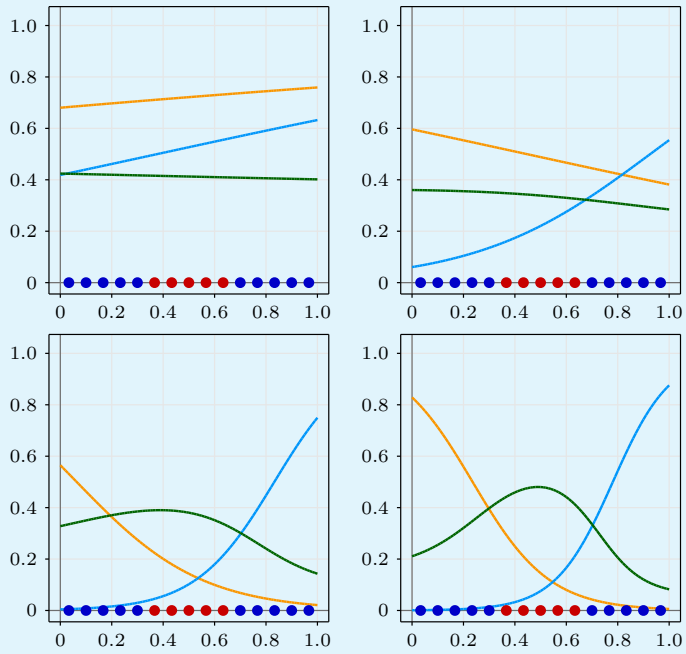
[9] Ovaj primer inspirisan je diskusijom iz literature [20].

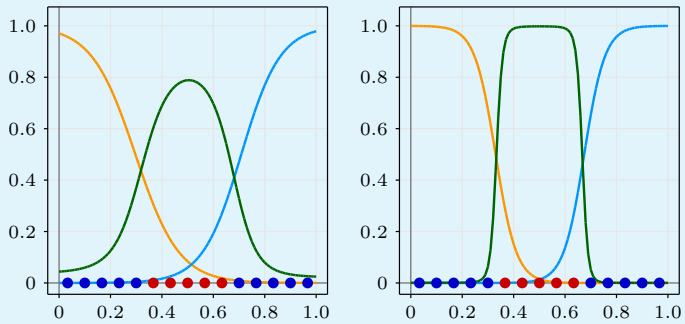
Primer 11.12. *Logistička regresija pogodna je za probleme nalik onom ilustrovanom na slici 11.3. Čak i ako ima nekog preklapanja klasa, ako postoje dve grupe koje se u velikoj meri mogu razdvojiti pomoću hiperavni, logistička regresija će naći neku od takvih hiperravni (videti primer 11.10). Postoje, međutim, klasifikacioni problemi za koje je logistička regresija suštinski nemoćna, dok neuronska mreža nije. Takve probleme i razliku između logističke regresije i neuronske mreže ilustrovaćemo na jednom primeru jednostavnih jednodimenzionalnih podataka.*

Pretpostavimo da su dati brojevi predstavljeni na narednima slikama tačkama na x osi: plave tačke pripadaju jednoj klasi, a crvene drugoj. Pošto se radi o binarnoj klasifikaciji, ove klase možemo, kao u slučaju logističke regresije, označiti 0 *(u slučaju plave) i* 1 *(u slučaju crvene). Plave i crvene tačke ne mogu se razdvojiti jednom hiperravni (što je u jednoj dimenziji tačka), pa je logistička regresija nemoćna tj. ne može ih sve klasifikovati ispravno. S druge strane, neuronska mreža može da nauči kako da razdvoji klase. U nastavku je opisano treniranje neuronske mreže koja ima jedan skriveni sloj sa dva neurona (definisana sa po dva parametra) i jedan neuron u izlaznom sloju (definisan sa tri parametra). Svaki od tih neurona ima sigmoidnu aktivacionu funkciju. Izlazni neuron, kao u slučaju logističke regresije, izračunava verovatnoću da instanca pripada jednoj izabranoj klasi, u ovom slučaju crvenoj. Opisana mreža prikazana je na narednoj slici.*



Kao funkcija greške, kao u slučaju logističke regresije, koristi se unakrsna entropija i ona se minimizuje gradijentnim spustom. Slike u nastavku ilustruju kako neuronska mreža konvergira ka modelu koji tačno razdvaja date klase (na početku treninga, nakon 140, nakon 160 i nakon 400 iteracija optimizacije). Narandžasta i svetlo-plava kriva odgovaraju neuronima u skrivenom sloju. Zelena kriva predstavlja verovatnoću da tačka x pripada crvenoj klasi. U poslednjoj iteraciji, dve tačke na x osi za koje zelena kriva ima vrednost 0.5 *razdvajaju crvene tačke od plavih. Zanimljivo je analizirati kako neuronska mreža razdvaja ove dve klase.*





Svaki od neurona u skrivenom sloju uči neku transformaciju podataka. U poslednjoj iteraciji, narandžasta kriva, definisana transformacijom $\sigma(-24.55x + 8.09)$, uzima vrednosti veće od 0.5 na levoj grupi plavih tačaka, a vrednosti manje od 0.5 u slučaju ostalih tačaka. Svetlo-plava kriva, definisana transformacijom $\sigma(24.78x - 16.72)$, uzima vrednosti veće od 0.5 na desnoj grupi plavih tačaka, a vrednosti manje od 0.5 u slučaju ostalih tačaka. Naravno, nijedan od ova dva neurona pojedinačno ne razdvaja klase ispravno. Ipak, ako se dve vrednosti koje ti neuroni izračunavaju daju kao ulaz dodatnom, izlaznom neuronu, on može da dâ visoku izlaznu vrednost ako skriveni neuroni daju niske vrednosti, a nisku ako oni daju visoke vrednosti. Konkretno, vrednost izlaznog neurona se u tački x izračunava izrazom $\sigma(-13.81h_1 - 14.59h_2 + 6.62)$, gde je h_1 izlazna vrednost prvog neurona, a h_2 izlazna vrednost drugog neurona u tački x. Mreža dostiže date vrednosti parametara pri minimizaciji greške na ovom skupu podataka u četiristotoj iteraciji.

Imajući u vidu da je broj ulaza fiksiran, potpuno povezane neuronske mreže su specijalizovane za primenu nad podacima predstavljenim u vidu vektora svojstava fiksne dužine. Zbog toga se potpuno povezane neuronske mreže ne mogu jednostavno primeniti na neke vrste podataka, poput slika (koje mogu biti različitih dimenzija), rečenica prirodnog jezika, bioinformatičkih sekvenci i vremenskih serija koje prirodno imaju različite dužine. Ipak, korisne su u slučaju modelovanja tabelarnih podataka ili, često, kao komponenta drugih vrsta neuronskih mreža.

Definisanje arhitekture neuronske mreže je netrivijalan posao. Pogodan broj slojeva i neurona u njima u velikoj meri zavisi od konkretnog problema. U mnogim problemima sa tabelarnim podacima, dva skrivena sloja mogu dati zadovoljavajuće rezultate. S druge strane, u nekim primenama potrebno je i više desetina slojeva (što je karakteristično za konvolutivne mreže, ali ne i za potpuno povezane). Uspešno dizajniranje mreže u velikoj meri se oslanja na iskustvo, kao i na višestruke pokušaje radi izbora pogodnih vrednosti hiperparametara.

11.5.3 Konvolutivne neuronske mreže

Zahvaljujući konvolutivnim neuronskim mrežama, ostvareni su najveći pomaci u praktičnim primenama mašinskog učenja, pre svega u oblasti računarskog vida. Razmotrimo najpre motivaciju za njihovo uvođenje.

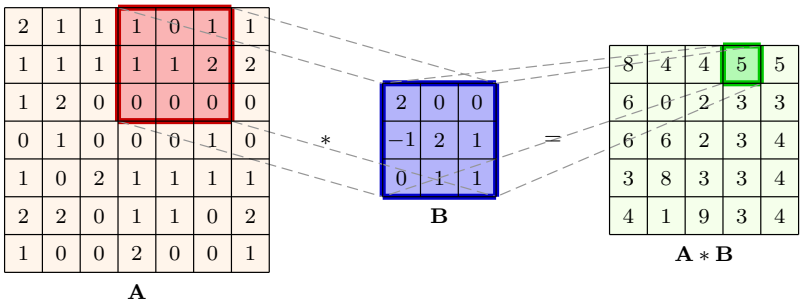
Mašinsko učenje može se koristiti nad slikama za raznovrsne zadatke, na primer, za određivanje da li se na slici nalazi zgrada, da li se nalazi semafor, da li je na slici mačka ili pas, da li je slika ispravno orijentisana, itd. Jedan pristup za rešavanje takvih problema može biti da se najpre primene vešto osmišljene transformacije koje od slike proizvode novu sliku istih ili sličnih dimenzija. Te transformacije mogu biti fokusirane na različite aspekte slike i mogu služiti da se slika ili njen deo izoštri, zatamni, da joj se pojača kontrast, da se istaknu ivice, itd. Izlazi takvih transformacija – neke nove slike – onda mogu da se koriste kao svojstva na osnovu kojih bi model (poput logističke regresije ili potpuno povezane mreže) mogao da uči. Na primer, ako je potrebno detektovati da li se na slici nalazi zgrada, onda bi lokacije vertikalnih i horizontalnih ivica verovatno predstavljale vrlo korisnu informaciju. Transformacije koje na slici izdvajaju vertikalne i horizontalne ivice, kao i mnoge druge slične, odavno su dizajnirane (od strane čoveka) u okviru klasične obrade signala i obrade slika. Raznovrsne transformacije slika mogu se vršiti primenom operacije *konvolucije* dve matrice – jedna matrica predstavlja ulazni signal (tj. sliku), a druga matrica, *filter* (eng. *filter, kernel*), određuje sâmu obradu koja se vrši.

Definicija 11.12 (Konvolucija matrica)**.** Konvolucija matrica $\mathbf{A} \in \mathbb{R}^{m\times n}$ *i* $\mathbf{B} \in \mathbb{R}^{p\times q}$ *je matrica dimenzija* $(m-p+1)\times(n-q+1)$ *koja se označava* $\mathbf{A} * \mathbf{B}$ *i definiše na sledeći način:*

$$(\mathbf{A} * \mathbf{B})_{i,j} = \sum_{k=0}^{p-1}\sum_{l=0}^{q-1} \mathbf{A}_{i+k,j+l}\mathbf{B}_{k,l} \qquad i = 0,\ldots,m-p,\ \ j = 0,\ldots,n-q$$

Strogo govoreći, operacija navedena u prethodnoj definiciji nije konvolucija, već *unakrsna korelacija* (eng. *cross-*

correlation), ali je razlika u slučaju neuronskih mreža nepostojeća i u praksi se operacija konvolucije implementira na prikazani način.



Slika 11.6: Ilustracija konvolucije: element rezultujuće matrice u zelenom polju dobija se kao skalarni proizvod crveno obojenog dela matrice **A** i matrice **B**. Ostatak rezultujuće matrice dobija se analogno. U opštem slučaju matrice ne moraju biti kvadratne.

Dakle, konvolucija za svako i i j izračunava skalarni proizvod elemenata matrice **B** sa odgovarajućim elementima dela matrice **A** koji je istih dimenzija kao matrica **B**.[10] Može se zamisliti da se matrica **B** pomera duž matrice **A** i računa skalarne proizvode na različitim lokacijama i tako daje rezultat konvolucije (slika 11.6).

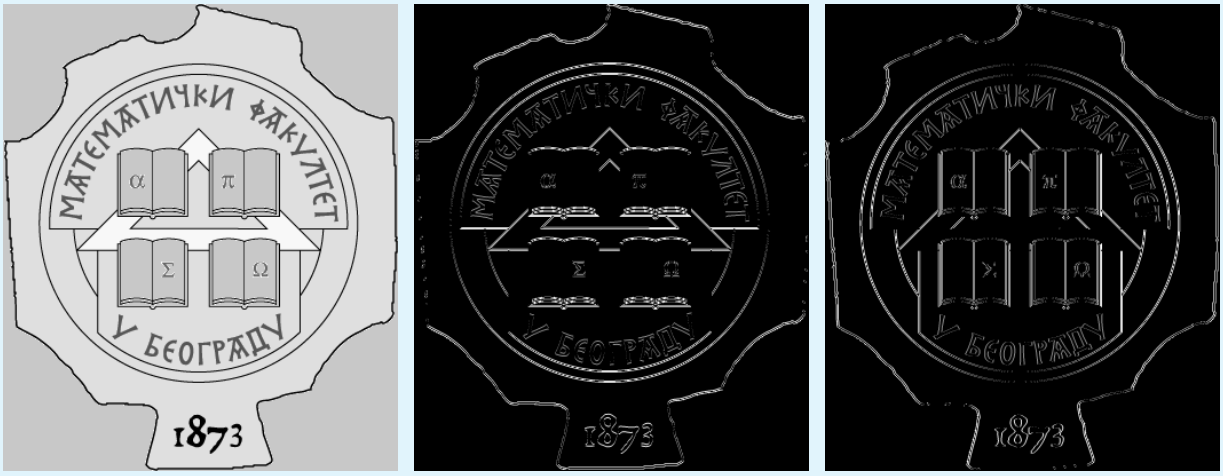
Primer 11.13. *Razmotrimo zadatak određivanja horizontalnih i vertikalnih ivica na slici. Ivice su granice između predmeta ili oblasti na slici i najčešće predstavljaju nagle promene boje. Pretpostavimo da se obrađuje rasterizovana slika (tj. slika opisana pikselima) u nijansama jedne boje, data u vidu matrice. Horizontalne ivice mogu biti predstavljene razlikama vrednosti piksela i piksela ispod njih. Ako je ta razlika velika, velike su i šanse da se na tom pikselu nalazi neka ivica. Analogno, vertikalne ivice mogu biti predstavljene razlikama vrednosti piksela i piksela desno od njih. Ako je slika data u vidu matrice* $\mathbf{A} \in \mathbb{R}^{m\times n}$*, sve opisane razlike vrednosti piksela i vrednosti piksela ispod njih (tj. sve horizontalne ivice), mogu se primenom konvolucije dobiti na sledeći način:*

$$\mathbf{A} * \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

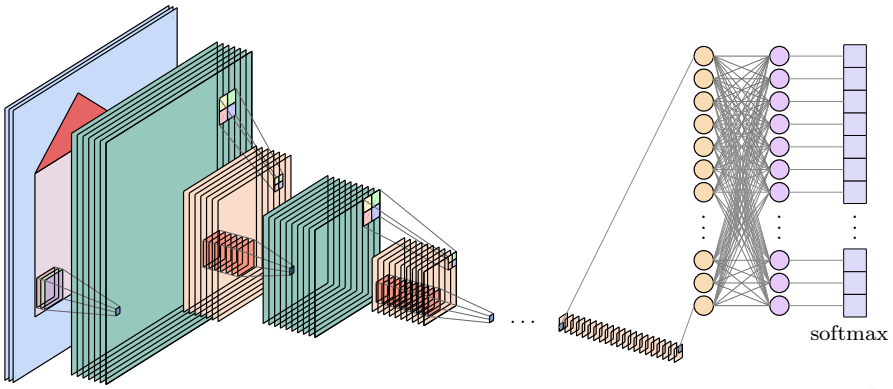
a razlike vrednosti piksela i vrednosti piksela desno (tj. sve vertikalne ivice) mogu se dobiti na sličan način:

$$\mathbf{A} * [1 \ \ -1]\,.$$

Naredna slika prikazuje izvornu sliku i izračunate horizontalne i vertikalne ivice (visoke vrednosti koje odgovaraju velikim razlikama u osvetljenju susednih piskela, predstavljene su belom bojom).



[10] Iako obično razmišljamo o skalarnom proizvodu vektora, nema ničega problematičnog u ideji skalarnog množenja dve matrice – sabiranjem proizvoda elemenata na odgovarajućim lokacijama.



Slika 11.7: Shema konvolutivne neuronske mreže. Konvolutivni slojevi su prikazani zelenom bojom, a slojevi agregacije narandžastom. Na desnoj strani je prikazana potpuno povezana mreža bez skrivenih slojeva kojom se konvolutivna mreža završava. Prvi, narandžasti element predstavlja preoblikovanu formu poslednjeg izlaznog tenzora u oblik u kojem potpuno povezana mreža može da ga prihvati kao svoj ulaz. Drugi, ljubičasti element predstavlja rezultat linearne transformacije nad narandžastim delom, a poslednji element predstavlja rezultat primene funkcije softmax nad prethodnim.

Korišćene matrice dimenzija 1×2 *i* 2×1 *su primeri filtera. Suštinski, filteri opisuju obrasce koje „traže“ na slikama – u delovima slike koji sadrže te obrasce, vrednosti skalarnog proizvoda su velike.*

Pristup za korišćenje mašinskog učenja nad slikama koji bi se zasnivao na ovakvim transformacijama slike, iako moguć, nije pogodan u praksi jer zahteva ekspertsko znanje o slikama i veliku inventivnost u izboru i dizajniranju relevantnih transformacija, a moguće je da ni eksperti nisu u stanju da definišu dovoljno dobre filtere za neke zadatke. Alternativna ideja je da se te pogodne transformacije ne moraju unapred definisati (od strane čoveka), već da se model može definisati tako da kao svoj sastavni deo nauči i sâme te transformacije (a ne samo zaključivanje na osnovu njihovih izlaza). To je osnovna ideja konvolutivnih neuronskih mreža. Objasnimo najpre kako se koncept konvolutivnih filtera uklapa u okvire neuronskih mreža. Kao što je već rečeno, može se zamisliti da se, prilikom izračunavanja konvolucije, matrica $\mathbf{B}$ pomera duž matrice $\mathbf{A}$, pri čemu se računaju skalarni proizvodi na različitim lokacijama. Svaki od ovih skalarnih proizvoda može se izračunati u neuronu definisanom formulom[11] $g\left(\sum_{i=1}^{n} w_i x_i + b\right)$ (u skladu sa definicijom 11.9). Skup svih ovih skalarnih proizvoda, tj. rezultat operacije konvolucije, može se onda izračunati grupom neurona koji dele isti skup parametara, koji čini odgovarajući filter. Zato se može zamišljati da se ova transformacija u okviru konvolutivne mreže vrši grupom neurona koji dele parametre ili jednim neuronom koji se primenjuje na različite pozicije ulaza. Cilj je izračunati, u fazi treninga, taj skup parametara koji čini jedan traženi filter. U daljem tekstu ćemo pod terminom filter, konciznosti radi, podrazumevati bilo skup parametara, bilo neurone kojima odgovaraju ti parametri.

Načelno, konvolutivna mreža obučava se tako da minimizuje greške svojih izlaza na datim ulazima iz trening skupa, a u tom procesu uči filtere koji realizuju korisne transformacije nad ulaznom slikom. Iako je iz prethodnog razmatranja jasno da konvolutivni filteri, u principu, mogu biti naučeni, i dalje je potrebno definisati arhitekturu mreže koja to može uspešno da radi. Shema jedne arhitekture konvolutivne mreže prikazana je na slici 11.7. Na njoj se vidi da se ulaz transformiše nizom konvolucija, aktivacionih funkcija i agregacija koji se međusobno prepliću, a da se na kraju nalazi potpuno povezana mreža koja, u ovom primeru, vrši finalnu klasifikaciju ulaza. U nastavku ćemo detaljnije opisati ovu strukturu, uključujući i novi koncept agregacije.

U mnogim situacijama, kao u obradi slika, potrebno je imati više ulaznih matrica (na primer, po jednu za crvenu, zelenu i plavu komponentu). Niz matrica iste dimenzije nazivamo *tenzorom*, a pojedinačne matrice njegovim *kanalima*. Operacija konvolucije lako se uopštava tako da joj argumenti budu tenzori, a izlaz i dalje matrica.

Definicija 11.13 (Konvolucija tenzora)**.** Konvolucija tenzora $\mathbf{A} \in \mathbb{R}^{m \times n \times c}$ *i* $\mathbf{B} \in \mathbb{R}^{p \times q \times c}$ *je matrica dimenzija* $(m - p + 1) \times (n - q + 1)$ *koja se definiše se na sledeći način:*

$$(\mathbf{A} * \mathbf{B})_{i,j} = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} \sum_{t=0}^{c-1} \mathbf{A}_{i+k,j+l,t} \mathbf{B}_{k,l,t} \qquad i = 0, \ldots, m-p,\ j = 0, \ldots, n-q$$

[11] Pri tome, elementi podmatrice matrice A i elementi matrice B tretiraju se kao vektori.

U navedenoj definiciji konvolucije tenzora pretpostavlja se da je poslednja dimenzija oba niza jednaka c.

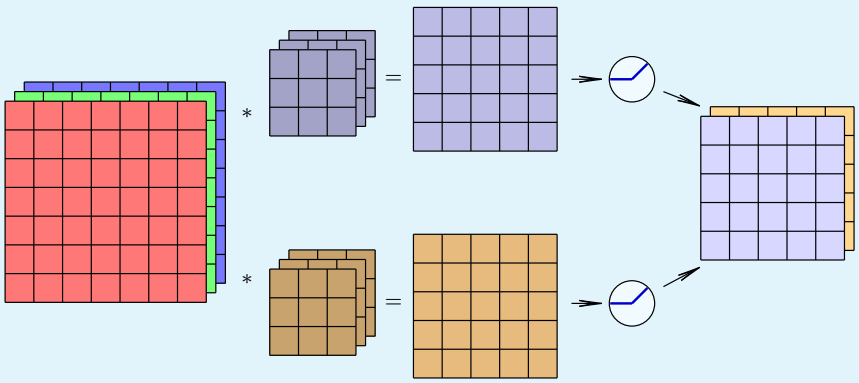
Neuronska mreža u svakom sloju uči određeni broj filtera. Skup filtera koji deluju nad istim ulazom nazivamo *konvolutivnim slojem* (eng. *convolutional layer*). On na svom izlazu daje tenzor čiji svaki kanal odgovara izlazu jednog filtera. Jedan filter može naučiti da detektuje vertikalne ivice, drugi horizontalne, treći kose i slično. Izlaz svakog konvolutivnog sloja transformiše se nelinearnom aktivacionom funkcijom, tako što se aktivaciona funkcija primeni na svaki element tenzora. Nad izlazom jednog konvolutivnog sloja moguće je izgraditi sledeći konvolutivni sloj i tako dalje. Stoga, filteri viših konvolutivnih slojeva konstruišu nova svojstva nad svojstvima koje su konstruisali niži slojevi. Na taj način oni traže složenije obrasce u slici od filtera nižeg nivoa, a koji su obrasci bitni za dati problem, uči se minimizacijom greške koju mreža pravi na podacima trening skupa.

Tenzor koji predstavlja izlaz konvolutivnog sloja formira se *slaganjem* matrica koje su rezultat konvolucije filterâ tog sloja sa izlazom prethodnog sloja. Definišimo operaciju slaganja matrica u tenzor.

Definicija 11.14 (Slaganje matrica)**.** Slaganje matrica *je operacija kojom se od* K *matrica* $A_1, \ldots, A_K$ *dimenzija* $m \times n$ *dobija tenzor* $[A_l | l = 1, \ldots, K]$ *dimenzija* $m \times n \times K$ *i definiše se na sledeći način:*

$$[A_l | l = 1, \ldots, K]_{i,j,k} = (A_k)_{i,j}, \quad i = 1, \ldots, m;\ j = 1, \ldots, n;\ k = 1, \ldots, K$$

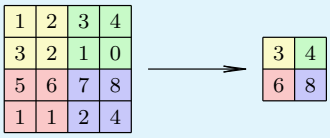
Primer 11.14. *Naredna slika prikazuje ilustraciju dejstva konvolutivnog sloja.*



Sloj kao ulaz uzima tenzor koji se u ovom primeru sastoji od tri kanala dimenzija 7×7. Sloj se sastoji od dva filtera. Primetimo da filter takođe ima treću dimenziju koja se poklapa sa brojem kanala ulaznog tenzora. Svaki od ovih filtera proizvodi jedan izlazni kanal. Kako konvolucija smanjuje dimenzije ulaznih kanala, izlazni su dimenzija 5×5. Na njih se primenjuje aktivaciona funkcija. Od dve tako dobijene matrice dimenzija 5×5, slaganjem se formira jedan izlazni tenzor dimenzija $5 \times 5 \times 2$.

Pored konvolutivnih slojeva, često se koriste i *slojevi agregacije* (eng. *pooling layer*) koji služe agregaciji informacije u cilju smanjenja količine izračunavanja i broja parametara u mreži. Obično se realizuju tako što se svaki kanal ulaznog tenzora podeli na delove određenih dimenzija koji se zamene jednom vrednošću poput maksimuma vrednosti u tom tenzoru. Konkretan izbor maksimuma je čest i intuitivan. Naime, visoke vrednosti u tenzoru znače da je filter koji je proizveo tu vrednost na nekom mestu u svom ulazu naišao na obrazac koji traži (tj. koji daje veliku vrednost skalarnog proizvoda). Uprosečavanjem bi se visoke vrednosti izgubile zbog prisustva niskih, dok maksimum čuva informaciju o tome da je traženi obrazac pronađen. Ipak, agregacijom se gubi informacija o tačnoj lokaciji maksimuma. Značaj tog gubitka zavisi od konkretne primene.

Primer 11.15. *Naredna slika prikazuje ilustraciju agregacije maksimumom dimenzija* 2×2*. Primetimo da se za razliku od konvolucije, maksimum ne računa nad preklapajućim delovima ulaznog kanala, već se on deli na susedne i disjunktne delove koji odgovaraju dimenzijama agregacije. Ukoliko ulazni sloj ima više kanala, svaki se obrađuje nezavisno.*



U ovom primeru je svaka dimenzija slike deljiva odgovarajućom dimenzijom filtera. Ukoliko to ne bi bio slučaj, određeni broj (koji odgovara ostatku pri deljenju) kolona, odnosno vrsta, bio bi zanemaren. Na primer, u slučaju da je slika dimenzija 7×8, a filter dimenzija 3×3, poslednja vrsta i dve poslednje kolone ulazne slike, biće zanemarene, a slika dobijena agregacijom biće dimenzija 2×2.

Na kraju, neretko se, kao na slici 11.7, na krajnje izlaze takve mreže nadovezuje potpuno povezana mreža koja kao svoje ulaze uzima sve vrednosti krajnjeg tenzora dobijenog iz konvolutivnog dela. Upotreba potpuno povezane mreže ipak nije neophodna i postoje arhitekture koje je ne uključuju. Na osnovu ovog razmatranja, konvolutivne mreže moguće je i formalno definisati.

Definicija 11.15 (Konvolutivna mreža)**.** Konvolutivna neuronska mreža *je funkcija* $f_{\mathbf{w}}$ *definisana na sledeći način:*

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{x} \\ \mathbf{h}_i &= a(g([\mathbf{W}_{ij} * \mathbf{h}_{i-1} + \mathbf{B}_{ij} | j = 1, \ldots, K_i])), \quad i = 1, \ldots, L \\ f_{\mathbf{w}}(\mathbf{x}) &= \tilde{g}_{\tilde{\mathbf{w}}}(\mathbf{h}_L) \end{aligned}$$

gde $\mathbf{x}$ *predstavlja ulazni tenzor,* L *je broj konvolutivnih slojeva,* K_i *je broj filtera u sloju* i*,* $\mathbf{h}_i$ *za* $i = 1, \ldots, L$ *predstavlja izlazni tenzor* i*-tog sloja,* $\mathbf{W}_{ij}$ *je tenzor parametara* j*-tog filtera* i*-tog sloja,* $\mathbf{B}_{ij}$ *matrica slobodnih koeficijenata koja po dimenzijama odgovara matrici* $\mathbf{W}_{ij} * \mathbf{h}_{i-1}$*, a je funkcija agregacije,* g *je aktivaciona funkcija,* $\tilde{g}_{\tilde{\mathbf{w}}}$ *je potpuno povezana neuronska mreža. Tenzori* $\mathbf{W}_i$*,* $\mathbf{B}_i$ *i vektor* $\tilde{\mathbf{w}}$ *čine parametre* $\mathbf{w}$*.*

Trening konvolutivnih mreža u svrhe klasifikacije i regresije algoritamski se ne razlikuje od treninga potpuno povezanih mreža. Specifičnosti vezane za neke druge vrste zadataka biće diskutovane kasnije kada bude reči o praktičnim primenama.

11.5.4 Rekurentne neuronske mreže

Mnogi zadaci mašinskog učenja vezani su za sekvencijalne podatke, tj. podatke koji se mogu predstaviti u vidu niski jednostavnih elemenata. Primeri takvih zadataka su klasifikacija tekstova prema njihovoj temi ili sentimentu (osećanju, raspoloženju, afektivnom tonu), kreiranje sažetaka tekstova, prevođenje rečenica sa jednog prirodnog jezika na drugi, predviđanje kretanja cena akcija na berzi, predviđanje budućeg stanja pacijenta na osnovu zdravstvene istorije i tako dalje. Sekvencijalni podaci najčešće su podaci promenljive dužine. Potpuno povezane mreže očekuju kao ulaze vektorske reprezentacije fiksne dimenzije i stoga su neprimerene obradi ovakvih podataka. Dodatno, redosled reči u rečenici ili redosled njenih delova nekada se može menjati bez promene smisla rečenice. S druge strane, u slučaju potpuno povezanih mreža, redosled svojstava u vektorskoj reprezentaciji podataka ne može se slobodno menjati jer mreža uči različite vrednosti parametara za različita svojstva. U skorije vreme, napravljeni su veliki pomaci u obradi sekvencijalnih podataka konvolutivnim mrežama, ali modeli koji su konstruisani za ovu namenu i koji su dugo dominirali primenama nad ovakvim podacima su rekurentne neuronske mreže.

Osnovna ideja rekurentnih neuronskih mreža je da se sekvenca čita element po element, pri čemu se održava stanje procesa obrade u vidu vektora koji se menja iz koraka u korak. Da bi ovaj proces funkcionisao, potrebno je definisati kako naredno stanje zavisi od prethodnog stanja i od tekućeg ulaza. Takođe, pošto mreža treba da daje i izlaze, potrebno je definisati i na koji način izlaz u svakom koraku zavisi od tekućeg stanja. Ove zavisnosti se, kao i u prethodnim slučajevima neuronskih mreža, izražavaju linearnim transformacijama sa nelinearnom aktivacionom funkcijom, pri čemu se matrice koje definišu linearne transformacije uče.

Definicija 11.16 (Rekurentna mreža)**.** Rekurentna neuronska mreža *je funkcija* $f_{\mathbf{w}}$ *definisana na sledeći način:*

$$\begin{aligned} \mathbf{h}_0 &= 0 \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}_h), \quad t = 1, \ldots, T \\ \mathbf{o}_t &= \tilde{g}(\mathbf{V}\mathbf{h}_t + \mathbf{b}_o), \quad t = 1, \ldots, T \end{aligned}$$

gde T *predstavlja dužinu sekvence,* $\mathbf{x}_t$ *element ulazne sekvence,* $\mathbf{h}_t$ *predstavlja vektor* skrivenog stanja *u koraku* t*,* $\mathbf{o}_t$ *izlaz mreže u koraku* t*, matrica* $\mathbf{U}$ *definiše zavisnost novog od prethodnog skrivenog stanja, matrica* $\mathbf{W}$ *zavisnost novog skrivenog stanja od novog ulaza, matrica* $\mathbf{V}$ *definiše zavisnost izlaza od tekućeg skrivenog stanja, a funkcije* g *i* $\tilde{g}$ *su aktivacione funkcije. Vektori* $\mathbf{b}_h$ *i* $\mathbf{b}_o$ *su vektori slobodnih članova. Pomenute matrice i slobodni članovi skupa čine parametre* $\mathbf{w}$ *rekurentne mreže.*

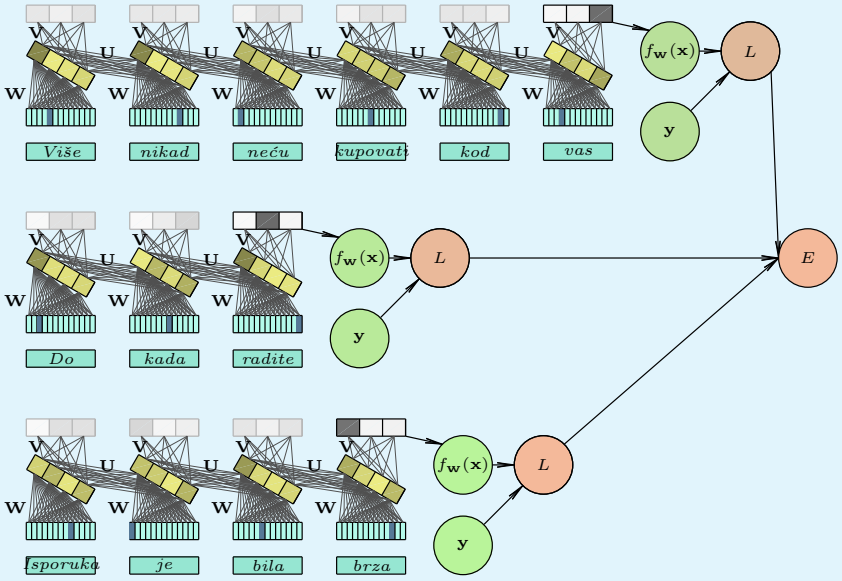
Bitno je primetiti da parametri $\mathbf{w}$ ne zavise od koraka t. Vrednost modela $f_{\mathbf{w}}(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ može biti bilo koji podskup izračunatih izlaznih vrednosti, zavisno od namene modela. Potom je nad datim izlazom moguće iz-

računati grešku u odnosu na zadate izlaze i formirati funkciju greške koju je moguće optimizovati gradijentnim metodama.

Primer 11.16. *Razmotrimo klasifikaciju sentimenta teksta u pozitivnu, neutralnu i negativnu kategoriju na primeru komentara kupaca na strani neke prodavnice. Za potrebe primera, smatraćemo da se trening skup sastoji samo od naredna tri komentara:*

- *„Više nikad neću kupovati kod vas.“*
- *„Do kada radite?“*
- *„Isporuka je bila brza.“*

Prvi komentar se može smatrati negativnim, drugi neutralnim, a treći pozitivnim. Na narednoj slici, prikazana je primena rekurentne mreže nad ovim rečenicama, koju ćemo objasniti u ostatku ovog primera.



Kako bi se primenila rekurentna mreža, svaka od reči iz trening skupa mora se predstaviti u numeričkom obliku i za to se koristi standardna reprezentacija kategoričkih vrednosti. Ako u trening skupu postoji ukupno n reči, one se predstavljaju binarnim vektorima dužine n, pri čemu vektor koji odgovara i-toj reči na i-toj poziciji sadrži jedinicu, a na svim ostalim pozicijama sadrži nule. U slučaju datog trening skupa, reprezentacija dimenzije 13 *je dovoljna. Na slici je reprezentacija svake reči prikazana neposredno iznad nje. U reprezentaciji reči, tamnije polje označava jedinicu, dok svetlija polja označavaju nule.*

Pretpostavimo da je dimenzija vektora skrivenog stanja jednaka 4 *(što bi se moralo eksperimentalno podesiti tako da rezultati budu što bolji). Tada je matrica* $\mathbf{W}$ *dimenzija* 4×13*, a matrica* $\mathbf{U}$ *dimenzija* 4×4*. Kako se klasifikacija vrši u* 3 *klase, matrica* $\mathbf{V}$ *je dimenzija* 3×4*. Za aktivacionu funkciju može se koristiti ReLU, osim na izlazu gde se koristi funkcija softmax. Za svaku reč neke rečenice, njena reprezentacija množi se matricom* $\mathbf{W}$*. Tako dobijen vektor sabira se sa vektorom koji se dobija primenom matrice* $\mathbf{U}$ *na skriveno stanje prethodne reči (za koji se uzima nula vektor ako prethodna reč ne postoji) i vektorom slobodnih članova, čime se uz primenu aktivacione funkcije dobija skriveno stanje za datu reč. Različite vrednosti skrivenog stanja su na slici prikazane različitim nijansama žute boje. Izračunavanje skrivenih stanja vrši se za svaku rečenicu posebno. Očigledno, izračunavanje se mora vršiti sekvencijalno, reč po reč, jer je za izračunavanje skrivenog stanja jedne reči potrebno skriveno stanje prethodne. Broj skrivenih stanja* T *je za svaku rečenicu različit i jednak broju reči u njoj. Za svako skriveno stanje, izračunava se izlaz mreže. To u ovom primeru nije potrebno jer nema smisla suditi o sentimentu rečenice nakon prve obrađene reči, pa su zato izlazi obojeni bledom bojom. U opštem slučaju obrade sekvencijalnih podataka, može biti potrebno računati izlaze u svakom koraku. Relevantan izlaz za svaku rečenicu je poslednji. Klasa koju mreža predviđa je označena tamnijom nijansom. Ako je* $\mathbf{x}$ *ulazna rečenica,* $f_{\mathbf{w}}(\mathbf{x})$ *predstavlja predviđenu klasu. Ovo predviđanje i stvarna vrednost* $\mathbf{y}$ *predstavljeni su na slici svetlozelenom bojom. Funkcija greške* L *primenjuje se na te parove, a uprosečavanjem grešaka po rečenicama dobija se prosečna greška* E*, čijom optimizacijom po matricama* $\mathbf{U}$*,* $\mathbf{W}$*,* $\mathbf{V}$ *i vektorima* $\mathbf{b}_h$ *i* $\mathbf{b}_o$ *se trenira mreža.*

Kada je mreža istrenirana, rečenice se klasifikuju tako što se mreža, kao i prilikom treninga, primeni na sve njene reči, nakon čega se očita izlaz za poslednju reč rečenice.

Moderne rekurentne mreže uključuju mnoštvo unapređenja u odnosu na osnovni princip koji je ovde izložen, ali njihovo razmatranje izlazi van okvira ovog teksta.

11.6 Metoda k najbližih suseda

Metoda k najbližih suseda je jedna od *metoda učenja zasnovanih na instacama*. Osnovna karakteristika ovakvih metoda je da ne grade eksplicitan model podataka u vidu neke funkcije kao što to radi većina metoda mašinskog učenja. Umesto izgradnje modela, instance predviđene za treniranje se čuvaju i bivaju upotrebljene tek kad je potrebno izvršiti predviđanje za novu, nepoznatu instancu. Time se većina izračunavanja premešta iz faze učenja u fazu primene. Ove metode često ne uključuju sva četiri pomenuta elementa dizajna algoritama nadgledanog učenja. Na primer, algoritam k najbližih suseda nema eksplicitnu formu modela, funkciju greške, niti koristi optimizaciju.

Metoda k najbližih suseda zasniva se na vrlo jednostavnoj ideji — pronaći k instanci najsličnijih nepoznatoj instanci, takozvanih suseda i predvideti vrednost njene ciljne promenljive na osnovu vrednosti koje odgovaraju susedima. U slučaju klasifikacije, predviđanje odgovara najčešćoj klasi među klasama suseda, a u slučaju regresije, predviđanje može biti prosečna vrednost ciljne promenljive suseda. Tekst u nastavku se odnosi na problem klasifikacije, ali analogni zaključci mogu se izvesti i u slučaju regresije. Pojam sličnosti instanci najjednostavnije se formalizuje preko funkcija rastojanja.

Definicija 11.17 (Rastojanje)**.** *Neka je X skup instanci. Funkcija $d : X \times X \to \mathbb{R}$ predstavlja* rastojanje na skupu X *ukoliko zadovoljava sledeće uslove:*

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$, *pritom* $d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$ *(pozitivna definitnost)*
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ *(simetričnost)*
3. $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ *(nejednakost trougla)*

Primer 11.17. *Neki primeri rastojanja su:*

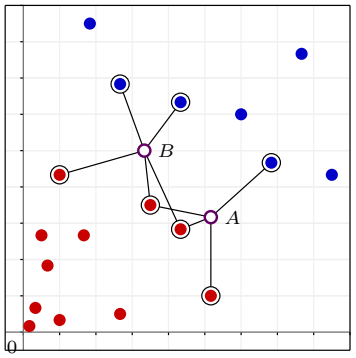
- $d(\mathbf{x}, \mathbf{y}) = \sqrt[n]{\sum_i (x_i - y_i)^n}$
- $d(\mathbf{x}, \mathbf{y}) = \begin{cases} 0, & \mathbf{x} = \mathbf{y} \\ 1, & \mathbf{x} \neq \mathbf{y} \end{cases}$

Intuitivno, što je rastojanje između dva objekta veće, to je sličnost između njih manja i obratno.[12] Drugim rečima, rastojanje između objekata predstavlja meru njihove različitosti. Moguće je izabrati raznovrsne funkcije rastojanja, a naravno poželjno je izabrati funkciju koja stvarno oslikava različitost između dva objekta, u smislu relevantnom za razmatrani problem.

Analizirajmo detaljnije metodu k najbližih suseda. Razmotrimo nepoznate instance A i B prikazane na slici 11.8. Metodom k najbližih suseda uz korišćenje euklidskog rastojanja, instanca A biva klasifikovana u crvenu klasu za sve vrednosti k od 1 do 5. Klasifikacija instance A je postojana zato što se ona nalazi blizu crvenih instanci, a udaljena je od plavih instanci. S druge strane, klasa instance B može da varira u zavisnosti od broja k. Za $k = 1$, instanca B klasifikuje se u crvenu klasu. Za $k = 2$, ne može se odlučiti. Za $k = 3$, instanca B klasifikuje se u plavu klasu. Za $k = 4$, ponovo nije moguće odlučiti, a za $k = 5$, instanca B ponovo se klasifikuje u crvenu klasu. Klasifikacija instance B nije postojana jer se ona nalazi blizu instanci iz obe klase. Dakle, metoda k najbližih suseda postojana je u unutrašnjosti oblasti koju zauzimaju instance jedne klase, ali je nepostojana na obodu te oblasti. Nepostojanost klasifikacije može se demonstrirati menjanjem hiperparametra k. Ali i za fiksiranu vrednost hiperparametra k može se uočiti nepostojanost pri variranju vrednosti svojstava instance, jer instanca iz oblasti jedne klase može takvim variranjem preći u oblast druge klase. Ovakvo ponašanje bi se moglo uočiti i kod drugih metoda klasifikacije.

Kao što se može videti u slučaju metode najbližih suseda, bitno svojstvo metoda zasnovanih na instancama je njihova lokalnost. Nepoznata instanca klasifikuje se isključivo ili uglavnom na osnovu poznatih instanci koje se nalaze u njenoj blizini. Ovo svojstvo doprinosi fleksibilnosti modela koje ove metode grade. Samim tim,

[12] Kao mere sličnosti, nekad se koriste i funkcije koje nisu zasnovane na rastojanjima kao, na primer, funkcija $\cos(\angle(\mathbf{x}, \mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})/(\sqrt{\mathbf{x} \cdot \mathbf{x}}\sqrt{\mathbf{y} \cdot \mathbf{y}})$.



Slika 11.8: Stabilnost klasifikacije pomoću algoritma k najbližih suseda.

za manje vrednosti hiperparametra k dobijaju se fleksibilniji modeli, koji su stoga skloniji preprilagođavanju, dok se za veće vrednosti hiperparametra k dobijaju manje fleksibilni modeli manje skloni preprilagođavanju. Naravno, premala fleksibilnost vodi modelima koji se ne mogu dovoljno prilagoditi podacima i stoga loše uče, pa ni premala ni prevelika vrednost hiperparametra k nije dobra. Očigledno, hiperparametar k ima ulogu sličnu ulozi regularizacionog hiperparametra λ. Određivanje njihovih vrednosti biće zajednički diskutovano kasnije.

Kako ovaj metod ne uključuje trening, već se instance koriste u vreme klasifikacije, trening skup se uvek može ažurirati, recimo dodavanjem novih instanci, bez utroška vremena na ponovni trening kao u slučaju drugih algoritama.

Razmotrimo primer primene algoritma k najbližih suseda u problemu regresije.

Primer 11.18. *Vratimo se na primer procene zagađenja vazduha. U nastavku ponavljamo standardizovane podatke, ali bez kolone jedinica. Naime, u slučaju primene algoritma najbližih suseda nema slobodnog člana pa ni dodatna kolona nije potrebna.*

$$\bar{\mathbf{X}} = \begin{bmatrix} 1.41 & 1.81 & -0.71 \\ 1.41 & -0.44 & -0.71 \\ 1.41 & -0.75 & -0.71 \\ 1.41 & -0.44 & -0.71 \\ -0.63 & 0.38 & 1.41 \\ -0.63 & -1.26 & 1.41 \\ -0.63 & -1.26 & 1.41 \\ -0.63 & -0.44 & 1.41 \\ -0.78 & 2.01 & -0.71 \\ -0.78 & 0.17 & -0.71 \\ -0.78 & 0.38 & -0.71 \\ -0.78 & -0.14 & -0.71 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 70 \\ 168 \\ 181 \\ 141 \\ 121 \\ 245 \\ 247 \\ 112 \\ 63 \\ 80 \\ 67 \\ 61 \end{bmatrix}$$

Neka je potrebno predvideti zagađenje za novu instancu $(1000000, 15, 0)$ *koja predstavlja grad od milion stanovnika, van kotline, na dan kada vetar duva brzinom od* 15 *kilometara na sat. Prvo je potrebno izvršiti standardizaciju ove instance kako bi promenljive koje je opisuju bile izražene na istoj skali kao promenljive u matrici podataka, čime se dobija instanca* $(0.17, -0.24, -0.71)$*. Vektor rastojanja do datih instanci je:*

$$\mathbf{d} = \begin{bmatrix} 2.39 \\ 1.25 \\ 1.34 \\ 1.25 \\ 1.01 \\ 2.49 \\ 2.49 \\ 2.28 \\ 3.24 \\ 1.04 \\ 1.13 \\ 0.96 \end{bmatrix}$$

Regresija može biti izvedena na primer pomoću tri najbliža suseda, uprosečavanjem njihovih vrednosti. To su instance sa rastojanjima 0.96, 1.01 *i* 1.04*, a prosek njihovih y vrednosti je* 87.33*, što je konačno predviđanje.*

11.7 Stabla odlučivanja

Razmotrimo „igru 20 pitanja“. Jedan igrač zamišlja neki predmet, a drugi treba da pogodi o kom je predmetu reč. Kako bi pogodio o kom predmetu se radi, igrač koji pogađa ima pravo da postavi 20 pitanja na koje odgovor može biti „da“ ili „ne“. Kada smatra da je postavio dovoljno pitanja, igrač može dati svoj sud o kom predmetu se radi i igra se završava. Očito, proces ispitivanja može se predstaviti u vidu stabla koje u svakom čvoru ima po jedno pitanje, osim u listovima u kojima se nalazi sud igrača o nepoznatom predmetu. Iz svakog čvora, osim listova polaze dve grane označene sa „da“ ili „ne“, koje vode u podstablo koje odgovara nastavku ispitivanja posle razmatranog pitanja. Ovo je primer stabla odlučivanja. Ovakva stabla mogu se uopštiti odbacivanjem ograničenja na 20 pitanja i tako što bi se dozvolilo da odgovori ne moraju biti samo „da“ ili „ne“, već da mogu pripadati nekom drugom skupu.

Stabla odlučivanja mogu se automatski naučiti iz skupa primera koji za svaku instancu uključuju vrednosti njenih bitnih svojstava i vrednost ciljne promenljive. Mogu se primeniti i na probleme klasifikacije i na probleme regresije. Slično „igri 20 pitanja“, svakom čvoru takvog stabla odgovara test nekog svojstva instance, a grane koje izlaze iz čvora različitim vrednostima tog svojstva. Listovima odgovaraju predviđene vrednosti ciljne promenljive. Instance su opisane vrednostima svojih svojstava. Predviđanje se vrši polazeći od korena, spuštajući se niz granu koja odgovara vrednosti testiranog svojstva instance za koju se vrši predviđanje. Predviđanje se dodeljuje instanci kad se dođe do lista.

Učenje stabala odlučivanja uspešno se primenjuje u različitim problemima. Još sredinom devedesetih godina prošlog veka, stabla odlučivanja primenjena su u klasifikaciji tumora i prognozi njihovog ponašanja. Svaka trening instanca opisivana je pomoću 31 svojstva, a klasifikacije su date nezavisno od strane više stručnjaka. U astronomiji su stabla odlučivanja primenjena u cilju razlikovanja zvezda i tragova kosmičkih zraka na snimcima teleskopa Habl. Na osnovu 20 numeričkih svojstava, sa stablima dubine do 9 čvorova, postignuta je preciznost klasifikacije od 95%. Postoje primene stabala odlučivanja i u ekonomiji i drugim oblastima. Određivanje poze igrača kakvo vrši Majkrosoftov proizvod Kinekt biće detaljnije diskutovano kasnije.

Korišćenje stabla odlučivanja nije podjednako pogodno za sve probleme učenja. Poželjno je da skup vrednosti svojstava bude diskretan i mali, mada postoje varijante koje rade sa neprekidnim svojstvima. Stabla odlučivanja su posebno pogodna u slučajevima kada je potrebno izraziti model na interpretabilan način. Ukoliko stablo odlučivanja instanci dodeljuje neko predviđanje, to znači da instanca ispunjava sve uslove koji su definisani putanjom od korena do odgovarajućeg lista kroz stablo i oblika su *svojstvo=vrednost*. Stoga putanje kroz stablo predstavljaju konjunkcije ovakvih uslova. U slučaju klasifikacije, za svaku klasu moguće je uočiti putanje koje se završavaju listovima koji odgovaraju toj klasi. Disjunkcija svih takvih konjunkcija definiše instance koje pripadaju datoj klasi prema datom stablu. Pravilo izražena u vidu disjunkcije konjunkcija elementarnih uslova omogućava interpretabilnost, odnosno razumevanje načina na koji model donosi odluke od strane čoveka.

Jedan od najpoznatijih algoritama za učenje stabla odlučivanja je ID3, na koji ćemo se fokusirati. Ovaj algoritam primenjuje se samo na instance koje su opisane isključivo kategoričkim atributima. On konstruiše stablo od korena ka listovima, određujući u svakom čvoru najbolje svojstvo koje se u tom čvoru može testirati. To svojstvo se određuje statističkim kriterijumom koji meri koliko dobro neko svojstvo samostalno klasifikuje podatke. Za sve vrednosti odabranog svojstva kreiraju se grane do čvorova naslednika, a podaci za treniranje se dele između ovih čvorova tako da svaki od njih nasleđuje primere koji imaju odgovarajuću vrednost prethodno testiranog svojstva. Za svaki od čvorova naslednika ovaj postupak se primenjuje rekurzivno sve dok nije ispunjen bar jedan od sledeća dva uslova: (1) u putanji od korena do trenutnog čvora iskorišćena su sva svojstva, (2) sve instance za trening koje su pridružene trenutnom čvoru imaju istu vrednost ciljne promenljive. Svakom listu se pridružuje najčešća oznaka instanci za trening koje su mu pridružene. Algoritam je preciznije opisan na slici 11.9.

Veoma je važno pitanje statističkog kriterijuma koji se koristi za izbor najboljeg svojstva za testiranje u nekom čvoru. Algoritam ID3 obično bira svojstvo koje maksimizuje *dobitak informacije* na skupu instanci koje su pridružene razmatranom čvoru. Dobitak informacije predstavlja razliku entropije u odnosu na ciljnu promenljivu skupa instanci za trening D pre deljenja i prosečne entropije posle deljenja prema nekom svojstvu S. Entropija predstavlja meru neuređenosti nekog sistema. Ako sa p_i označimo verovatnoću da instanca pripada i-toj klasi, onda se entropija može definisati sledećim izrazom (pri čemu smatramo da važi $0 \cdot \log_2 0 = 0$):

$$H(D) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

Algoritam: ID3

Ulaz: Skup instanci za trening D, n dimenzija instanci

Izlaz: Stablo odlučivanja koje odgovara datim instancama

1: inicijalizuj $S = \{1, 2, \ldots, n\}$;
2: napravi koreni čvor stabla R;
3: **ako** sve instance iz D pripadaju istoj klasi **onda**
4: vrati čvor R sa oznakom te klase;
5: **ako** je S prazan skup **onda**
6: formiraj čvor R označen oznakom najčešće klase koja se javlja u D;
7: **inače**
8: neka je $j \in S$ indeks *najboljeg svojstva* (prema nekom statističkom kriterijumu) za testiranje u odnosu na D;
9: označi čvor R j-tim svojstvom;
10: **za** svaku moguću vrednost v j-tog svojstva **radi**
11: dodaj granu iz R koja odgovara testu $x_j = v$;
12: neka je D_v podskup od D takav da za sve njegove elemente važi $x_j = v$;
13: **ako** je skup D_v prazan **onda**
14: na dodatu granu iz R dodaj list sa oznakom najčešće klase u D;
15: **inače**
16: na dodatu granu nadoveži podstablo ID3(D_v,$S \setminus \{j\}$);
17: vrati R.

Slika 11.9: Algoritam ID3.

gde je c broj klasa, tj. broj mogućih vrednosti ciljne promenljive, p_i udeo instanci iz skupa D koje pripadaju klasi i u celom skupu D. Dobitak informacije stoga predstavlja smanjenje u potrebnom broju bitova za kodiranje klase proizvoljne instance, kada je poznata vrednost koju na njoj ima svojstvo S. Dobitak informacije se formalno definiše na sledeći način:

$$G(D, j) = H(D) - \sum_{v \in V(j)} \frac{|D_v|}{|D|} H(D_v) \tag{11.1}$$

gde $V(j)$ predstavlja skup svih mogućih vrednosti j-tog svojstva, a $D_v = \{\mathbf{x} \in D | x_j = v\}$.

Pored entropije, postoje i druge mere koje se mogu koristiti za merenje neuređenosti nekog skupa. Jedna jednostavna mera je greška klasifikacije. Ona predstavlja grešku koja se čini ukoliko se sve instance nekog skupa klasifikuju u najbrojniju klasu u tom skupu. Stoga, ako je p_i verovatnoća da instanca pripada i-toj klasi, greška klasifikacije za skup podataka D definiše se izrazom

$$Err(D) = 1 - \max_i p_i$$

Za ovu meru može se definisati dobitak analogan opisanom dobitku informacije ukoliko se u izrazu (11.1) entropija zameni greškom klasifikacije.

Primer 11.19. *U narednoj tabeli date su instance koje opisuju različite životinje, sa datom klasifikacijom koja označava da li je životinja opasna po čoveka. Izdvojeno je nekoliko karakteristika koje bi mogle biti relevantne u određivanju vrednosti ciljne promenljive, ali su namerno dodata dva svojstva koji nisu relevantna — broj nogu i da li životinja živi u Evropi.*

Životinja	*Veličina*	*Ishrana*	*Otrovnost*	*Noge*	*Evropa*	*Opasna*
Lav	*Velika*	*Meso*	*Ne*	*4*	*Ne*	*Da*
Mačka	*Mala*	*Meso*	*Ne*	*4*	*Da*	*Ne*
Koala	*Mala*	*Biljke*	*Ne*	*4*	*Ne*	*Ne*
Zec	*Mala*	*Biljke*	*Ne*	*4*	*Da*	*Ne*
Komodo zmaj	*Velika*	*Meso*	*Da*	*4*	*Ne*	*Da*

Da bi se izgradilo stablo odlučivanja, za svako od svojstava je potrebno izračunati dobitak informacije pri deljenju skupa podataka prema tom svojstvu. U prvom koraku, važi:

$$H(D) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$

Ukoliko se izvrši podela instanci po vrednosti prvog svojstva, dobijamo

$$H(D_{Mala}) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0$$

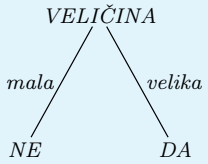
$$H(D_{Velika}) = -0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0$$

$$G(D, Velicina) = 0.971 - \frac{2}{5} \cdot 0 - \frac{3}{5} \cdot 0 = 0.971$$

gde zarad čitljivosti, umesto indeksa svojstva pišemo naziv. Slično se dobija:

$$G(D, Ishrana) = 0.42$$
$$G(D, Otrovnost) = 0.322$$
$$G(D, Noge) = 0$$
$$G(D, Evropa) = 0.42$$

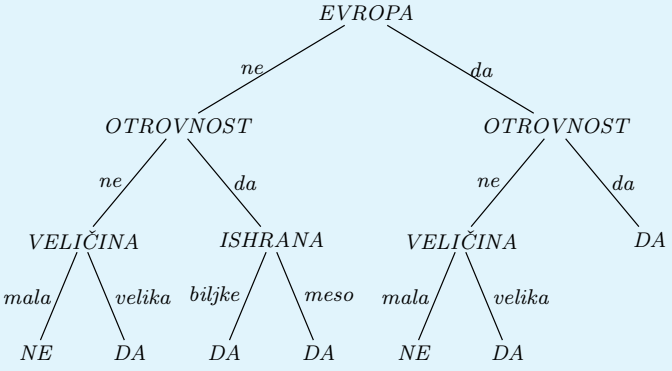
Odavde se vidi da je najbolje svojstvo za testiranje u prvom čvoru svojstvo Veličina. Stablo koje se u ovom slučaju dobija primenom algoritma ID3 je dato na narednoj slici. U slučaju datih primera za učenje, dobijeno stablo je bilo očigledan izbor i bez primene bilo kakve metodologije.



Nešto komplikovanije stablo odlučivanja može se dobiti dodavanjem primera iz naredne tabele.

Životinja	Veličina	Ishrana	Otrovnost	Noge	Evropa	Opasna
Zmija	Mala	Meso	Da	0	Da	Da
Pčela ubica	Mala	Biljke	Da	6	Ne	Da
Morska krava	Velika	Biljke	Ne	0	Ne	Ne

Jedno ručno konstruisano stablo koje je saglasno sa podacima za trening dato je na narednoj slici.



Izbor lošeg svojstva za testiranje u korenom čvoru je namerno učinjen. To dovodi do potrebe za ponavljanjem istih testova u levom i desnom podstablu, pošto informacija dobijena testiranjem u korenu nije relevantna za određivanje klase instance. Takođe, u slučaju životinja koje ne žive u Evropi, prisutan je nepotreban test vezan za njenu ishranu. U oba slučaja klasa je ista, pa se čvor sa tim testom može zameniti listom sa klasom DA. Upotreba algoritma ID3 daje bolje stablo odlučivanja.

Vrednosti entropije i dobitka informacije se sada razlikuju:

$$H(D) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$

Ukoliko se izvrši podela instanci po vrednosti prvog svojstva, dobijamo

$$H(D_{Mala}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971$$

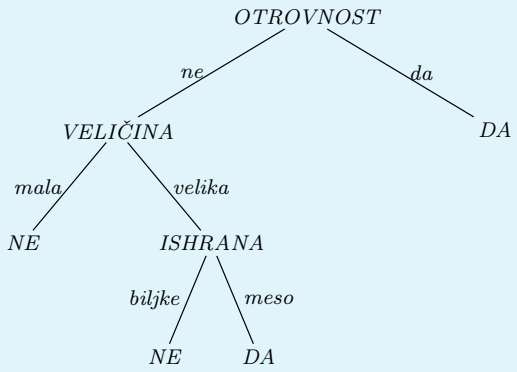
$$H(D_{Velika}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.918$$

$$H(D, Velicina) = 1 - \frac{5}{8}\cdot 0.971 - \frac{3}{8}\cdot 0.918 = 0.049$$

Slično se dobija:

$$G(D, Ishrana) = 0.189$$
$$G(D, Otrovnost) = 0.549$$
$$G(D, Noge) = 0$$
$$G(D, Evropska) = 0.02$$

Posle dodavanja novih instanci, najbolja svojstva za testiranje su Otrovnost i Ishrana. Rekurzivnom primenom ovog postupka dobija se stablo prikazano na narednoj slici. Ono je očigledno manje od ručno konstruisanog i ima relevantnija svojstva pri vrhu stabla, dok se dva nebitna svojstva uopšte ne testiraju.



Stabla odlučivanja mogu se koristiti i za probleme regresije. U tom slučaju, za merenje neuređenosti skupa instanci prirodan izbor je disperzija vrednosti ciljne promenljive za te instance:

$$\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})^2$$

gde važi $\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$. U slučaju regresije potrebno je modifikovati i način na koji se listovima pridružuju predviđanja koja daju. Ona se mogu definisati kao prosek instanci koje su prilikom treninga pridružene listu.

Kao i drugi metodi učenja, ID3 se može shvatiti kao pretraga skupa dopustivih modela za onim koji najviše odgovara podacima za trening. Prostor pretrage je potpun prostor svih stabala odlučivanja. Svaka diskretna funkcija može se predstaviti nekim stablom odlučivanja, tako da se učenjem stabala odlučivanja može postići greška 0 na trening podacima ukoliko podaci nisu protivrečni. Pošto nema vraćanja u pretrazi (tj. nema bektrekinga), već se stablo gradi od jednostavnijeg ka složenijem, postoji opasnost od dostizanja lokalnih optimuma koji nisu globalni.

Algoritam ID3 preferira stabla manje dubine, kao i stabla u kojima se svojstva koja nose veći dobitak informacije nalaze bliže korenu. Afinitet prema kraćim stablima je zanimljivo svojstvo jer je u skladu i sa

odavno poznatim filozofskim principom kojim se često vode i naučnici — Okamovom oštricom: entitete ne treba umnožavati preko potrebe, tj. najjednostavnije objašnjenje je verovatno i najbolje. Svakoj putanji od korena do nekog od listova odgovara po jedno pravilo oblika

$$\texttt{if } x_1 = v_1 \ \wedge \ x_2 = v_2 \ \wedge \ \ldots \ \wedge \ x_n = v_n \texttt{ then}$$
$$Klasa = klasa\ koja\ odgovara\ listu$$

gde su x_i $0 \leq i \leq n$ svojstva koja se testiraju na putanji od korena do odgovarajućeg čvora, a v_i neke njihove moguće vrednosti. Kako stabla odlučivanja sa manjom dubinom imaju manji broj listova, ona predstavljaju manje skupove ovakvih pravila, te ih možemo smatrati jednostavnijim.

Osvrnimo se i na problem preprilagođavanja. Sa povećanjem dozvoljene dubine stabla, povećava se moć učenja, odnosno verovatnoća da će u skupu dopustivih modela biti nađen onaj koji dobro opisuje podatke. Zato se sa povećanjem dozvoljene dubine stabala, smanjuje greška na trening podacima. Međutim, ako nema ograničenja na dubinu stabla, takav skup modela je očigledno vrlo bogat i stoga postoji opasnost od preprilagođavanja. Jedan pristup rešavanju ovog problema je ograničavanje maksimalne dubine stabla nakon koje algoritam učenja neće dalje razgranavati stablo. O načinu na koji se vrši izbor maksimalne dubine biće reči kasnije, pošto se bira na sličan način kao vrednost regularizacionog hiperparametra λ. Zapravo, kako se ovom tehnikom smanjuje fleksibilnost modela u vreme učenja, ona upravo jeste vid regularizacije.

11.8 Evaluacija modela i konfigurisanje algoritama učenja

Za primenu modela u praksi, od presudnog je značaja znati koliko dobro model generalizuje, odnosno koliko su mu dobra predviđanja. Za to je s jedne strane potrebno imati nekakve mere kvaliteta, s druge procedure koje obezbeđuju da se tako izračunatim merama može dovoljno verovati.

Kao što se moglo primetiti, algoritmi mašinskog učenja su konfigurabilni. Naime, moguće je podešavati regularizacioni hiperparametar kod linearnih modela, k kod algoritma k najbližih suseda, maksimalnu dubinu stabla odlučivanja, arhitekturu mreže, itd. Izbor različite konfiguracije vodi različitom optimizacionom problemu i stoga različitom rešenju. Poželjno je izabrati konfiguraciju koja vodi dobrom rešenju. Ovo ne mora biti trivijalan posao.

11.8.1 Mere kvaliteta regresije

Osnovna mera kvaliteta regresije je srednjekvadratna greška, koja meri odstupanje predviđenih od stvarnih vrednosti na nekom test skupu.

Definicija 11.18 (Srednjekvadratna greška)**.** *Za date podatke* $\{(\mathbf{x}_i, y_i)|i = 1 \ldots, N\}$, srednjekvadratna greška *modela* f_w *data je sledećom formulom:*

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\mathbf{w}}(\mathbf{x}))^2.$$

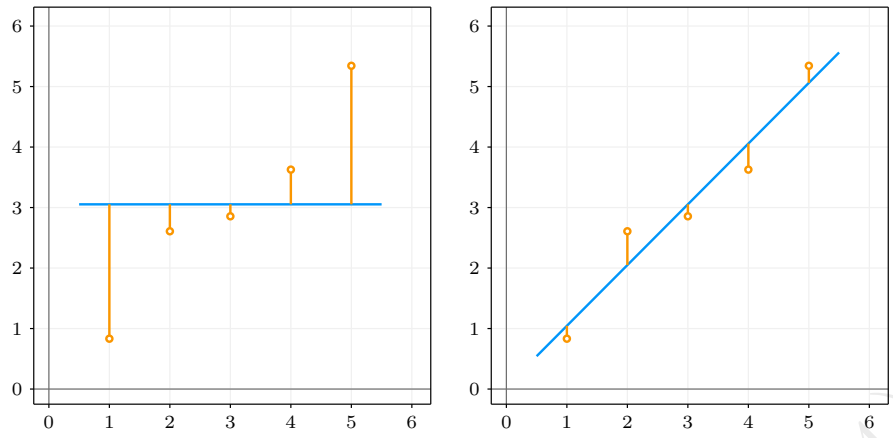
Obično je bolje koristiti koren srednjekvadratne greška, pošto se on izražava na istoj skali kao i veličina y, pa je taj broj razumljiviji u kontekstu domena. Poželjno je da srednjekvadratna greška bude što manja, međutim ukoliko nemamo konkretan zahtev za postizanjem određene srednjekvadratne greške, teško je reći da li je učenje uspešno ili ne. Zbog toga se često koriste i druge mere. Česta je upotreba *koeficijenta determinisanosti.*

Definicija 11.19 (Koeficijent determinisanosti)**.** Koeficijent determinisanosti *modela* f_w *na datim podacima* $\{(\mathbf{x}_i, y_i)|i = 1 \ldots, N\}$ *dat je sledećom formulom:*

$$R^2(\mathbf{w}) = 1 - \frac{\sum_{i=1}^{N}(y_i - f_{\mathbf{w}}(\mathbf{x}))^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$$

gde $\overline{y}$ *označava prosek uzorka promenljive* y*.*

Vrednost 1 koeficijenta determinisanosti označava potpuno podudaranje stvarnih i predviđenih vrednosti. Što je vrednost koeficijenta manja, to je poklapanje lošije. Smisao koeficijenta determinisanosti može se bolje razumeti ako se primeti da se u njegovom izračunavanju zapravo poredi greška proseka $\overline{y}$ u odnosu na stvarne podatke sa greškom modela u odnosu na stvarne podatke. Prosek se može smatrati vidom trivijalnog modelovanja ciljne



Slika 11.10: Grafici grešaka u odnosu na prosek (levo) i u odnosu na model (desno).

promenljive (bez korišćenja svojstava) i stoga se njegova greška može smatrati relevantnom referentom vrednošću koju model treba da nadmaši. Srednjekvadratna greška proseka je

$$E(\overline{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2.$$

Rastojanja koja ulaze u ovu grešku prikazana su na slici 11.10, levo.

U slučaju korišćenja linearnog regresionog modela, srednjekvadratna greška je

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{x})^2$$

Rastojanja koja ulaze u ovu grešku prikazana su na slici 11.10, desno. Što su svojstva na osnovu kojih je izgrađen model informativnija, greške koje pravi model će biti manje u odnosu na greške koje pravi predviđanje pomoću proseka, a koeficijent determinisanosti je utoliko veći što je taj odnos povoljniji za dati model.

Primer 11.20. *Neka u primeru predviđanja telesne težine na osnovu visine, koeficijent determinisanosti iznosi 0.51. To znači da promenljiva* $\mathbf{x}$ *objašnjava oko pola varijanse promenljive* y *i da postoji prostor da se predikcija dalje popravi dodavanjem novih svojstava.*

11.8.2 Mere kvaliteta klasifikacije

Kvalitet klasifikacije zavisi od broja i vrste grešaka koje klasifikator pravi. Polazna tačka u definisanju mera kvaliteta klasifikacije je prepoznavanje mogućih ishoda rada klasifikatora. U nastavku razmatramo slučaj binarne klasifikacije, ali je to razmatranje moguće generalizovati i na veći broj klasa. Pod *stvarno pozitivnim* instancama podrazumevamo pozitivne instance koje su od klasifikatora prepoznate kao pozitivne. Pod *stvarno negativnim*, instance koje su negativne i prepoznate su kao negativne. Pod *lažno pozitivnim* podrazumevamo instance koje su negativne, ali su greškom u klasifikaciji proglašene pozitivnim i pod *lažno negativnim* podrazumevamo instance koje su pozitivne, ali su greškom klasifikovane kao negativne. Brojeve ovih instanci označavamo redom SP, SN, LP i LN. Ove brojeve obično zapisujemo u takozvanoj *matrici konfuzije* koja po vrstama prikazuje kako su instance klasifikovane, a po kolonama prikazuje koje su stvarne klase instanci. Ova matrica je prikazana u vidu tabele 11.1.

	P	N
KP	SP	LP
KN	LN	SN

Tabela 11.1: Vrste matrice konfuzije u zbiru daju broj instanci koje su klasifikovane pozitivno (KP) i broj instanci koje su klasifikovane negativno (KN). Kolone u zbiru daju broj instanci koje su zaista pozitivne (P) ili negativne (N).

U slučaju problema klasifikacije, zbog svoje intuitivnosti, najčešće korišćena mera kvaliteta je *tačnost* (eng. *accuracy*).

Definicija 11.20 (Tačnost)**.** *Za datu matricu konfuzije modela,* tačnost *modela predstavlja udeo tačno klasifikovanih među svim instancama:*

$$tačnost = \frac{SP + SN}{SP + LP + SN + LN}.$$

U primeru sa prepoznavanjem računarskih članaka, koristili smo upravo tačnost kao meru kvaliteta.

U nekim slučajevima tačnost nije adekvatna mera. Ukoliko postoje dve klase i jedna je značajno manja od druge, moguće je dobiti visoku tačnost tako što će se sve instance klasifikovati u veću klasu. Takav je slučaj sa testovima koji ustanovljavaju da li je pacijent oboleo od neke bolesti. Ako bolest ima samo 1% ljudi u populaciji, test koji bi uvek prijavljivao da pacijent nema bolest imao bi tačnost od 99%, ali bi bio neupotrebljiv. Otud se koriste i druge mere. Dve često korišćene mere su *preciznost* (eng. *precision*) i *odziv* (eng. *recall*).

Definicija 11.21 (Preciznost i odziv)**.** *Za datu matricu konfuzije modela,* preciznost *modela predstavlja udeo stvarno pozitivnih instanci među onima koje su klasifikovane kao pozitivne:*

$$prec = \frac{SP}{SP + LP}$$

a odziv *modela predstavlja udeo instanci koje su klasifikovane kao pozitivne među onima koje zaista jesu pozitivne:*

$$odziv = \frac{SP}{SP + LN}$$

Intuitivno, prva mera nam kaže koliko često smo u pravu kada tvrdimo da je neka instanca od interesa, dok druga govori koliki udeo instanci od interesa uspevamo da detektujemo. Poželjno je da ove vrednosti budu što veće. Svaka od ove dve mere lako se maksimizuje odvojeno, te je uvek potrebno razmatrati ih zajedno. Ako nikad ne tvrdimo da je neka instanca pozitivna, nikad ne grešimo u pozitivnoj klasifikaciji i preciznost je jednaka 1 (podrazumevamo da je u ovom kontekstu 0/0 jednako 1), ali će tada odziv biti 0. S druge strane, ako sve instance proglasimo za pozitivne odziv je jednak 1, ali će preciznost biti mala (jednaka udelu pozitivne klase). Kako je ponekad teško razmatrati dve mere paralelno, posebno kada se želi poređenje dva ili više različitih modela, često se koristi njihova harmonijska sredina, takozvana F_1 mera. Harmonijska sredina bliža je nižoj od dve vrednosti koje se usrednjavaju, tako da se na taj način dobija stroža ocena od aritmetičke sredine. Primer konteksta u kojem se ova mera koristi je pronalaženje dokumenata iz neke kolekcije koji odgovaraju nekom upitu, na primer prilikom pretrage na vebu. Korisna je i jer, za razliku od tačnosti, ne daje nerealističnu sliku u slučaju neizbalansiranih klasa.

Primer 11.21. *Neka je testirano 400 ispitanika. Od toga 5 imaju bolest zbog koje se testiraju, a ostali ne. Neka je test dao pozitivnu klasifikaciju u slučaju 3 osobe koje sve tri imaju bolest. Matrica konfuzije data je narednom tabelom.*

	P	*N*
KP	*3*	*0*
KN	*2*	*395*

Odavde se mogu izračunati mere kvaliteta:

$$tačnost = \frac{398}{400} = 0.995$$

$$prec = \frac{3}{3} = 1$$

$$odziv = \frac{3}{5} = 0.6$$

$$F_1 = 2\frac{1 \cdot 0.6}{1 + 0.6} = 0.75$$

Tačnost je izuzetno visoka i sugeriše odlično ponašanje klasifikatora. Preciznost sugeriše isto, ali poznavanje udela stvarno pozitivnih otkriva da je ovaj utisak lažan zato što je identifikovan mali broj elemenata pozitivne klase. Kako F_1 mera uzima u obzir i preciznost i odziv i po njoj se vidi da klasifikator ipak nije dobar kao što deluje na osnovu tačnosti.

Primer 11.22. *Pretpostavimo da je potrebno razviti sistem koji prepoznaje potencijalne teroriste među putnicima na nekom aerodromu. Kroz aerodrom prođe dnevno oko* 100000 *putnika, a očekivani broj potencijalnih terorista je, na primer,* 4*. Sistem procenu pravi na osnovu više informacija, uključujući starosnu dob putnika, državljanstvo, zemlje iz koje putuje i zemlje u koju putuje, arhivu ekstremističkih pokreta i slično. Svaki putnik označen kao potencijalni terorista mora da prođe kroz detaljnu dodatnu proveru.*

Ako bi sistem trivijalno za svakog putnika davao odgovor „ne“ (tj. procenu da nije potencijalni terorista), onda bi sistem imao odličnu tačnost ($99996/100000 = 99.996\%$*), ali je jasno da je takav sistem potpuno beskoristan. Upotrebljivost ovakvog sistema mora se proveriti kroz druge mere kvaliteta. U ovom slučaju, preciznost je savršena (jednaka* 1*) jer sistem nikoga nije pogrešno označio kao teroristu, ali odziv je jednak* 0*, što jasno ukazuje da je sistem beskorisan. F_1 mera je takođe jednaka* 0*.*

Upotrebljivost ima različite aspekte i ponekad postavlja suprotstavljene zahteve. U ovom konkretnom problemu, sa aspekta bezbednosti, potrebno je da odziv bude jednak 1 *ili što bliži toj vrednosti. Da bi se to obezbedilo, potrebno je sprovoditi dodatne provere nad što većim brojem putnika. Međutim, s druge strane, sa aspekta aerodromske logistike i ugođaja putnika, potrebno je da i preciznost bude što veća, jer nije moguće organizovati dodatni pregled za veliki broj putnika, niti je poželjno izlagati putnike neprijatnosti i gubitku vremena.*

Pretpostavimo da je ponašanje sistema opisano matricom konfuzije koja je data u narednoj tabeli.

	P	*N*
KP	*3*	*5996*
KN	*1*	*94000*

Odatle se mogu izračunati mere kvaliteta:

$$tačnost = \frac{94003}{100000} = 0.94003$$

$$prec = \frac{3}{5999} \approx 0.0005$$

$$odziv = \frac{3}{4} = 0.75$$

$$F_1 = 2\frac{0.0005 \cdot 0.75}{0.7505} \approx 0.001$$

Tačnost je lošija nego kod prethodnog, trivijalnog sistema, ali i dalje deluje visoka. Ipak, poređenjem F_1 mera moglo bi se zaključiti da ovaj sistem nije mnogo bolji od prethodnog sistema, ali se razlozi bolje vide analizom ostalih mera. Preciznost je očito vrlo niska, a ni odziv verovatno nije dovoljno visok za praktičnu upotrebljivost ovakvog sistema.

11.8.3 Tehnike evaluacije i konfigurisanja algoritama učenja

Pored izbora mere kvaliteta, bitno je izabrati i način na koji se ta mera ocenjuje. Česta praksa je da se model trenira na trening skupu, a da se evaluira na odvojenom skupu podataka za testiranje. Pritom se podela raspoloživih podataka na podatke za trening i podatke za testiranje vrši slučajnim izborom podataka za testiranje. Postoje i naprednije i pouzdanije tehnike evaluacije.

Iako nije očigledno da je u vezi sa tehnikama evaluacije, u nastavku će biti razmotreno dugo odlagano pitanje izbora vrednosti regularizacionog hiperparametra λ, kao i broja suseda k i maksimalne dubine stabla odlučivanja za koje smo najavili da ćemo ih razmatrati skupa. Izbor vrednosti ovih hiperparametara predstavlja samo primer opštijeg problema konfigurisanja algoritama učenja. U opštem slučaju, algoritmi učenja se mogu podešavati na različite načine, pri čemu različite konfiguracije daju različite modele za iste ulazne podatke. U nastavku će biti reči samo o izboru hiperparametra λ, ali se diskusija odnosi i na probleme konfigurisanja algoritama učenja u opštijem smislu.

Za dati skup podataka, svakoj vrednosti hiperparametra λ odgovara neka vrednost optimalnih parametara $\mathbf{w}_\lambda$ i samim tim neki model $f_{\mathbf{w}_\lambda}(\mathbf{x})$. Postavlja se pitanje koji od ovih modela je najbolji. Osim ako je dostupna

velika količina podataka, male vrednosti hiperparametra λ uzrokuju loše rezultate zbog preprilagođavanja, a velike vrednosti uzrokuju loše rezultate zbog premale fleksibilnosti modela. Poželjne vrednosti hiperparametra se obično nalaze negde između dva ekstrema. Stoga je prvi korak u pronalaženju pogodne vrednosti određivanje granica intervala u kojem će se vrednost tražiti, što se može uraditi eksperimentalno. Recimo interval $[10^{-10}, 10^5]$ je verovatno dovoljno širok u većini slučajeva. Potom se formira niz vrednosti hiperparametra koje se ispituju. Na primer, često se koristi geometrijska progresija $\lambda_1 = 10^{-6}, \lambda_2 = 10^{-5}, \ldots, \lambda_{10} = 10^3$. Potom se model za svaku od tih vrednosti evaluira i bira se najbolji. Ipak, postavlja se pitanje kako se vrši evaluacija.

Prva ideja bi bila da se za svaku od izabranih vrednosti hiperparametra izvrši treniranje na trening skupu i da se dobijeni model evaluira na test skupu nekom merom kvaliteta i da se izabere najbolji od njih. Pažljivijim razmatranjem se uviđa da je ovaj postupak pogrešan. Naime, na ovaj način se podaci iz test skupa koriste pri izboru modela, što je sve deo treninga i samim tim i oni predstavljaju deo trening skupa. Međutim, disjunktnost trening i test skupa je osnovno pravilo evaluacije modela u mašinskom učenju. Korektan postupak bi bio da se umesto podele ukupnog skupa podataka na trening i test skup izvrši njegova podela na *trening skup*, *validacioni skup* i *test skup*. Tada se na trening skupu vrši treniranje svakog od modela (dobijenih za različite vrednosti λ_i), na validacionom skupu se vrši evaluacija na osnovu koje se bira najbolji model (koji odgovara nekoj vrednosti hiperparametra lambda) i potom se taj model evaluira na test skupu i tako dobijena mera predstavlja finalnu meru kvaliteta modela.

Hiperparametar koji se obično ne bira pomoću validacionog, već pomoću trening skupa je brzina učenja. Naime, brzina učenja bitno utiče na postizanje i brzinu konvergencije procesa minimizacije greške. Kako se ta minimizacija vrši na trening skupu, na njemu se i posmatra njen napredak. Nasuprot tome, ostali hiperparametri (poput regularizacionog hiperparametra ili broja neurona i broja slojeva neuronske mreže) suštinski su vezani za kvalitet generalizacije modela na druge podatke, te se oni biraju na validacionom skupu.

11.9 Ilustracija praktične primene mašinskog učenja

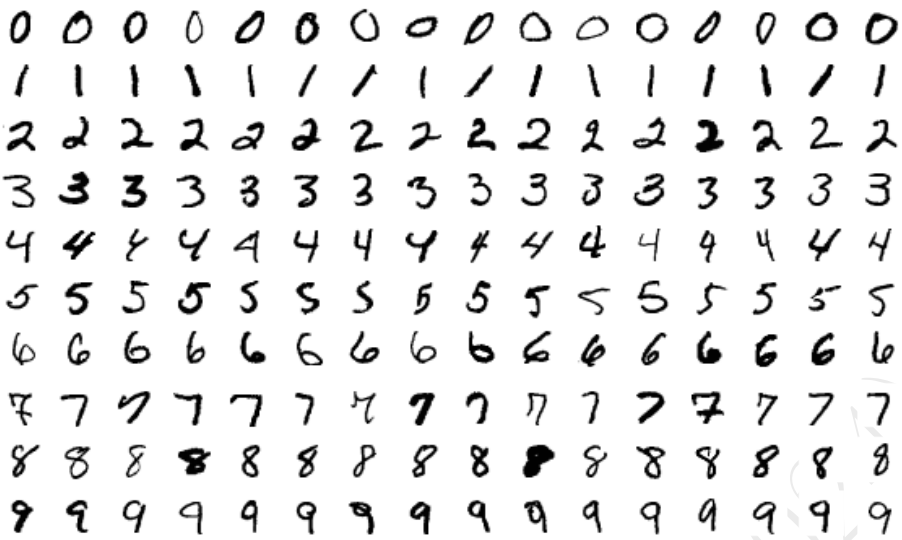
Širenje industrijskih primena metoda mašinskog učenja dovelo je do potrebe za olakšavanjem njihovog implementiranja i za nekim vidom standardizacije koda. To je dovelo do razvoja velikog broja biblioteka koje olakšavaju rad sa metodama mašinskog učenja, zahvaljujući kojima korisnici sve ređe sâmi implementiraju izabrane algoritme. Umesto toga, koriste se gotove implementacije i korisnici se fokusiraju na rešavanje i modelovanje problema na nešto apstraktnijem nivou. Naredna diskusija i primer koji sledi stavljaju akcenat na metode nadgledanog učenja i neuronske mreže, ali situacija je vrlo slična i u drugim oblastima mašinskog učenja.

Jedna od najšire korišćenih biblioteka za mašinsko učenje je biblioteka `scikit-learn` koja pruža mogućnost korisnicima da na jednostavan način, često u svega nekoliko linija, primene širok spektar algoritama i pomoćnih alata. Ipak, ta jednostavnost dolazi i sa ograničenjima koja su posebno vidljiva u radu sa neuronskim mrežama, te su međuvremenu razvijene i druge, složenije i fleksibilnije biblioteke.

Dostupne biblioteke za trening neuronskih mreža značajno variraju po nivou apstrakcije koji omogućavaju korisniku. Neke od njih, poput pomenute biblioteke `scikit-learn`, pokušavaju da definisanje mreže maksimalno pojednostave, po cenu značajnih ograničenja fleksibilnosti. U takvim bibliotekama od korisnika se obično očekuje da jednoj ili manjem broju funkcija koje vrše trening, kao argumente prosledi informacije o arhitekturi mreže (poput broja slojeva, broja neurona po sloju i vrste aktivacione funkcije), osnovne hiperparametre (poput brzine učenja, regularizacionog hiperparametra i drugih), kao i podatke na kojima je potrebno trenirati mrežu. Druge biblioteke, poput biblioteka `TensorFlow` i `PyTorch`, korisniku pružaju maksimalnu fleksibilnost, po cenu potrebe za dubljim razumevanjem modela, strukture biblioteke i obimnijeg pisanja programskog kôda. U slučaju ovakvih biblioteka, korisnik piše programski kôd kojim se definiše arhitektura mreže, mehanizam učitavanja i pretprocesiranja podataka, metoda za optimizaciju i procedura kojom se pomenuti elementi koriste kako bi se izvršio trening, beležile statistike treninga i vršila validacija i testiranje modela. Ovaj pristup ilustrovan je detaljnije u nastavku. Neke biblioteke kao što je `Keras`, pokušavaju da pronađu srednji put između ova dva ekstrema.

Primer treniranja neuronske mreže za klasifikaciju slika. Treniranje neuronske mreže je netrivijalan posao. Podsetimo se — neuronska mreža određena je svojom arhitekturom i vrednostima parametara. Za datu arhitekturu, vrednosti parametara dobijaju se minimizacijom greške. Izbor arhitekture predstavlja veći izazov, pa će u daljem tekstu akcenat biti na njemu. Sve faze treniranja biće ilustrovane kôdom u programskom jeziku `Python` uz korišćenje biblioteke `PyTorch`. Proces treniranja biće ilustrovan u pojednostavljenom obliku na primeru prepoznavanja rukom pisanih cifara.

Za ilustraciju procesa treninga mreže biće korišćen skup MNIST, koji se često koristi za poređenje različitih modela mašinskog učenja. Ovaj skup sadrži deset klasa, sa jednakim brojem slika, koje odgovaraju različitim ciframa. Slike su crno-bele, dimenzija 28×28, ali se sâme cifre uklapaju u kvadrate dimenzija 20×20 u središtu



Slika 11.11: Uzorak skupa podataka MNIST.

slike. Skup se sastoji se od 60000 instanci za trening i 10000 instanci za testiranje. Uzorak skupa može se videti na slici 11.11.

Podatke je obično potrebno učitati i na neki način transformisati u pogodan oblik. Naredni deo koda prikazuje jednostavan način da se sa interneta dopreme MNIST podaci, da se konvertuju u `PyTorch` tenzore i da se transformišu tako da vrednosti piksela imaju prosek 0 i standardnu devijaciju 1.

```
transform = Compose([transforms.ToTensor(),
                     transforms.Normalize((0), (1))])

train_set = MNIST(root='./data',
                  train=True,
                  download=True,
                  transform=transform)
test_set = MNIST(root='./data',
                 train=False,
                 download=True,
                 transform=transform)
```

Izbor arhitekture i njeno podešavanje nosi sa sobom ozbiljnu opasnost od nepravilne evaluacije modela i najpre ćemo se fokusirati na to pitanje. Kao što je ranije rečeno, model se bira korišćenjem validacionog skupa (a ne korišćenjem test skupa). Validacioni skup kreira se kao reprezentativan podskup polaznog trening skupa od 60000 instanci: novi trening skup činiće 50000 instanci, a validacioni skup činiće 10000 preostalih instanci. Ova podela sprovodi se slučajnim izborom instanci, ali koristeći stratifikaciju, tako da svaka klasa ima jednako instanci i u trening i u validacionom skupu.

```
targets = train_set.targets
indices = np.arange(len(targets))
train_idx, valid_idx = train_test_split(indices,
                                        test_size=1/6,
                                        shuffle=True,
                                        stratify=targets)
```

Funkcija `train_test_split` vraća indekse instanci (a ne sâme instance) iz polaznog skupu koje će činiti novi trening i validacioni skup. Poslednji korak koji se tiče pripreme podataka je definisanje objekata koji će vršiti učitavanje podataka u radnu memoriju u toku treninga. Usled podele trening skupa posredstvom skupova indeksa, definisanje ovih objekata za trening i validacioni skup je malo komplikovanije nego za test skup.

```
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

train_loader = DataLoader(train_set,
                          batch_size=50,
                          sampler=train_sampler)
valid_loader = DataLoader(train_set,
                          batch_size=len(valid_sampler),
                          sampler=valid_sampler)
test_loader = DataLoader(test_set,
                         batch_size=len(test_set))
```

Učitavanje podataka vrši se po zahtevu u glavnoj proceduri treninga i u ovom kodu se pretpostavlja da će biti isporučivani paketi od 50 instanci pri svakom takvom zahtevu. Naime, u kontekstu neuronskih mreža uobičajeno je da se gradijentni spust ne primenjuje na ceo skup podataka, već se u svakom koraku računa gradijent greške na nekom podskupu podataka. Jedan razlog za to je što veličina podataka može biti prevelika imajući u vidu memorijska ograničenja, ali ima i drugih koji se tiču brzine optimizacije i kvaliteta generalizacije (i u te razloge nećemo ulaziti). Veličina podskupa često se meri desetinama ili stotinama. U našem slučaju, koristiće se veličina 50 (naknadno je provereno da je trening robustan na promene ove veličine, odnosno da se postižu praktično isti rezultati sa nešto većim ili manjim vrednostima). Podskupovi se biraju tako što se instance trening skupa nasumice podele u podskupove zadate veličine i koriste se nekim redom bez ponavljanja dok se ne iskoriste sve. Jedan prolaz kroz sve podatke u toku optimizacije se naziva epohom. Sledeća epoha sastoji se od novog prolaza. Kako se validacioni i test skup ne koriste pri gradijentnom spustu, a mogu da stanu u radnu memoriju, oni se u ovom primeru učitavaju u celosti.

Nameće se pitanje od kakve arhitekture krenuti i kako je modifikovati. Kako se radi o slikama, biće korišćena konvolutivna mreža. Filteri koji se koriste u obradi slika najčešće su dimenzija 3×3 ili 5×5. Probaćemo da koristimo filtere dimenzija 3×3. Mreža bi trebalo da ima nekoliko slojeva kako bi bila u stanju da konstruiše nešto složenije atribute i kako bi neuroni na višim slojevima uspeli da obuhvate informacije sa većeg dela slike. Ukoliko se koriste 3 sloja i nakon svakog se upotrebi agregacija dimenzija 2×2, svi izlazi nakon poslednjeg sloja agregacije će biti dimenzija 1×1 i mogu se proslediti potpuno povezanoj mreži. Zato možemo krenuti od mreže od 3 sloja konvolucija 3×3, praćenih agregacijama 2×2 i ReLU aktivacionim funkcijama. Pitanje je koliko filtera staviti u svaki sloj. Tipično se sa dubinom smanjuju dimenzije matrica koje su izlazi konvolutivnih slojeva, ali se povećava broj filtera kako bi se očuvala informacija iz polazne slike, te se broj filtera često udvostručuje u odnosu na prethodni sloj. Krenućemo od minimalističkog dizajna i pretpostaviti da prvi sloj ima 2 filtera, drugi 4 i treći 8. Potpuno povezani sloj će biti jedan i uvek će imati onoliko ulaza koliko ima filtera na poslednjem sloju i 10 izlaza, pošto postoji 10 cifara. Ovakva arhitektura verovatno je premala za bilo kakve praktične primene i ne očekujemo da može da se dovoljno prilagodi podacima, ali može poslužiti kao polazna tačka u ovom jednostavnom primeru. Svaka sledeća arhitektura koju budemo isprobali će imati dvostruko više filtera u prvom sloju nego prethodna, a svaki njen sloj će imati dvostruko više filtera nego prethodni. Arhitekturu ćemo birati tako što ćemo pratiti tačnost modela na validacionom skupu i povećavati arhitekturu dok dobijena tačnost raste. Na kraju ćemo se opredeliti za arhitekturu koja postiže maksimalnu tačnost na validacionom skupu. Tako izabranu arhitekturu na kraju ćemo evaluirati na test skupu.

Polazna arhitektura definisana je narednim kodom. U konstruktoru se definišu konvolutivni slojevi, slojevi agregacije maksimumom i potpuno povezani sloj. Konvolutivni slojevi se definišu navođenjem broja ulaznih kanala, broja izlaznih kanala i veličine kernela (u ovom slučaju uvek 3×3). Slojevi agregacije definišu se navođenjem dimenzija agregacije (u ovom slučaju uvek 2×2). Potpuno povezani sloj slika vrednosti poslednjih 8 kanala u 10 novih vrednosti. U funkciji `forward` definiše se izračuavanje koje mreža vrši. Pored primena slojeva definisanih u konstruktuoru primećuje se i primena ReLU atkviacione funkcije. Primetimo da se funkcija softmax ne primenjuje u datom kodu. Naime, nju biblioteka sâma izračunava pri izračunavanju unakrsne entropije kao funkcije greške.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 2, 3)
        self.conv2 = nn.Conv2d(2, 4, 3)
        self.conv3 = nn.Conv2d(4, 8, 3)
        self.pool1 = nn.MaxPool2d(2, 2)
```

```
            self.pool2 = nn.MaxPool2d(2, 2)
            self.pool3 = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(8, 10)

        def forward(self, x):
            x = self.pool1(F.relu(self.conv1(x)))
            x = self.pool2(F.relu(self.conv2(x)))
            x = self.pool3(F.relu(self.conv3(x)))
            x = x.view(-1, 8)
            x = self.fc1(x)
            return x
```

Naredni kôd instancira mrežu, funkciju greške i metod optimizacije. Kao funkciju greške koristimo uobičajeni izbor unakrsne entropije. „Adam“ je u praksi često korišćeno unapređenje gradijentnog spusta. Potrebno je proslediti mu parametre mreže i brzinu učenja.

```
net = Net()
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

Brzina učenja je vrlo bitan hiperparametar i u eksperimentima nećemo koristiti jednu fiksiranu vrednost kao u ovom primeru koda, već ćemo je birati empirijski, isprobavši različite vrednosti ovog hiperparametra. Konačno će biti korišćena ona koja najbrže smanjuje vrednost greške. Detalji će biti prikazani kasnije.

Evaluacija modela vrši se kako na validacionom skupu u toku treninga, tako i na test skupu na kraju celokupnog procesa treninga. U ovom primeru evaluacija se sastoji od izračunavanja tačnosti (definisane u poglavlju 11.8.2). Imajući u vidu da su sve klase jednako bitne i da su sve podjednako zastupljene u sva tri skupa, tačnost je adekvatna mera kvaliteta klasifikacije. Naredni kod izračunava tačnost na nekom skupu podataka. Kontekst `torch.no_grad()` govori da se izračunavanja vrše bez čuvanja informacija koje algoritmu propagacije unazad služe da izračuna gradijent (što se u toku validacije i testiranja i ne radi).

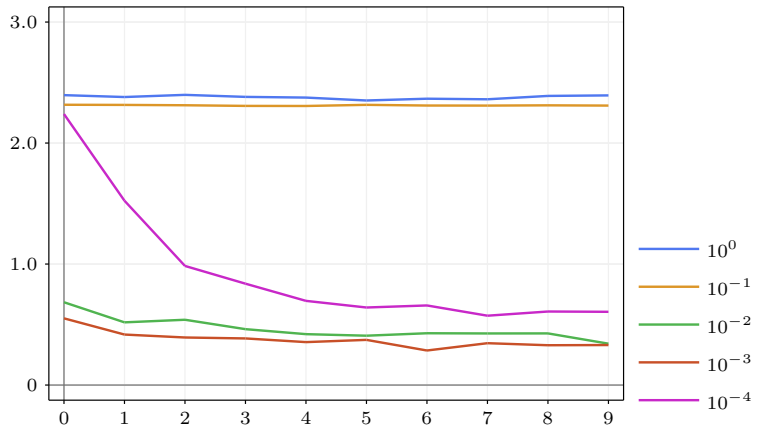
```
def accuracy(model, data_loader):
    images, labels = next(iter(data_loader))
    with torch.no_grad():
        outputs = model(images)
    _, predicted = torch.max(outputs.data, dim=1)
    correct = (predicted == labels).sum().item()
    return 100 * correct / labels.size(0)
```

Naredni kôd prikazuje proceduru treninga. Koristićemo 10 epoha. Ako bi se uočilo da tačnost i dalje raste u poslednjim epohama, trening bi mogao biti produžen. U svakoj epohi prolazi se kroz ceo trening skup i za svaki od podskupova na kojima se vrši trening mreže radi se sledeće: anulira se tekuća vrednost gradijenta, izračunavaju se izlazne vrednosti mreže, izračunava se greška, poziva se algoritam propagacije unazad i vrši se korak optimizacije. Nakon što je trening u datoj epohi završen, pre prelaska na sledeću, vrši se validacija.

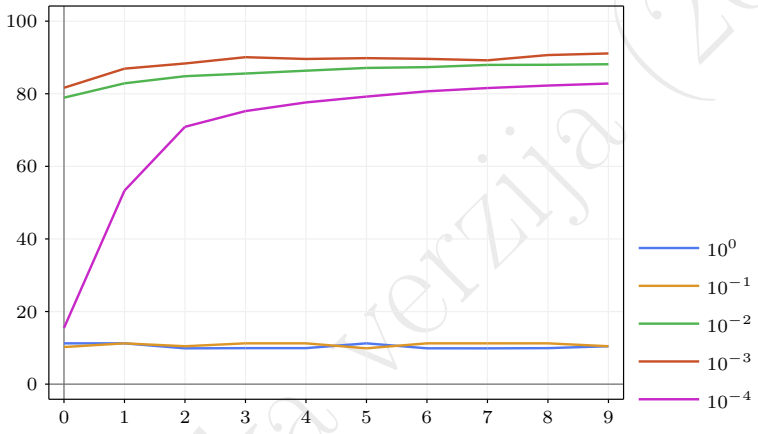
```
for epoch in range(10):
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()

    print(accuracy(net, valid_loader))
```

Na kraju celokupnog treninga, što uključuje izbor arhitekture i svih hiperparametara, testiranje se vrši naredbom `print(accuracy(net, test_loader))`.

Varirajmo brzinu učenja za polaznu arhitekturu sa dva filtera u prvom sloju. Vrednosti poput 1 su tipično vrlo visoke, pa ćemo isprobati niz vrednosti 1, 0.1, 0.01, 0.001 i 0.0001, a ako bude potrebe, možemo nastaviti dalje. Na slici 11.12 prikazano je kako se smanjuje vrednost funkcije greške na trening skupu kroz 10 epoha za različite brzine učenja. Zarad računske efikasnosti treninga, vrednost greške e_t se u svakom koraku t izračunava



Slika 11.12: Ponašanje funkcije greške na trening skupu kroz 10 epoha treninga za različite brzine učenja.

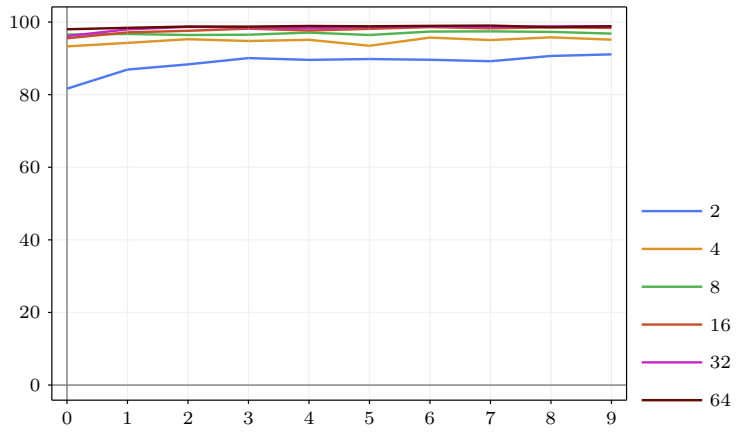


Slika 11.13: Ponašanje tačnosti na validacionom skupu kroz 10 epoha treninga za različite brzine učenja.

samo na tekućem podskupu trening skupa (u nasem slučaju velicine 50). Takve vrednosti se uprosečavaju u svakom koraku tako što se od tekućeg proseka m_{t-1} izračunava novi po formuli $m_t = 0.9m_{t-1} + 0.1e_t$. Na grafiku je prikazana ova vrednost na kraju svake epohe. Primećuje se da su vrednosti 1 i 0.1 toliko velike da se greška ne smanjuje. Za sve ostale, greška se smanjuje, ali je vrednost 0.001 najbolja. Primetimo da je u slučaju vrednosti 0.0001 optimizacija sporija i stoga nećemo ni isprobavati manje vrednosti. U ovom primeru, brzina učenja birana je na osnovu rezultata na trening skupu, a ne na validacionom skupu, za razliku od izbora drugih hiperparametara. Ova praksa diskutovana je u poglavlju 11.8. Na slici 11.13 je prikazano i ponašanje tačnosti na validacionom skupu, ali zaključak o brzini učenja je isti.

Za svaku veličinu arhitekture koju ćemo isprobati, koristićemo prethodni način izbora brzine učenja i taj postupak nadalje nećemo komentarisati. Na slici 11.14 prikazano je ponašanje tačnosti na validacionom skupu kroz 10 epoha za različite veličine arhitekture. U svim slučajevima osim za najmanju arhitekturu, učenje je završavano u 10 epoha. Arhitekture sa 32 i 64 filtera u prvom sloju očito se ponašaju najbolje, ali se sa povećanem sa 32 na 64 ne dobija na tačnosti. Isti rezultat — 98.85%, dobija se i sa 128 filtera u prvom sloju, tako da ćemo se odlučiti za 32. Konačno, evaluiranjem na test skupu modela naučenog sa ovom arhitekturom dobija se tačnost od 98.57%. Ona odgovara tačnosti sa validacionog skupa, što znači da izabrani model dobro generalizuje.

Treba imati u vidu da je prethodni postupak pojednostavljen i da je za rešavanje realnih problema, ipak potrebno više znanja i iskustva. Najčešće se umesto samostalnog dizajna arhitekture koriste već postojeće arhitekture, koje su često definisane u različitim veličinama, tako da je moguće, u prethodno opisanom duhu, birati jednu od njih. Često se koriste i modeli koji su već istrenirani na nekom bogatom skupu slika, pa se samo prilagođavaju skupu kojim raspolaže korisnik. Slična je situacija i u slučaju drugih vrsta podataka.



Slika 11.14: Ponašanje tačnosti na validacionom skupu kroz 10 epoha treninga za različite veličine arhitekture. Brojevi u legendi predstavljaju broj konvolutivnih filtera u prvom sloju mreže.

11.10 Primeri primena nadgledanog učenja

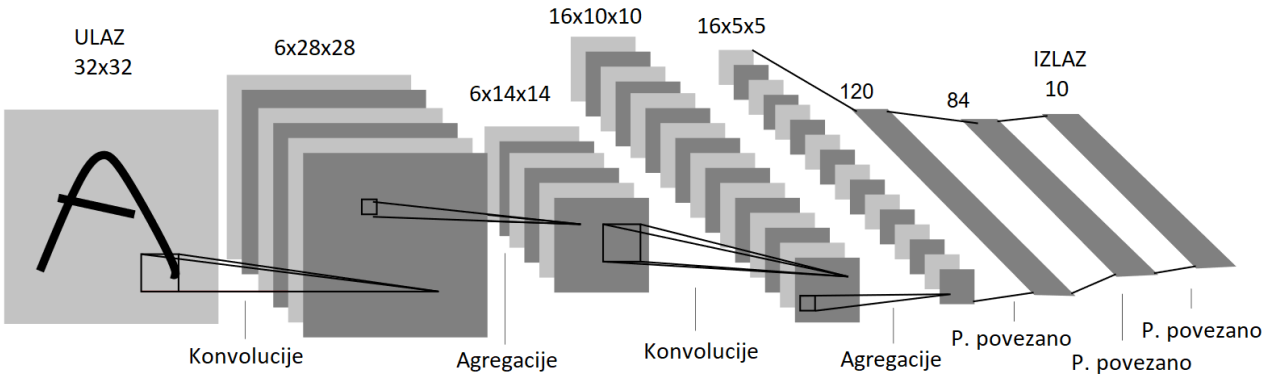
Primeri praktičnih primena nadgledanog učenja su u poslednjoj deceniji sve brojniji. Najuzbudljivije i praktično najvažnije primene odnose se na neuronske mreže, pa će i akcenat u daljem tekstu biti na njima, pre svega na obradi slika i teksta. Ipak, biće diskutovane i primene drugih metoda.

11.10.1 Prepoznavanje rukom pisanih cifara

Problem prepoznavanja rukom pisanih cifara predstavlja „*hello world*“ problem za neuronske mreže. On je specijalan slučaj opštijeg problema prepoznavanja rukom pisanog teksta i važan je u mnogim kontekstima, poput digitalizacije pisanog teksta, prepoznavanja teksta na tabletima i slično. Kao skup podataka u eksperimentima vezanim za ovaj problem obično se koristi već pomenuti skup MNIST sa 60000 slika za trening i 10000 slika za testiranje, dimenzija 28×28.

Na ovom skupu testirano je, tokom prethodnih godina, mnogo različitih tehnika mašinskog učenja. Rani, jednostavniji pristupi, pravili su relativno velike greške. Na primer, neuronska mreža bez skrivenih slojeva imala je tačnost od 88%, a sa dodatnim pretprocesiranjima podataka tačnost 91.6%. Metod k najbližih suseda sa euklidskim rastojanjem postigao je 95% tačnosti, a uz određeno pretprocesiranje podataka i izmenu metrike čak 98.8%. Potpuno povezana mreža sa 5 skrivenih slojeva (sa 500 do 2500 neurona po sloju) i dodatnim pretprocesiranjima podataka postigla je tačnost od 99.6%. Od potpuno povezanih mreža sa jednim ili dva skrivena sloja može se očekivati tačnost od oko 98%. Konvolutivne neuronske mreže lako postižu tačnost od preko 99% i bez pretprocesiranja podataka, što je i očekivano imajući u vidu da su namenjene obradi slika. Najbolji rezultat od 99.8% prikazan na zvaničnoj strani MNIST-a postignut je od strane skupa od 35 dubokih konvolutivnih mreža. Kako različite mreže mogu predvideti različite klase, bira se ona koju je predvidelo najviše mreža. Imajući u vidu dostignute procente, može se smatrati da je ovaj problem lak. Ipak, treba imati u vidu da je skup podataka pažljivo sastavljen i da bi u realističnijem kontekstu bio teži. Na primer, prepoznavanje cifara sa fotografija papira pri različitim osvetljenjima, odstojanjima i slično sigurno bi predstavljalo veći izazov.

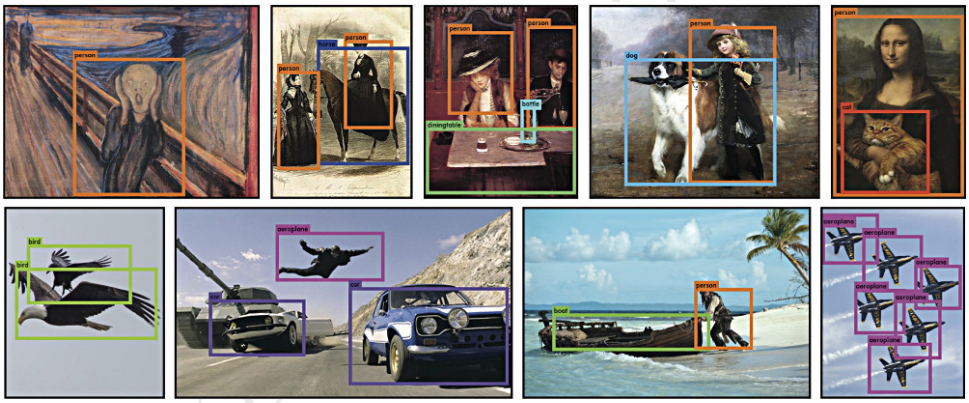
Razmotrimo primer jedne konkretne arhitekture, nešto pojednostavljenu varijaciju mreže LeNet-5 iz 1998. godine. Ova arhitektura prikazana je shematski na slici 11.15. Sastoji se od dva konvolutivna sloja isprepletana sa dva sloja agregacije, iza kojih slede tri potpuno povezana sloja. Prvi konvolutivni sloj sastoji se od 6 filtera dimenzija 5×5, a drugi od 16 filtera istih dimenzija. Oba sloja agregacije vrše agregaciju uprosečevanjem dimenzija 2×2, tako da su njihovi izlazi dvostruko manjih dimenzija nego ulazi. Potpuno povezani slojevi imaju 120, 84 i 10 neurona. Aktivacione funkcije na svim nivoima osim poslednjeg su hiperbolički tangens, a na poslednjem softmax. Primetimo da su dimenzije ulaza 32×32, iako su slike u MNIST skupu dimenzija 28×28, a unutar njih sâm broj staje u centralnu oblast dimenzija 20×20. Postavlja se pitanje zašto je bilo potrebno ovakvo proširivanje ulaza. Svaki neuron u prvom nivou slike „vidi” 5×5 vrednosti iz ulazne slike. Neuroni koji su u slojevima bližim izlazu „vide” veće delove slike jer vrše izračunavanja nad izlazima više neurona prethodnog sloja. Deo slike koji neuron „vidi” naziva se *receptivno polje*. Proširivanje slike na dimenzije 32×32 vrši se kako bi se osiguralo da je za svaki sloj, svaki piksel iz centralnog kvadrata dimenzija 20×20 u centru receptivnog polja nekog neurona. Ova arhitektura postigla je tačnost od 98.6%. Originalna, nešto komplikovanija, meža LeNet-5, postigla je tačnost od 99.1%



Slika 11.15: Arhitektura mreže LeNet-5.

11.10.2 Detekcija objekata na slikama

Prepoznavanje objekata na slikama je problem od velikog praktičnog značaja. U autonomnoj vožnji bitno je prepoznati druge učesnike u saobraćaju, saobraćajne znake i slično. Prepoznavanje lica može služiti identifikaciji ljudi u različite svrhe (koje se mogu smatrati legitimnim ili ne). Pristupi rešavanju ovih problema su raznovrsni. Ovde će biti diskutovan jedan pristup, zasnovan na konvolutivnim neuronskim mrežama, poznatiji kao YOLO (*you only look once*). YOLO je poznat po svojoj brzini i preciznosti.[13] Primer detekcije objekata dat je na slici 11.16.



Slika 11.16: Detekcija objekata na slici pomoću YOLO mreže.

Opišimo najpre kako funkcioniše već trenirana mreža. Slika se deli na $S \times S$ disjunktnih pravougaonih ćelija, kao što je prikazano na slici 11.17 (levo). Za detekciju objekta zadužena je ćelija u kojoj je centar tog objekta. Za ćeliju je vezan konvolutivni klasifikator kojim se objekat prepoznat u toj ćeliji klasifikuje u jednu od klasa, sa pratećom verovatnoćom. Klasifikacija je ilustrovana slikom 11.17 (dole). Za svaku ćeliju vezan je i unapred definisan broj konvolutivnih mreža za regresiju čija je svrha da predvide pouzdanost detekcije objekta (koja zavisi od verovatnoće koju daje klasifikator), kao i pravougaonik (stranica paralelnih koordinatnim osama) koji sadrži ceo objekat od značaja. Taj pravougaonik predviđa se predviđanjem x i y koordinata centra, širine i visine. Predviđeni pravougaonici prikazani su na slici 11.17 (gore). Mreža predviđa skor koji odražava ukupnu pouzdanost detekcije i koji je jednak proizvodu sledeće dve veličine:

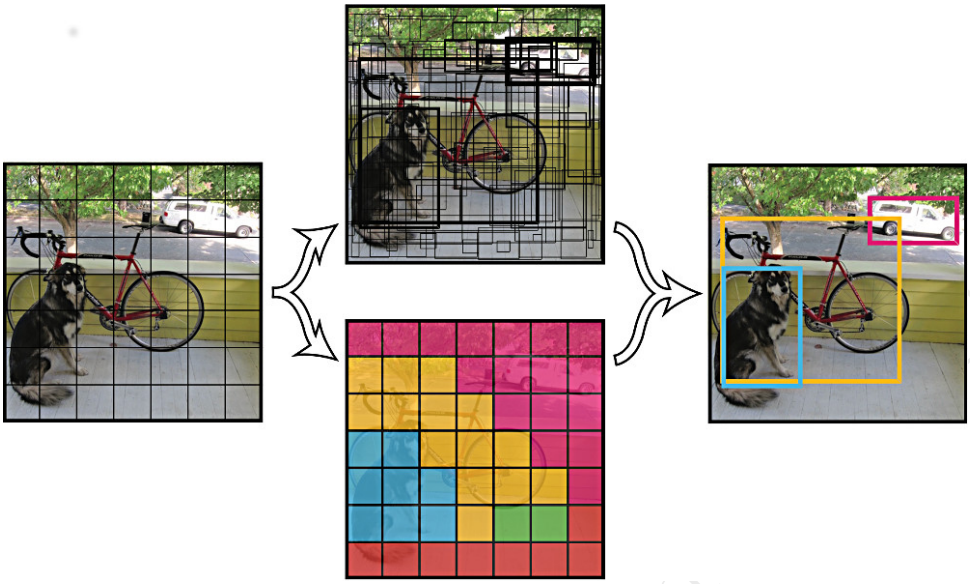
- verovatnoća koju daje klasifikator – ova veličina odražava sigurnost klasifikatora u predloženu klasu objekta;
- IoU skor[14] za predviđeni pravougaonik i stvarni pravougaonik koji obuhvata objekat na slici – ova veličina odražava preciznost detekcije.

Ukoliko ovaj skor pouzdanosti detekcije ima veću vrednost od nekog unapred definisanog praga, smatra se da je detekcija uspešna. U suprotnom, pravougaonik kojem ovaj skor odgovara se zanemaruje. Očigledno je da broj

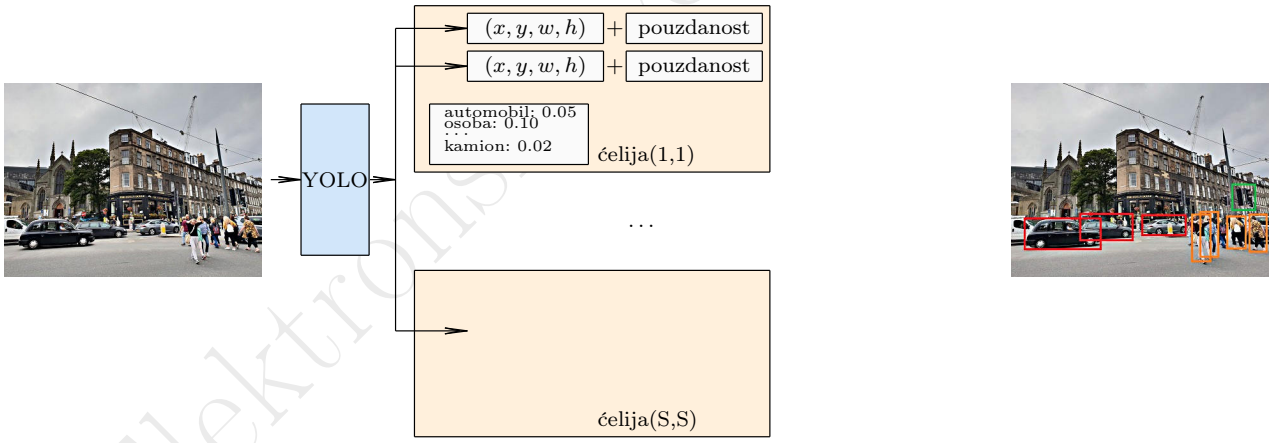
[13] Ovaj pristup ima više varijanti. U ovom opisu akcenat nije na njegovim najmodernijim detaljima, već na ključnim idejama.

[14] IoU skor (eng. *intersection over union*) za dva pravougaonika jednak je količniku površine njihovog preseka i površine njihove unije.

objekata koje YOLO može da detektuje zavisi od broja ćelija $S \times S$, pa je taj broj značajan hiperparametar. Primer pravougaonika vezanih za ćeliju može se videti na slici 11.17 (levo), a gruba shema modela vezanog za ćeliju na slici 11.18.



Slika 11.17: Ilustracija rada YOLO mreže. Levo je prikazana ulazna slika izdeljena na ćelije. Dole su različitim bojama prikazane različite klase objekata koji se prepoznaju u svakoj ćeliji. Gore su prikazana predviđanja svih pravougaonika koji obuhvataju objekte, pritom se značajno preklapajući. Desno je prikazan finalni izlaz nakon izbora manjeg broja relevantnih pravougaonika.



Slika 11.18: Shema modela pridruženog ćeliji.

Pri primeni modela, dolazi do značajnog preklapanja velikog broja predviđenih pravougaonika (kao što prikazuje slika 11.17) koji mogu prepoznavati iste ili različite objekte, sa različitim pouzdanostima. Heuristika koja se koristi kako bi se otklonio ovaj problem podrazumeva sortiranje predloženih pravougaonika po pouzdanosti i njihovu selekciju na sledeći način. Bira se i zadržava pravougaonik sa najvećom pouzdanošću. Odbacuju se svi pravougaonici čiji je IoU skor sa tim pravougaonikom veći od nekog unapred odabranog praga. Analogan postupak se nastavlja nad preostalim pravougaonicima sve dok taj skup nije prazan. Izabrani pravougaonici čine predviđanje modela i ilustrovani su slikom 11.17 (desno).

Postavlja se pitanje kako ovako složene modele treba trenirati. Za svaki od modela treba da budu poznate ciljne vrednosti na svakoj slici i za svaki treba da bude definisana greška na sledeći način.

- Za trening klasifikacionih modela potrebno je da za svaki objekat na svakoj od slika bude unapred obeležena klasa (na primer, od strane čoveka). Za svaki objekat na slici, klasifikatoru ćelije koja sadrži centar tog objekta pridružuje se kao greška unakrsna entropija njegovog predviđanja i stvarne klase tog objekta (na način opisan u poglavlju 11.5.2). Za datu sliku, klasifikatori vezani za ćelije koje ne sadrže centar nijednog objekta po definiciji imaju grešku 0.

- Kako bi se trenirao regresioni model koji predviđaja koordinate centra i dimenzije pravougaonika, potrebno je da za svaki objekat na slici budu obeležene ove vrednosti. Za svaki tako obeležen pravougaonik P, odgovaran je regresioni model iz odgovarajuće ćelije čiji izlaz je pravougaonik koji se najviše preklapa sa P. Greška tog regresionog modela je suma kvadratnih grešaka između 4 obeležene i 4 predviđene vrednosti koje definišu pravougaonik. Modeli koji nisu odgovorni ni za jedan obeleženi pravougaonik po definiciji imaju grešku 0.

- Za regresione modele koji predviđaju pouzdanost, ciljna promenljiva se za svaki obeleženi pravougaonik definiše kao proizvod naredne dve vrednosti. Prva je verovatnoća najverovatnije klase prema predviđanju klasifikatora u ćeliji koja odgovara obeleženom pravougaoniku P. Druga je IoU skor između pravougaonika P i pravougaonika koji je rezultat modela odgovornog za P. Greška tih regresionih modela je suma kvadratnih grešaka između tog proizvoda i predviđene vrednosti. Modelima ćelija koje ne odgovaraju nijednom objektu pridružuje se greška 0.

Ukupna greška se dobija sabiranjem svih navedenih grešaka po svim slikama iz trening skupa, uz određene težinske koeficijente koji se mogu podešavati kako bi se dobili bolji rezultati. Tako dobijena greška minimizuje se gradijentnim metodama optimizacije. U ovom opisu nismo diskutovali konkretnu arhitekturu mreže pošto detalji arhitekture nisu jednoznačno definisani.

11.10.3 Prepoznavanje poze sa slika

Jedna od primena konvolutivnih mreža je i prepoznavanje poza u kojima se osobe nalaze na slikama. Primer prepoznatih poza dat je na slici 11.19. Jedan od ključnih elemenata u rešavanju ovog problema je određivanje pozicija zglobova. Ipak, pozicije zglobova nisu dovoljne za određivanje poze. Naime, pitanje je koji zglob treba povezati sa kojim. Čak i u slučaju da se na slici nalazi samo jedna osoba i da su poznate lokacije svih njenih zglobova, iako je jasno da treba povezati njeno koleno sa člankom, u nekim pozama članak jedne noge može biti bliži kolenu druge, pa može doći do greške. U slučaju da slika sadrži veći broj osoba, ovaj problem još je izraženiji. Zato je, pored detektovanja zglobova, potrebno predvideti i orijentacije delova tela (podlaktice, potkolenice, itd).

Metod OpenPose vrši prepoznavanje poze tela u dve faze. U prvoj fazi određuje pozicije zglobova i orijentacije delova tela, a u drugoj fazi povezuje zglobove i tako vrši prepoznavanje poze. Na mašinskom učenju zasnovana je ova prva faza i ona će biti grubo opisana u nastavku. Koristi se konvolutivna mreža koja se trenira na mnoštvu slika i ručno napravljenih izlaza — slika na kojima su pogodno označene pozicije zglobova i orijentacije delova tela. Za svaku novu ulaznu sliku, model daje dve vrste izlaza: *mape pouzdanosti* i *vektorska polja*. Pozicije zglobova određuju se mapama pouzdanosti, a orijentacije delova tela vektorskim poljima. Postoji po jedna mapa pouzdanosti za svaku vrstu zgloba i to je slika istih dimenzija kao što je ulazna. Konkretno, mapa koja odgovara levom laktu detekuje leve laktove svih ljudi na zadatoj ulaznoj slici, a mapa koja odgovara levom ramenu detektuje sva leva ramena na zadatoj ulaznoj slici. Mapa pouzdanosti ima visoku vrednost tamo gde je najverovatnije da se na slici nalazi konkretan zglob, a nisku tamo gde to nije verovatno. Postoji po jedno vektorsko polje za svaki deo tela (na primer, desna podlaktica) i ono svakom pikselu ulazne slike pridružuje dvodimenzioni vektor koji predstavlja koordinate vektora koji određuje kako je orijentisan deo tela kojem taj piksel pripada. Ovo je ilustrovano na slici 11.20.

Kao funkcija greške za obe vrste predviđanja koristi se kvadratna greška između predviđene i stvarne vrednosti kako za mapu pouzdanosti, tako i za vektorska polja koja određuju orijentaciju delova tela.

U drugoj fazi, metod OpenPose na osnovu dobijenih mapa pouzdanosti i vektorskih polja određuje šta se sa čime povezuje. To nije trivijalno uraditi čak ni kad su raspoložive savršene mape pouzdanosti i vektorska polja. Ovaj problem je NP-težak problem kombinatorne optimizacije (gde je dimenzija problema ukupan broj zglobova svih osoba na slici). Za njega postoji efikasno aproksimativno rešenje čiji je kvalitet dovoljan za praktične potrebe, ali neće biti ovde opisan.

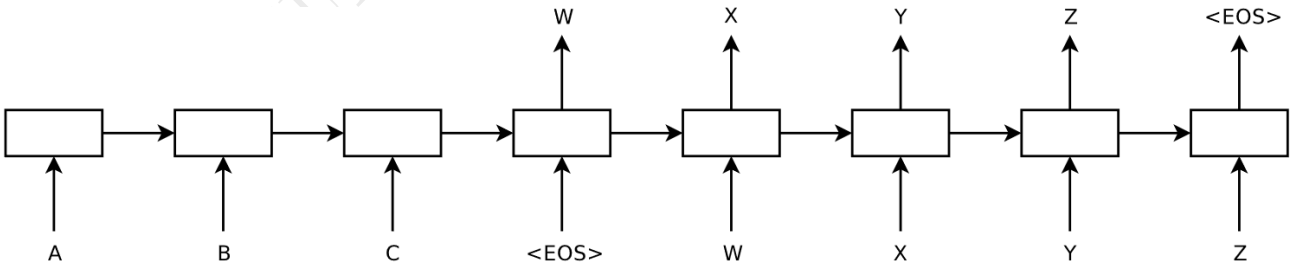
11.10.4 Mašinsko prevođenje i srodni problemi

Rekurentne mreže najčešće nalaze svoju primenu u obradi prirodnog jezika. Jedna od najzanimljivijih takvih primena je mašinsko prevođenje teksta sa jednog jezika na drugi. Prvo će biti opisan jedan od ranih načina na koji se vrši mašinsko prevođenje pomoću rekurentnih mreža, a onda će biti dat osvrt na to kako se na sličan način mogu rešiti problemi sumiranja teksta i opisivanja slika tekstom.

Osnova mašinskog prevođenja pomoću neuronskih mreža je enkoder-dekoder arhitektura. Sastoji se od dve rekurentne mreže. Prva se naziva enkoderom i uloga joj je da pročita ulaznu rečenicu (uključujući znak za kraj rečenice) i da u poslednjem koraku pruži njenu reprezentaciju u vidu vektora fiksne dimenzije – svog skrivenog stanja. Druga se naziva dekoder i uloga joj je da polazeći od tog vektora kao svog inicijalnog skrivenog

Slika 11.19: Primer prepoznavanja poza na slici.

Slika 11.20: Primer ulazne slike (levo), mapa pouzdanosti za dve vrste zgloba (gore desno) i vektorskog polja koje definiše orijentaciju dela tela koji ih povezuje (dole desno).



Slika 11.21: Ilustracija enkoder-dekoder arhitekture. Slova A, B i C predstavljaju ulazne reči, a W, X, Y i Z izlazne reči, a EOS kraj sekvence.

stanja, generiše prevod polazne rečenice na drugi jezik (uključujući znak za kraj rečenice). Ova arhitektura (u razmotanom obliku) ilustrovana je na slici 11.21.

Ako je $\mathbf{x}_1, \ldots, \mathbf{x}_T$ polazna rečenica, $\mathbf{c}$ reprezentacija generisana od strane enkodera, a $\mathbf{y}_1, \ldots, \mathbf{y}_{T'}$ tačan prevod, učenje se vrši maksimizovanjem verodostojnosti parametara $\mathbf{w}$:

$$P_{\mathbf{w}}(\mathbf{y}_1, \ldots, \mathbf{y}_{T'} | \mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T'} P_{\mathbf{w}}(\mathbf{y}_t | \mathbf{c}, \mathbf{y}_1, \ldots, \mathbf{y}_{t-1})$$

Pritom, raspodela $P_{\mathbf{w}}(\mathbf{y}_t|\mathbf{c}, \mathbf{y}_1, \ldots, \mathbf{y}_{t-1})$ se definiše pomoću funkcije softmax nad izlazima mreže. Naravno, kao i ranije, ne maksimizuje se verodostojnost, već se minimizuje negativna vrednost logaritma verodostojnosti u odnosu na neki skup parova rečenica na dva različita jezika.

U ranim eksperimentima sa ovim modelom, ustanovljeno je da lakše uči kada se obrne redosled reči ulazne rečenice, pošto je za generisanje prvih reči prevoda potrebno imati informacije iz prvih reči ulazne rečenice, koje su pouzdanije sačuvane u skrivenom stanju enkodera ukoliko se te reči pojave na kraju. Naravno, ovo znači da će generisanje poslednjih reči prevoda biti teže. Treba imati u vidu da je ovaj problem svojstven i čoveku. Naime, kada pročitamo rečenicu na jednom jeziku, ako je ona duga i komplikovana, nije nam lako da je celu prevedemo odjednom na drugi jezik. Obično ćemo u toku prevođenja s vremena na vreme obraćati pažnju na različite delove ulazne rečenice dok ne kompletiramo prevod. Ovo je inspirisalo *mehanizam pažnje* (eng. *attention*) koji danas predstavlja nezaobilazan deo neuronskih modela za obradu prirodnog jezika. Objasnićemo ga detaljnije. Neka su $\mathbf{h}_1, \ldots, \mathbf{h}_T$ skrivena stanja enkodera. Prethodni model koristi samo poslednje stanje enkodera i koristi ga da inicijalizuje prvo stanje dekodera. Umesto toga, mehanizam pažnje pretpostavlja da se izlaz generiše na osnovu stanja dekodera $\mathbf{s}_i$ i vektora konteksta $\mathbf{c}_i$ koji se računa u svakom koraku na sledeći način:

$$\mathbf{c}_i = \sum_{j=1}^{T} \frac{e^{a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j)}}{\sum_{k=1}^{T} e^{a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_k)}} \mathbf{h}_j$$

Primetimo da su vrednosti

$$\frac{e^{a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j)}}{\sum_{k=1}^{T} e^{a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_k)}}$$

nenegativne i da u zbiru daju vrednost 1. Otud vektor $\mathbf{c}_i$ predstavlja težinski prosek stanja enkodera. Kojim stanjima će biti data veća težina, odnosno na koje delove ulazne rečenice će biti obraćena pažnja, zavisi od vrednosti $a_{\mathbf{w}}(\mathbf{s}_i, \mathbf{h}_j)$. Funkcija $a_{\mathbf{w}}$ predstavlja potpuno povezanu neuronsku mrežu čija je uloga da prepozna relevantna stanja enkodera $\mathbf{h}_j$ za dato stanje dekodera $\mathbf{s}_i$ i da im pridruži veću vrednost. Kako sada dekoder ima pristup svim skrivenim stanjima enkodera i ima mehanizam njihovog izbora (mehanizam pažnje) u svakom koraku, posao enkodera je značajno olakšan – više nije neophodno da sve relevantne informacije, uključujući one sa početka rečenice, budu sačuvane u jednom vektoru.

U izvornoj implementaciji ovog pristupa, za treniranje je korišćen dvojezični paralelni korpus koji svaku rečenicu uparuje sa njenim prevodom. Ukupna veličina korpusa bila je 348 miliona reči. Korpusi pomoću kojih se treniraju najnoviji modeli za prevođenje i razne druge poslove obrade prirodnog jezika su značajno veći.

Vrlo slični modeli predstavljaju osnovu i za problem sumiranja teksta – generisanje kratkih sažetaka teksta koji sadrže najvažnije informacije iz njega. Taj problem može se smatrati problemom mašinskog prevođenja na isti jezik sa gubitkom. Otud i sličnost osnovnog mehanizma.

Još jedan srodan problem je opisivanje slika. Naime, o tom problemu može se razmišljati kao o problemu prevođenja sa „jezika slike“ na, recimo, srpski jezik. Pritom se enkoder implementira pomoću konvolutivne mreže koja predstavlja prirodan izbor za obradu slika. Poslednji sloj mreže predstavlja skriveno stanje koje se predaje dekoderu.

Ovi primeri ukazuju na jedan zanimljiv opštiji zaključak – skriveno stanje predstavlja značenje ulaznog sadržaja. Enkoder ekstrahuje to značenje iz ulazne reprezentacije, a dekoder ga transformiše u izlaznu reprezenzaciju. Postoji još argumenata koji opravdavaju razmišljanje o vektorskim reprezenzacijama koje daju skrivena stanja enkodera kao o značenju i javljaju se i u mnogim drugim kontekstima. Na primer, uočeno je da je vektorska aritmetika nad tim reprezentacijama često saglasna sa značenjima ulaznih objekata. Konkretno, ako su $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ vektorske reprezentacije (dobijene iz enkodera) reči Pariz, Francuska, Rim i Italija, ispostavlja se da važi $\mathbf{a} - \mathbf{b} + \mathbf{d} \approx \mathbf{c}$, iako enkoder uopšte nije treniran da prepoznaje relaciju glavni grad, već je treniran na velikom neobeleženom skupu podataka samo pokušavajući da za svaku reč što bolje pogodi reči u njenoj okolini pri svakom njenom pojavljivanu u tekstu. Štaviše, isto zapažanje važi i za mnoge druge relacije i čak je na ovaj način moguće odgovarati na pitanja o analogijama. Ovo i druga slična zapažanja stvorila su novu podoblast obrade prirodnog jezika – razumevanje prirodnog jezika, ali dalja diskusija na tu temu izlazi iz okvira ove knjige.

11.10.5 Klasifikacija teksta algoritmom k najbližih suseda

Mnoge metode mašinskog učenja, pa i algoritam k najbližih suseda, formulisane su tako da se jednostavno primenjuju na numeričke podatke, ali teško na tekst. Rekurentne mreže prirodno rade sa tekstualnim podacima, ali nose mnoštvo tehničkih izazova poput hardverske zahtevnosti i trikova koji olakšavaju trening i, pored toga, traže velike količine podataka. Otud ne moraju u svim primenama biti prvi izbor. Kako bi se primenili stariji i jednostavniji algoritmi, potrebni su načini da se tekstualni podaci predstave u numeričkom obliku. U obradi teksta, proteinskih sekvenci i sličnih podataka, često se za predstavljanje podataka u numeričkom obliku koriste n-gramski profili.

Ako je data niska $S = s_1 s_2 \dots s_N$ nad azbukom Σ, gde je N pozitivan ceo broj, n-gram niske S, za $n \leq N$, je bilo koja podniska susednih simbola dužine n. Na primer, za nisku `sad_ili_nikad`, 1-grami su: `s`, `a`, `d`, `_`, `i`, `l`, `i`, `_`, `n`, `i`, `k`, `a`, `d`. 2-grami su: `sa`, `ad`, `d_`, `_i`, `il`, `li`, `i_`, `_n`, `ni`, `ik`, `ka`, `ad`. 3-grami su: `sad`, `ad_`, `d_i`, `_il`, `ili`, `li_`, `i_n`, `_ni`, `nik`, `ika`, `kad`, itd.

N-gramski profil niske je lista uređenih parova *(n-gram, frekvencija)*, gde su frekvencije izračunate za sve n-grame niske.

Osnovne prednosti korišćenja n-grama su robusnost (na primer, nisu mnogo osetljivi na greške u kucanju ili na pojavljivanje reči u različitim gramatičkim oblicima), nezavisnost od domena koji se analizira, efikasnost (dovoljan je jedan prolaz kroz tekst) i jednostavnost. Mana je eksponencijalna zavisnost broja mogućih n-grama u odnosu na dužinu n-grama.

N-gramski profili uspešno se koriste u različitim primenama koje uključuju prepoznavanje autorstva tekstova, prepoznavanje jezika kojim je tekst pisan, prepoznavanje govora i određene probleme iz oblasti bioinformatike. n-gramski profili često se koriste u sadejstvu sa metodom k najbližih suseda.

Razmotrićemo na konkretnom primeru klasifikaciju tekstova na osnovu jezika. Srpski i engleski jezik biće predstavljeni po jednim kraćim tekstom označenim sa S1 i E1. Pošto se izračunaju frekvencije n-grama za ta dva teksta, njihovi n-gramski profili čine trening skup. Test skup će biti dobijen na osnovu četiri kratka teksta od kojih su dva na srpskom označena sa S2 i S3, a dva na engleskom jeziku označena sa E2 i E3.[15] Klasifikacija će biti izvršena pomoću algoritma jednog najbližeg suseda. U tekstovima na srpskom jeziku nisu korišćena srpska slova kako bi se izbegla laka identifikacija na osnovu pisma.

S1: U prethodnom delu prikazani su teorijski okviri i algoritmi pomocu kojih je moguce sprovoditi logicko zakljucivanje. Iako zakljucci moraju nuzno slediti iz zadatih pretpostavki, proces njihovog dokazaivanja nije pravolinijski vec ukljucuje odredene odluke o pravcu u kome ce se postupak sprovoditi. Drugim recima, uocljivo je traganje za dokazom nekog tvrdenja. Primera radi, u primeni procedure DPLL moguce je uociti i korake zakljucivanja i korake pretrage. Kada se uoci jedinicna klauza u nekoj formuli, njeno zadovoljenje je nuzno i predstavlja korak zakljucivanja. S druge strane kada je nemoguce direktno zakljucivanje, potrebno je pretpostaviti vrednost iskazne promenljive. U daljem toku dokazivanja, ta akcija ce se pokazati kao opravdana ili neopravdana. U slučaju da se pokaze kao neopravdana, preduzima se alternativna akcija. Znaci, situacija u kojoj nije moguce izvrsiti direktno zakljucivanje zahteva primenu pretrage.

Manje apstraktan primer je upravljanje robotskom rukom. Pretpostavimo da robotska ruka ima nekoliko mehanickih zglobova cije se kretanje kontrolise elektricnim impulsima. Pritom, neki zglobovi omogucavaju rotacije samo oko jedne ose (kao ljudski lakat ili zglobovi na pristima), a drugi rotaciju oko veceg broja osa (kao ljudsko rame ili zglobovi u korenu prstiju). Pokret hvatanja case ovakvom robotskom rukom je netrivijalan zadatak, ali se moze razbiti na sekvencu atomicnih koraka — pokreta pojedinacnih zglobova oko razlicitih osa za odreden ugao. Mozemo zamisliti da se ovi koraci izvrsavaju strogo jedan po jedan u kom bi slucaju kretanje ruke bilo znacajno razlicito od ljudskog i sporo, ali bi problem bio laksi jer ne bi bila potrebna sinhronizacija razlicitih zglobova i svaki bi se pojedinacno dovodio u zeljeni polozaj. Druga mogucnost je da se kretanja zglobova izvode simultano, kao kod coveka, pri tom povecavajuci broj mogucih kombinacija u svakom trenutku.

Dati primeri motivisu razmisljanje o pretrazi kao o nalazenju niza akcija kojima se ostvaruje cilj kada to ne moze biti ostvareno pojedinacnim akcijama. Iako u opstem slucaju ovakva definicija ne mora delovati adekvatno, u kontekstu vestacke inteligencije u kome obicno pretpostavljamo postojanje nekog entiteta koji deluje preduzimanjem nekih akcija (agenta), ona je prirodna.

E1: There are two paths to achieving an AGI, says Peter Voss, a software developer and founder of the firm Adaptive A.I. Inc. One way, he says, is to continue developing narrow AI, and the systems will become generally competent. It will become obvious how to do that. When that will happen or how it will come about, whether through simbots or some DARPA challenge or something, I dont know. It would be a combination of those kinds of things. The other approach is to specifically engineer a system that can learn and think. Thats the approach that [my firm] is taking. Absolutely I think thats possible, and I think its closer than most people think five to 10 years, tops. The two approaches outlined by Vosseither tinkering with mundane programs to make them more capable and effective or designing a single comprehensive AGI system speak to the long-standing philosophical feud that lies at the heart of AI research: the war between the neats and the scruffies. J. Storrs Hall, author of Beyond AI: Creating the Conscience of the Machine (Prometheus Books, 2007), reduces this dichotomy to a scientific approach vs. an engineering mind-set. The neats are after a single, elegant solution to the answer of human intelligence, Hall says. Theyre trying to explain the human mind by turning it into a math problem. The

[15] Tekst S1 potiče iz radne verzije ove knjige, S2 je preuzet iz knjige Programiranje 1 Filipa Marića i Predraga Janičića, a S3 sa Vikipedije. Tekst E1 preuzet je iz članka The AI Chasers Patrika Takera, E2 iz Rata Svetova H. Dž. Velsa, a E3 iz Stenfordske enciklopedije filozofije.

scruffies just want to build something, write narrow AI codes, make little machines, little advancements, use whatever is available, and hammer away until something happens. The neat approach descends from computer science in its purest form, particularly the war game studies of Von Neumann and his colleagues in the 1930s and 1940s. The 1997 defeat of world chess champion Garry Kasparov by IBMs Deep Blue computer is considered by many the seminal neat success. Up until that moment, the mainstream scientific community generally accepted the premise that AIs could be written to perform specific tasks reasonably well, but largely resisted the notion of superhuman computing ability. Deep Blue proved that an AI entity could outperform a human at a supposedly human task, perceiving a chess board (Deep Blue could see 200 million board positions per second) and plotting a strategy (74 moves ahead as opposed to 10, the human record).

S2: Precizni postupci za resavanje matematickih problema postojali su u vreme starogrckih matematicara (npr. Euklidov algoritam za odredivanje najveceg zajednickog delioca dva broja), a i pre toga. Ipak, sve do pocetka dvadesetog veka nije se uvidala potreba za preciznim definisanjem pojma algoritma. Tada je, u jeku reforme i novog utemeljivanja matematike, postavljeno pitanje da li postoji algoritam kojim se (pojednostavljeno receno) mogu dokazati sve matematicke teoreme. Da bi se ovaj problem uopste razmatrao, bilo je neophodno najpre definisati (matematicki precizno) sta je to precizan postupak, odnosno sta je to algoritam.

S3: Dositej Obradovic (svetovno ime Dimitrije) (Cakovo, 1744 — Beograd, 1811) je bio srpski prosvetitelj i reformator revolucionarnog perioda nacionalnog budjenja i preporoda. Rodjen je u rumunskom delu Banata tadasnje Austrije. Skolovao se za kaludjera, ali je napustio taj poziv i krenuo na putovanja po celoj Evropi, gde je primio ideje evropskog prosvetiteljstva i racionalizma. Ponesen takvim idejama radio je na prosvecivanju svog naroda, prevodio je razna dela medju kojima su najpoznatije Ezopove basne, a potom je i sam pisao dela, prvenstveno programskog tipa, medju kojima je najpoznatije „Zivot i prikljucenija“. Dositej je bio prvi popecitelj (ministar) prosvete u Sovjetu i tvorac svecane pesme „Vostani Serbie“. Njegovi ostaci pocivaju u Beogradu, na ulazu u Sabornu crkvu.

E2: The planet Mars, I scarcely need remind the reader, revolves about the sun at a mean distance of 140,000,000 miles, and the light and heat it receives from the sun is barely half of that received by this world. It must be, if the nebular hypothesis has any truth, older than our world; and long before this earth ceased to be molten, life upon its surface must have begun its course. The fact that it is scarcely one seventh of the volume of the earth must have accelerated its cooling to the temperature at which life could begin. It has air and water and all that is necessary for the support of animated existence.

E3: Principia Mathematica, the landmark work in formal logic written by Alfred North Whitehead and Bertrand Russell, was first published in three volumes in 1910, 1912 and 1913. Written as a defense of logicism (the view that mathematics is in some significant sense reducible to logic) the book was instrumental in developing and popularizing modern mathematical logic. It also served as a major impetus for research in the foundations of mathematics throughout the twentieth century. Along with the Organon written by Aristotle and the Grundgesetze der Arithmetik written by Gottlob Frege, it remains one of the most influential books on logic ever written.

Koriste se n-grami dužine 3, tj. koristi se vrednost $n = 3$. Iz tekstova S1 i E1 izdvojeno je po 10 najfrekventnijih n-grama. Ovi n-gramski profili činiće svojstva instanci. Svojstva trening i test skupa dati su u tabeli 11.2.

Prilikom klasifikacije biće korišćeno Euklidsko rastojanje

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Potrebno je ispitati rastojanja od instanci test skupa do instanci trening skupa.

$$d(S2, S1) = 0.0124$$
$$d(S2, E1) = 0.0417$$

$$d(S3, S1) = 0.0133$$
$$d(S3, E1) = 0.0450$$

$$d(E2, S1) = 0.0482$$

n-gram	Trening skup		Test skup			
	S1	E1	S2	S3	E2	E3
JE_	0.0129	0	0.0131	0.0201	0	0
_PR	0.0125	0.0023	0.0098	0.0148	0	0
ANJ	0.0076	0	0.0082	0.0027	0	0
_KO	0.0076	0	0.0016	0.0027	0	0
JA_	0.0076	0	0.0033	0.0040	0	0
_JE	0.0067	0	0.0082	0.0121	0	0
_PO	0.0067	0.0009	0.0147	0.0080	0	0.0016
_SE	0.0062	0.0018	0.0049	0.0027	0.0016	0.0032
NJE	0.0058	0	0.0065	0.0027	0	0
_U_	0.0058	0	0.0033	0.0067	0	0
_TH	0	0.0212	0	0	0.0270	0.0175
THE	0	0.0148	0	0	0.0202	0.0191
HE_	0	0.0120	0	0	0.0185	0.0127
ING	0	0.0088	0	0	0.0017	0.0032
NG_	0	0.0078	0	0	0.0034	0.0048
_CO	0.0004	0.0074	0	0	0.0051	0
ER_	0.0009	0.0069	0	0	0.0051	0.0032
ND_	0	0.0065	0	0	0.0101	0.0079
_TO	0.0018	0.0065	0.0049	0	0.0034	0.0016
TO_	0.0009	0.0065	0.0033	0	0.0034	0.0016

Tabela 11.2: Trening i test skup za klasifikaciju tekstova prema jeziku. Za svaki 3-gram prikazana je njegova frekvencija u tekstu S1 i E1.

$$d(E2, E1) = 0.0149$$

$$d(E3, S1) = 0.0397$$
$$d(E3, E1) = 0.0141$$

Pošto je rastojanje od instance S2 do S1 manje nego od S2 do E1, zaključuje se da je S1 najbliži sused instance S2. Zbog toga se instanca S2 prepoznaje kao tekst na sprskom jeziku. Slično se ispravno zaključuje i da je S3 tekst na srpskom, E2 tekst na engleskom i E3, takođe, tekst na engleskom jeziku. Posebno je zanimljivo da tekstovi S3 i E2 po svom sadržaju nemaju dodira sa instancama za trening S1 i E1, što ne ometa uspešnu klasifikaciju.

11.10.6 Prepoznavanje poze i delova tela iz dubinskih slika stablima odlučivanja

Majkrosoftov sistem Kinekt (eng. *Microsoft Kinect*) sa svojom mogućnošću prepoznavanja poze tela igrača predstavlja jedan od zanimljivijih primera primene mašinskog učenja koja je doprla do velikog broja ljudi. Sistem prepoznaje poze i pokrete igrača i preduzima akcije u igri u skladu sa njima. Kinekt je prvi sistem koji je omogućio prepoznavanje punog spektra pokreta i veliku robusnost u odnosu na oblik tela, odeću i slično, a pri brzini od 200 slika u sekundi. Ovakva brzina posledica je dve odluke u dizajnu sistema. Prvo, svojstva na osnovu kojih se uči definisana su tako da se izrazito brzo izračunavaju. Drugo, koriste se stabla odlučivanja, a kod njih je postupak predviđanja vrlo brz, značajno brži nego kod modernih neuronskih mreža: u slučaju izbalansiranih stabala, vreme je logaritamsko u odnosu na veličinu stabla. Sistem se u potpunosti oslanja na slike dubinske kamere koje kao vrednost svakog piksela daju *dubinu* — rastojanje od kamere do lokacije na objektu na kojoj se taj piksel nalazi.

Sistem je treniran na osnovu više stotina hiljada dubinskih slika (pravih i simuliranih), pri čemu je za svaku takvu sliku dostupna i druga, njoj odgovarajuća slika sa datim informacijama o delovima tela kojima pikseli pripadaju (ilustracija na slici 11.22). Prepoznavanje dela tela vrši se za svaki piksel pomoću skupa stabala odlučivanja, takozvane *slučajne šume*. Kao i svaki model, i ovaj zahteva svojstva na osnovu kojih će se vršiti predviđanja. Svaki piksel opisan je skupom svojstava uniformne strukture. Za dati piksel, svako svojstvo predstavlja razliku između dubina neka dva, pogodno odabrana, piksela bliska polaznom. Neka je $d_{\mathbf{I}}(\mathbf{x})$ dubina piksela na poziciji $\mathbf{x}$ na slici $\mathbf{I}$ (u slučaju pozicija koje su van slike ili na pozadini, za dubinu se uzima velika pozitivna vrednost). Onda, za par vektora $\theta = (\mathbf{u}, \mathbf{v})$ u prostoru koordinata slike, odgovarajuće svojstvo piksela

$\mathbf{x}$ definisano je na sledeći način:

$$f_\theta(\mathbf{I}, \mathbf{x}) = d_\mathbf{I}\left(\mathbf{x} + \frac{\mathbf{u}}{d_\mathbf{I}(\mathbf{x})}\right) - d_\mathbf{I}\left(\mathbf{x} + \frac{\mathbf{v}}{d_\mathbf{I}(\mathbf{x})}\right)$$

Razmotrimo smisao ovog svojstva. Za dati piksel $\mathbf{x}$, $\mathbf{x} + \mathbf{u}/d_\mathbf{I}(\mathbf{x})$ predstavlja piksel u relativnom susedstvu u smeru vektora $\mathbf{u}$. Deljenje dubinom piksela $\mathbf{x}$ služi da umanji pomeraj na slici u slučaju daljih piksela kako bi relativni odnos pomeraja za piksele na različitim dubinama ostao isti. Onda svojstvo definisano za dati par θ predstavlja razliku dubina odgovarajućih suseda datog piksela $\mathbf{x}$. Mnoštvo ovakvih svojstava za različite parove θ opisuje oblik okoline piksela $\mathbf{x}$, što može dati dovoljno informacija za prepoznavanje dela tela. Jedan od ključnih motiva za izbor ovakvih svojstava je njihovo izrazito brzo izračunavanje. Konkretno, za izračunavanje jednog svojstva potrebno je izvršiti samo tri čitanja iz memorije kako bi se odredile dubine i pet aritmetičkih operacija kako bi se izračunala njegova vrednost.

Slika 11.22: Dubinske slike i odgovarajuće slike sa obeleženim delovima tela.

Algoritam učenja je po duhu sličan algoritmu ID3 koji je predstavljen u poglavlju 11.7, ali kako su vrednosti svojstava neprekidne, ID3 se ne može direktno primeniti. Dodatno, jedno stablo je manje moćno od većeg broja relativno nezavisnih stabala, pa se umesto jednog koristi šuma stabala. Najpre ćemo opisati kako se trenira jedno stablo, a potom kako se vrši zajedničko odlučivanje pomoću većeg broja stabala.

Algoritam učenja primenjuje se nad skupom piksela D koji je dobijen nasumičnim izborom po 2000 piksela sa svake slike. Algoritam je određen sledećom rekurzivnom procedurom:

1. Nasumice generisati testove kao parove $\phi = (\theta, \tau)$, gde je τ skalar.
2. Za svaki test ϕ, izračunati dobitak informacije kada se podaci podele na skup $D_l(\phi)$ u kojem su pikseli za koje je vrednost svojstva f_θ manja od praga τ i na skup $D_r(\phi)$ koji sadrži ostale piksele.
3. Izabrati za koren stabla test ϕ^* za koji je dobitak informacije maksimalan.
4. Rekurzivno konstruisati levo podstablo nad skupom piksela $D_l(\phi^*)$ i desno nad skupom $D_r(\phi^*)$.

Svako stablo t u svakom listu definiše raspodelu po klasama $P_t(c|\mathbf{I}, \mathbf{x})$ gde je verovatnoća svake klase c propocionalna broju piksela iz trening skupa koji su pridruženi tom listu, a pripadaju klasi c. Naravno, nije za svaki list potrebno čuvati sve instance koje su mu u treningu pridružene, već samo (diskretnu) raspodelu verovatnoće po klasama. Kada je na ovaj način generisano T stabala, raspodela verovatnoće po klasama se definiše kao

$$P(c|\mathbf{I}, \mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} P_t(c|\mathbf{I}, \mathbf{x}),$$

a za datu sliku I i piksel $\mathbf{x}$, klasa se prirodno bira kao najverovatnija klasa u odnosu na ovu raspodelu.

Primetimo da pomenuta relativna nezavisnost stabala dolazi iz nasumičnog izbora podataka i testova pri njihovoj izgradnji. Osnovna ideja algoritma je da će ovako generisana stabla grešiti nezavisno jedna od drugih, pa ako je svako od njih bolje od slučajnog pogađanja, većina stabala će biti u pravu i moći će da nadglasa manjinu koja greši na datoj instanci.

Na ovaj način, moguće je označiti sve piksele najverovatnijim klasama koje se odnose na delove tela. Ipak, za potrebe praćenja pokreta, to je previše sirova informacija. Umesto da se koriste pozicije svih piksela koji

čine neki deo tela, poželjnije je koristiti svega nekoliko reprezentativnih tačaka. Ovo se postiže klasterovanjem odnosno grupisanjem svih tačaka koje čine neki deo tela i uzimanjem centralnih tačaka tih klastera koje nadalje predstavljaju taj deo tela. Na ovaj način je moguće oceniti pozicije različitih delova tela, a u procesu klasterovanja se eliminiše i šum koji nastaje usled pogrešne klasifikacije malog broja piksela.

Glava 12

Nenadgledano mašinsko učenje

Nenadgledano učenje je vid mašinskog učenja kod kojeg nisu date vrednosti ciljne promenljive. Naravno, mašinsko učenje ne može dati korisnu informaciju ni iz čega. Algoritmi nadgledanog učenja su takvi da često mogu učiti bilo kakve zakonitosti u datim podacima, a da se to što se uči definiše vrednostima ciljne promenljive. Kako u slučaju nenadgledanog učenja tih ciljnih vrednosti nema, ono što se uči mora biti definisano samim algoritmom. Dakle, algoritmi nenadgledanog učenja su algoritmi specifične namene.

Većina problema koji odgovaraju ovoj grupi potpada pod probleme klasterovanja, učenja reprezentacije i detekcije anomalija:

Klasterovanje predstavlja uočavanje grupa u podacima, što govori nešto o strukturi podataka i može biti korisno u različite svrhe, o kojima će biti reči u nastavku.

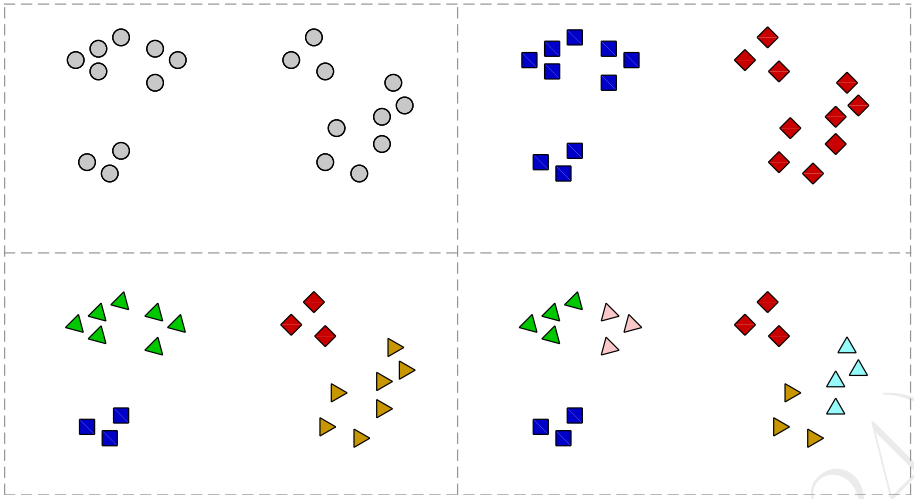
Učenje reprezentacije predstavlja sve značajniju vrstu zadataka u mašinskom učenju. Neretko podaci nisu u obliku u kojem ih algoritam učenja ili čovek mogu lako iskoristiti. Na primer, algoritmi mašinskog učenja obično zahtevaju više parametara ukoliko podaci imaju više dimenzija i zahtevaju više računskih operacija, a podaci su često visokodimenzionalni – recimo, 1000000 dimenzija u slučaju slika kod kojih svaki piksel predstavlja jednu dimenziju ili 100000 dimenzija u slučaju obrade teksta gde se za svojstva koriste frekvencije pojedinih reči. Međutim, ovako visoka dimenzionalnost najčešće je posledica izbora reprezentacije podataka. Na primer, slike lica, čak i u rezoluciji 1000×1000, ne popunjavaju ravnomerno 1000000-dimenzionalni prostor, već samo njegov delić. Ostatak odgovara drugim slikama, od kojih mnoge ne predstavljaju ništa prepoznatljivo čoveku. To sugeriše da postoji reprezentacija manje dimenzionalnosti koja takođe opisuje sva lica. Nalaženje takvih reprezentacija i učenje nad njima značajno povećava uspešnost algoritama učenja. Nekada ovakvi algoritmi služe i za smanjenje dimenzionalnosti podataka na dve ili tri dimenzije koje najbolje oslikavaju njihovu varijabilnost, što omogućava čoveku da u nekim slučajevima posmatranjem uoči neke važne aspekte podataka.

Detekcija anomalija se tiče uočavanja podataka koji odudaraju od ostalih, bilo kako bi se izbacili iz skupa podataka, čime se često olakšava njihovo modelovanje i analiza, bilo kako bi se dalje analizirali. Primera radi, transakcije kreditnim karticama koje predstavljaju prevaru, lako mogu odudarati od uobičajenog načina na koji korisnik koristi karticu, bilo po vremenu upotrebe, bilo po vrsti usluge ili proizvoda koji se kupuje, bilo po iznosu transakcije. Slično važi i za upade u računarske sisteme – ta vrsta ponašanja često odudara od uobičajenog ponašanja korisnika.

U nastavku ćemo se fokusirati na problem klasterovanja.

12.1 Klasterovanje

Klasterovanje predstavlja identifikaciju grupa u datim podacima. Potreba za rešavanjem ovakvog zadatka može se javiti u različitim praktičnim problemima, poput identifikacije zajednica u društvenim mrežama (na primer, za potrebe oglašavanja), detekcije raznorodnih tkiva na medicinskim snimcima, ustanovljavanja zajedničkog porekla jezika, identifikacije grupa živih bića i, specifično, ljudskih zajednica i slično. Pored primena koje u sebi direktno kriju problem klasterovanja, ova tehnike korisne su i u pretprocesiranju podataka na koje kasnije treba da budu primenjene metode nadgledanog učenja. Na primer, u slučaju obrade ogromnog broja podataka, cele grupe podataka mogu biti zamenjene svojim reprezentativnim predstavnicima. Ovo nije uvek idealno sa tačke gledišta kvaliteta dobijenog prediktivnog modela, ali sa tačke gledišta vremenske i memorijske efikasnosti može biti isplativo.



Slika 12.1: Različita klasterovanja nad istim podacima: jedan klaster (gore levo), dva klastera (gore desno), četiri klastera (dole levo), šest klastera (dole desno).

Pojam klasterovanja nije jednoznačno definisan. Kao što primer prikazan na slici 12.1 ukazuje, u jednom skupu može se identifikovati više različitih grupisanja, često različite granularnosti. Pritom, takvi slučajevi nisu posledica nedovoljnog promišljanja definicije klasterovanja, već raznovrsnosti konteksta u kojima se grupisanje može vršiti i ciljeva koji se pomoću klasterovanja žele postići. Za očekivati je da je nekada potrebno izvršiti grublje klasterovanje – u manji broj klastera, a nekada finije – u veći broj klastera. Algoritmi klasterovanja obično omogućavaju podešavanje nivoa granularnosti tj. broja klastera koji se u podacima pronalazi.

Pojam klasterovanja nije jednoznačno definisan ne samo u odnosu na broj klastera koji se u podacima mogu naći, već i u odnosu na ideju šta jednu grupu instanci čini klasterom. U odnosu na to, postoji više neformalnih definicija klasterovanja. *Globularni* ili *centrični* klasteri su grupe tačaka koje popunjavaju unutrašnjost lopte ili, opštije, elipsoida. *Dobro razdvojeni* klasteri su grupe tačaka koje su bliže drugim tačkama svoje grupe nego bilo kojoj tački iz neke druge grupe. *Gustinski* klasteri su klasteri čije su tačke razdvojene od tačaka drugih klastera regionima manje gustine. *Hijerarhijski* klasteri su ili pojedinačne tačke ili klasteri čije su tačke takođe organizovane u strukturu hijerarhijskih klastera.

U nastavku je prikazano nekoliko jednostavnih, a često korišćenih algoritama klasterovanja.

12.1.1 Algoritam k sredina

Algoritam k sredina pronalazi k klastera koje predstavlja pomoću k *težišta* tih klastera, od kojih se svako dobija uprosečavanjem elemenata datog klastera. Taj korak čini algoritam primenljivim samo na podatke koji se mogu uprosečavati, poput vektora. Postoje uopštenja algoritma i na drugačije vrste podataka, ali o njima ovde neće biti reči. Polaznih k težišta bira se nasumično (mada, ako korisnik zna nešto o strukturi svojih podataka, mogu biti i unapred data), a potom se ponavljaju koraci pregrupisavanja tačaka u nove klastere prema bliskosti sa postojećim težištima i preračunavanja novih težišta sve dok se ona menjaju. Algoritam je preciznije formulisan na slici 12.2.

Primer klastera koje pronalazi ovaj algoritam, dat je na slici 12.3. Može se pokazati da ovaj algoritam minimizuje veličinu

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{c}_i)^2$$

gde je d euklidsko rastojanje, C_i i-ti klaster, a $\mathbf{c}_i$ njegovo težište. Na osnovu toga može se nešto zaključiti i o njegovom ponašanju. Zahvaljujući tome što je zasnovan na minimizaciji euklidskog rastojanja, algoritam teži pronalaženju klastera u obliku lopte. Kako je rastojanje kvadrirano, algoritam je osetljiv na podatke koji značajno odudaraju od ostalih. Naime, u slučaju takvog podatka, veće rastojanje će uticati na ukupnu grešku neproporcionano u odnosu na ostala rastojanja i takav podatak će neproporcionalno uticati na lokaciju težišta. Takođe, ako gustina podataka, tj. tačaka prostora ne varira drastično i rastojanja među klasterima nisu velika, algoritam preferira klastere sa sličnim brojem tačaka u njima, pošto bi u suprotnom brojan klaster sadržao i tačke dalje od težišta koje bi značajno povećavale sumu kvadrata rastojanja.

Činjenica da algoritam k sredina minimizuje navedenu sumu navodi na njenu dalju analizu. Bitno je pitanje da li ona ima jedan globalni minimum, odnosno da li je najbolje klasterovanje u odnosu na datu sumu kvadrata

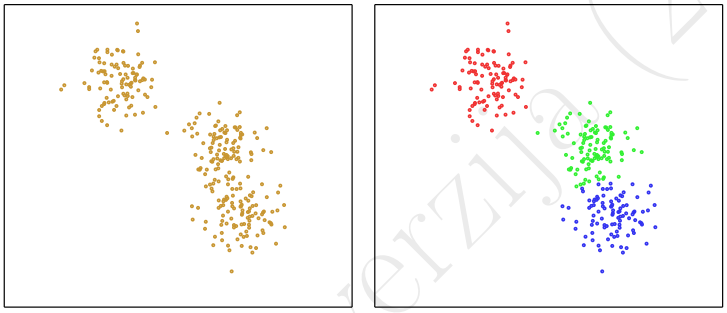
Algoritam: Algoritam k sredina

Ulaz: Trening skup $T \subset \mathbb{R}^n$, broj klastera k

Izlaz: Particionisanje skupa T na disjunktne neprazne podskupove $C_1, \ldots, C_k$

1: Nasumice izaberi težišta $\mathbf{c}_1, \ldots, \mathbf{c}_k$ iz skupa T;
2: **ponavljaj**
3: Postavi skupove $C_1, \ldots, C_k$ na prazne skupove;
4: **za** svaku instancu $\mathbf{x} \in T$ **radi**
5: pronađi težište $\mathbf{c}_i$ koje je najbliže instanci $\mathbf{x}$;
6: dodaj $\mathbf{x}$ u skup C_i;
7: Izračunaj proseke $\mathbf{c}_1, \ldots, \mathbf{c}_k$ instanci iz skupova $C_1, \ldots, C_k$;
8: **dok nije ispunjen** uslov da su težišta ista kao u prethodnoj iteraciji;
9: vrati $C_1, \ldots, C_k$ kao rešenje.

Slika 12.2: Algoritam klasterovanja k sredina.



Slika 12.3: Klasteri pronađeni algoritmom k sredina.

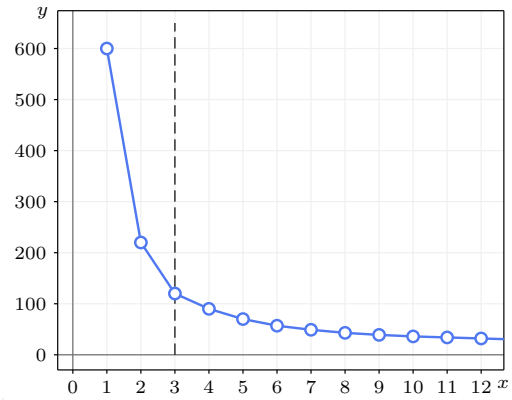
rastojanja jedinstveno. Odgovor na ovo pitanje je negativan. Moguće je da postoji veći broj klasterovanja jednakog kvaliteta. Jedan primer u kojem bi to bilo i intuitivno je kada su tačke uniformno raspoređene unutar kruga i potrebno ih je podeliti na dva klastera. Rotiranje dobijenih težišta u odnosu na centar kruga daje podjednako dobro klasterovanje. Drugim rečima, u slučaju takvog skupa podataka, postoji puno globalnih, i samim tim podjednako dobrih, minimuma. Takva situacija nije zabrinjavajuća. Ipak, ispostavlja se da mogu postojati i lokalni minimumi slabijeg kvaliteta od globalnog i da algoritam može naći takav minimum, što nije dobro. Ovaj problem ublažava se tako što se klasterovanje pokreće veći broj puta sa različitim inicijalnim težištima, a za rezultat se uzima klasterovanje sa najmanjom vrednošći sume kvadrata rastojanja.

Algoritam k sredina omogućava fleksibilnost pri pronalaženju klastera kroz mogućnost podešavanja broja k. Ipak, ta fleksibilnost često vodi do nedoumica jer u praksi često nije jasno kako izabrati broj k. Jedno heurističko pravilo je „pravilo lakta“ koje sugeriše da se za različite vrednosti broja k izvrši klasterovanje, da se nacrta grafik zavisnosti sume kvadrata rastojanja u zavisnosti od k i da se izabere klasterovanje koje odgovara broju k koji odgovara tački nagle promene brzine opadanja grafika, tj. na njegovom „laktu“. Ovakva situacija prikazana je na grafiku 12.4. Intuitivno obrazloženje je da su nakon „lakta“ klasteri već homogeni i dodavanje novih težišta ne doprinosi značajno smanjenju sume kvadrata rastojanja.

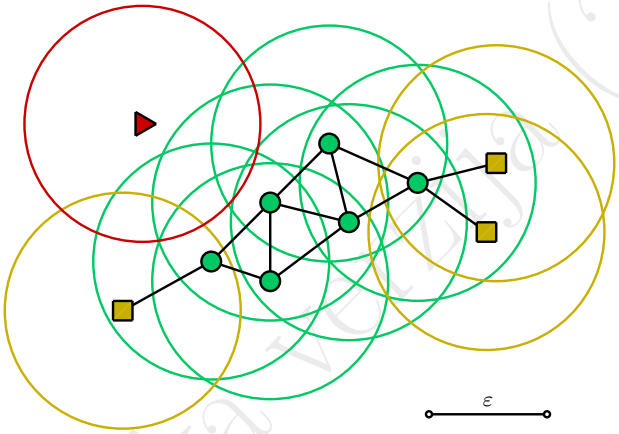
12.1.2 DBSCAN

Algoritam DBSCAN (eng. *density-based spatial clustering of applications with noise*) služi za detekciju gustinskih klastera i nije ograničen njihovim oblikom. Algoritam ima dva hiperparametra – rastojanje ε i minimalan broj tačaka μ. Algoritam razvrstava tačke na *tačke koje čine jezgro* — čija ε okolina sadrži bar μ tačaka, *granične tačke* — koje u svojoj okolini imaju neku tačku koja čini jezgro i *tačke koje čine šum* — koje nisu ni granične niti čine jezgro. Tačke koje čine šum se zanemaruju, dok se ostale grupišu u različite klastere na osnovu bliskosti. Ilustracija vrsta tačaka data je na slici 12.5. Algoritam je preciznije opisan na slici 12.6.

DBSCAN očigledno ne pretpostavlja oblik klastera, tako da se može koristiti za detekciju klastera najrazličitijih oblika ako ih je moguće razdvojiti regionima niže gustine. Još jedna dobra strana ovog algoritma je to



Slika 12.4: Pravilo lakta sugeriše da za broj k treba uzeti vrednost koja odgovara uspravnoj isprekidanoj liniji. Na x osi je broj klastera, a na y osi suma kvadrata rastojanja.



Slika 12.5: Tri vrste tačaka kojima operiše algoritam DBSCAN (za vrednost $\mu = 4$): (zelenim) kružićima označene su tačke jezgra, (žutim) kvadratićima granične tačke, a (crvenim) trouglićima tačke koje čine šum.

Algoritam: Algoritam DBSCAN

Ulaz: Trening skup T, rastojanje ε i broj μ

Izlaz: Particionisanje skupa T na disjunktne neprazne podskupove $C_1, \ldots, C_k$

1: Formiraj skup $\mathcal{C}$ svih tačaka čija ε okolina sadrži bar μ tačaka;

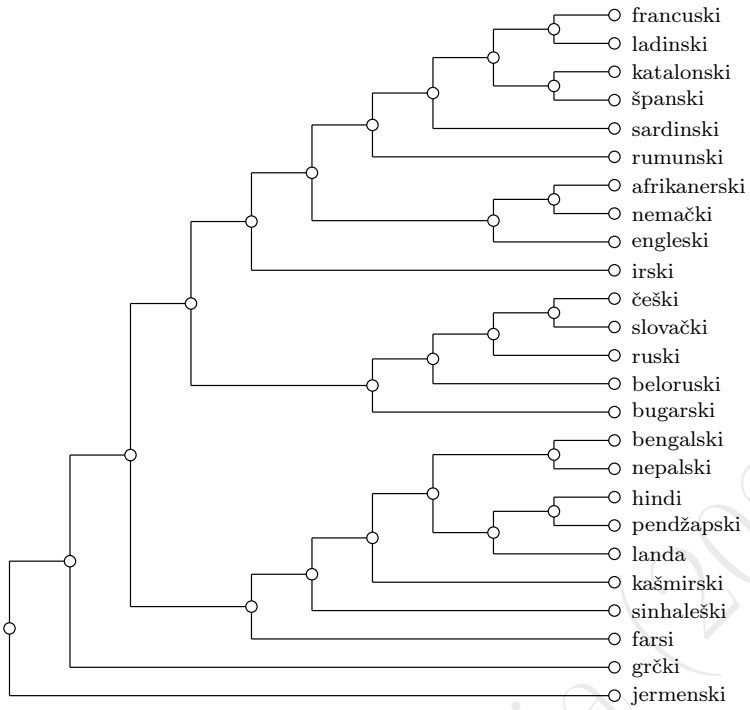
2: Formiraj skup $\mathcal{B}$ svih tačaka iz $T \setminus \mathcal{C}$ koje u svojoj ε okolini imaju bar jednu tačku iz $\mathcal{C}$;

3: Formiraj graf G čiji su čvorovi tačke iz $\mathcal{C} \cup \mathcal{B}$, a grana postoji između svake dve tačke koje su na rastojanju najviše ε;

4: vrati komponente povezanosti $C_1, \ldots, C_k$ grafa G kao rešenje.

Slika 12.6: Algoritam klasterovanja DBSCAN.

što može odstraniti odudarajuće podatke kao što je šum (dok je algoritam k najbližih suseda vrlo osetljiv na njihovo prisustvo). Potencijalni problem nastaje kada su klasteri, iako razdvojeni, sami vrlo različite gustine. Naime, u okolini vrlo gustog klastera, šum može imati veću gustinu nego ceo drugi klaster (koji ipak može biti prepoznatljiv po tome što će značajno odudarati po gustini od svoje okoline). Kao i u slučaju algoritma k sredina, nema jasnih opštih pravila za izbor vrednosti ε i μ.



Slika 12.7: Primer hijerarhijskog klasterovanja nekih indoevropskih jezika.

12.1.3 Hijerahijsko klasterovanje

Hijerarhijsko klasterovanje konstruiše stablo u čijim se listovima nalaze instance trening skupa, a unutrašnji čvorovi definišu strukturu klastera. Klaster koji odgovara nekom unutrašnjem čvoru sastoji se iz klastera koji odgovaraju njegovim direktnim potomcima. Problem klasterovanja svodi se na problem konstrukcije ovakvog stabla. Postoji više pristupa rešavanju ovog problema. Jedan koji se često koristi je *hijerarhijsko aglomerativno klasterovanje* pri kojem se skup klastera inicijalizuje pojedinačnim instancama, a potom se u svakom koraku, spajaju dva najsličnija klastera u jedan, čime se konstruiše binarno stablo. Takvo stablo naziva se *dendogram* i ilustrovano je na slici 12.7. Sličnost nad klasterima nije trivijalno definisati i ne postoji jedan izbor. Najčešće se definiše neka mera sličnosti ili rastojanja nad pojedinačnim instancama (poput euklidskog rastojanja), pa se mera sličnosti ili rastojanja klastera definiše na osnovu nje. Na primer, rastojanje između dva klastera može se definisati kao minimum, prosek ili maksimum rastojanja njihovih elemenata. Precizniji opis hijerarhijskog aglomerativnog klasterovanja dat je na slici 12.8. Ovaj opis koristi meru rastojanja, ali se algoritam lako može izraziti i u terminima sličnosti.

Algoritam: Algoritam aglomerativnog hijerarhijskog klasterovanja

Ulaz: Trening skup $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, mera rastojanja između klastera d

Izlaz: Stablo klasterovanja $\mathcal{T}$

1: Inicijalizuj skup $\mathcal{C}$ na skup listova $\{\{(\mathbf{x}_1, 0)\}, \ldots, \{(\mathbf{x}_N, 0)\}\}$;
2: **ponavljaj**
3: Izračunaj rastojanja $d_{ij} = d(\mathbf{c}_i, \mathbf{c}_j)$ među svim elementima $(\mathbf{c}_i, d_i), (\mathbf{c}_j, d_j) \in \mathcal{C}$;
4: Pronađi najbliži par elemenata $(\mathbf{c}_i, d_i), (\mathbf{c}_j, d_j)$ iz $\mathcal{C}$;
5: Konstruiši unutrašnji čvor $(\mathbf{c}_i \cup \mathbf{c}_j, d_{ij})$ i dodaj mu čvorove $(\mathbf{c}_i, d_i)$ i $(\mathbf{c}_j, d_j)$ kao direktne potomke;
6: $\mathcal{C} \leftarrow (\mathcal{C} \setminus \{(\mathbf{c}_i, d_i), (\mathbf{c}_j, d_j)\}) \cup \{(\mathbf{c}_i \cup \mathbf{c}_j, d_{ij})\}$;
7: **dok nije ispunjen** uslov da je skup $\mathcal{C}$ jednočlan;
8: Vrati stablo čiji je koren jedini čvor u skupu $\mathcal{C}$.

Slika 12.8: Algoritam aglomerativnog hijerarhijskog klasterovanja.

Sečenjem stabla na različitim nivoima mogu se dobiti klasterovanja različite granularnosti. Konkretno, za neku vrednost rastojanja d, na osnovu stabla moguće je identifikovati poslednji skup $\mathcal{C}$ koji je u toku kreiranja stabla nastao spajanjem čvorova čije rastojanje ne prelazi datu vrednost d. Dobra strana ovakvog klasterovanja je to što pruža više informacija nego algoritmi koji pronalaze samo jednu podelu podataka na klastere. Takođe, različitim izborima mere sličnosti nad klasterima mogu se dobiti različiti klasteri. Loša strana algoritma je to što u toku rada izračunavanje sličnosti svih klastera sa svim klasterima vodi visokoj vremenskoj, a najčešće i prostornoj složenosti zbog čuvanja matrice rastojanja.

12.2 Primeri primena klasterovanja

Kao što je već navedeno, klasterovanje se često koristi kao korak pretprocesiranja podataka za potrebe nadgledanog učenja. Ipak, zanimljivije je razmotriti primere primena u kojima je rezultat klasterovanja relevantan sam za sebe. U nastavku diskutujemo dve takve primene.

12.2.1 Utvrđivanje srodnosti prirodnih jezika

Jedno od ključnih pitanja uporedne lingvistike je ustanovljavanje srodnosti između postojećih prirodnih jezika. Smatra se da se svi jezici mogu hijerarhijski grupisati u skladu sa njihovom srodnošću koja je posledica nastanka više jezika od zajedničkog pretka, takozvanog proto jezika. Ovaj postupak grupisanja može se hijerarhijski ponavljati. Iako ne postoji saglasnost u vezi sa tačnim brojem grupa i njima odgovarajućih proto jezika, obično postoji saglasnost do nekog nivoa grupisanja. Na primer, neke jezičke grupe su grupe indo-evropskih, sino-tibetanskih, turkijskih i uralskih jezika. Neke od podgrupa indo-evropskih jezika su germanska, balto-slovenska, indo-iranska i keltska. Ove grupe se mogu dalje deliti. Ovakvo grupisanje obično se ustanovljava temeljnom višedecenijskom analizom koju sprovode zajednice lingvista koje se bave ovim problemima. Nekada postoje neslaganja u vezi sa pripadnostima nekih jezika određenim podgrupama, posebno u vezi sa starim jezicima na kojima nema mnogo pisanih dokumenata (često su to samo natpisi u kamenu).

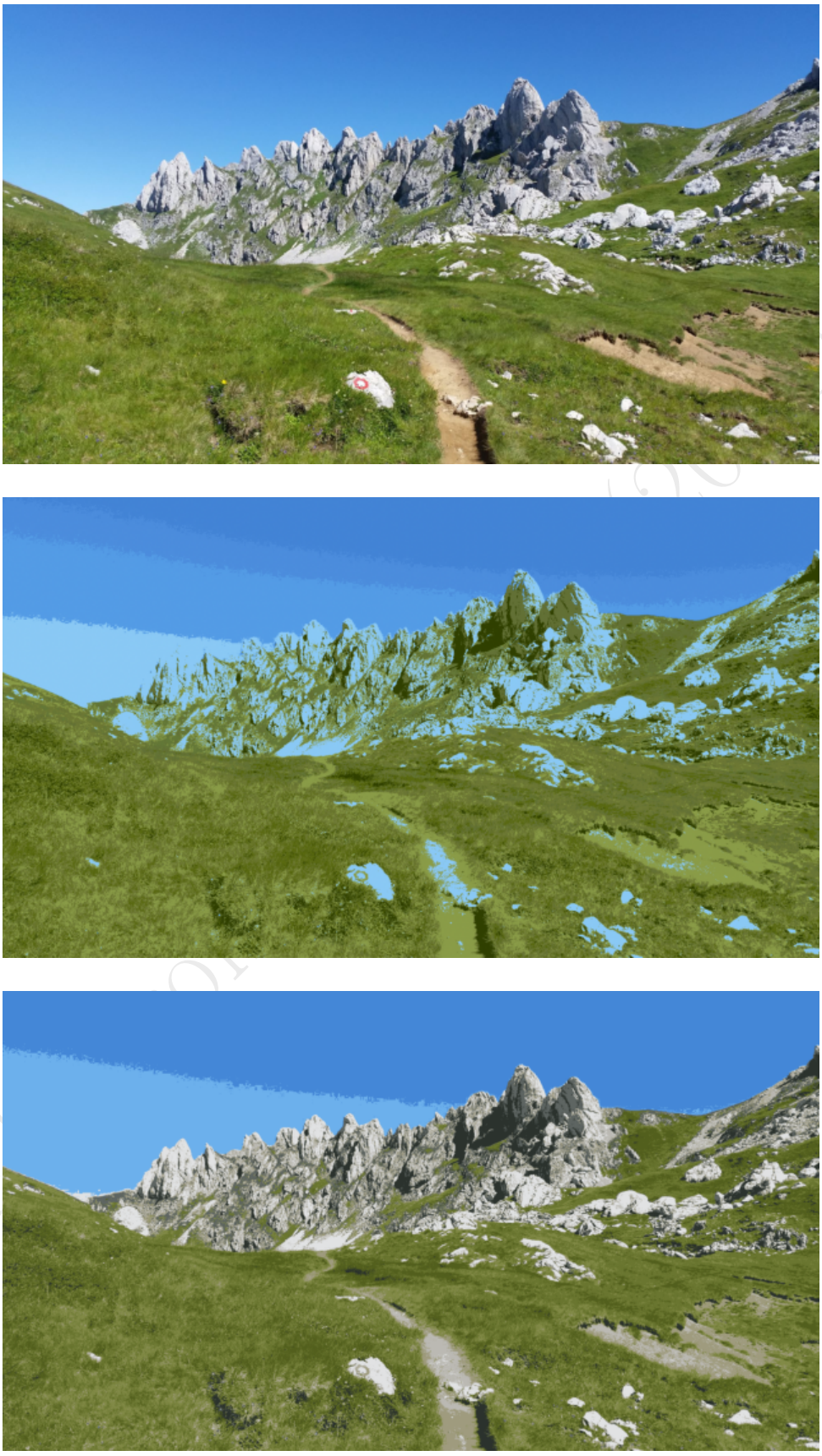
Zanimljivo je da se ovakva grupisanja vrše i automatskim metodama – metodama hijerarhijskog klasterovanja koje prirodno formiraju ugnežđene klastere. Naravno, pitanje je kako predstaviti jezike tako da se nad njima može izvršiti klasterovanje i kako meriti sličnost između njih. Jedan način da se ovo uradi sastoji se u uparivanju odgovarajućih reči različitih jezika. Sličnost dva jezika se onda može kvantifikovati zbirom edit rastojanja po svim parovima odgovarajućih reči za ta dva jezika. Koristeći te informacije, moguće je primeniti neku od diskutovanih tehnika hijerarhijskog klasterovanja. Izbor skupa reči kojima će biti predstavljeni jezici može uticati na ishode klasterovanja. Na primer, skorašnja preuzimanja iz stranih jezika verovatno će mnoge jezike učiniti naizgled srodnim sa engleskim jezikom. Otud je u tom izboru potrebno učešće stručnjaka koji bi sugerisali koje grupe reči su bolje za ovakva istraživanja (na primer, reči za male brojeve, floru, faunu, zemljoradnju i slično). Naravno, postupak može biti i složeniji od opisanog ili zasnovan na drugačijoj reprezentaciji. Recimo, jezik bi se mogao predstaviti frekvencijama bigrama izračunatih iz velikog neobeleženog korpusa. Takvi vektori mogu se porediti euklidskim rastojanjem. Ishod hijerarhijskog klasterovanja nekih predstavnika indoevropskih jezika zasnovanog na bigramskoj reprezentaciji i euklidskom rastojanju dat je na slici 12.7. Jasno se mogu prepoznati karakteristične podgrupe ove familije jezika.

Primetimo da je ovakav postupak mogao biti primenjen (i primenjuje se) i za utvrđivanje srodnosti živih bića, na osnovu njihovog genetskog materijala koji se takođe može predstaviti tekstom u vidu nizova aminokiselina od kojih svakoj odgovara po jedno slovo.

12.2.2 Smanjenje palete boja slike

Nekada je potrebno smanjiti broj različitih nijansi boja koje su prisutne na slici. Ovo može biti rađeno u svrhe smanjenja zapisa slike ili zbog potreba štampe. Jedan od mogućih pristupa bi bio uočavanje sličnih nijansi i njihova zamena jednom nijansom koja je u nekom smislu najsličnija svima njima. Pretpostavimo da se slika predstavlja pomoću 24-bitnog zapisa u kojem su boje predstavljene pomoću intenziteta crvene, zelene i plave boje (tzv. RGB sistem boja) tako da se intenzitet svake od njih predstavlja pomoću jednog bajta brojevima od 0 do 255. Ovakav zapis dozvoljava opisivanje preko 16 miliona boja. Za zapis slike rezolucije 1000×1000, potrebno je 3 miliona bajtova. Ukoliko bi se broj različitih boja sveo na 256, bilo bi potrebno, bez ikakve kompresije, milion bajtova za zapis slike u kojem bi svakom pikselu bio pridružen jednobajtni kôd boje i bilo bi potrebno 256×3 bajtova za predstavljanje palete – niza od 256 24-bitnih brojeva koji predstavljaju izbor 256 korišćenih nijansi iz polaznog 24-bitnog sistema. Problem se sastoji u identifikaciji izbora podskupa, odnosno palete boja koje će poslužiti za aproksimaciju svih ostalih boja na slici. Pritom, ta paleta može da varira od slike do slike i bira se tako da bude pogodna za konkretnu sliku.

Jedan jednostavan pristup smanjenju palete sastoji se u klasterovanju piksela pomoću algoritma k sredina u onoliko klastera kolika je željena veličina palete. Kao što je rečeno, ovaj algoritam minimizuje sume kvadrata rastojanja podataka od odgovarajućih težišta. Neka su podaci svi pikseli na slici, predstavljeni RGB vrednostima svojih boja. Nakon izvršavanja algoritma, za aproksimaciju svake boje može se koristiti odgovarajuće težište koje se interpretira kao boja u RGB sistemu. Ona su izabrana tako da budu u proseku najsličnija svim bojama iz klastera. Na slici 12.9 dat je primer smanjenja palete polazne slike na ukupno 10 boja, na dva načina – nasumičnim izborom od 166075 boja sa polazne slike i pomoću klasterovanja algoritmom k sredina. U oba slučaja, rezultujuće slike se dobijaju tako što se boja svakog piksela zameni najbližom bojom iz palete. Očigledno, pristup zasnovan na klasterovanju daje bolji rezultat, iako se može primetiti jasna granica između dve nijanse neba, nedostatak crvene boje u markaciji na kamenu i sivkasta boja zemljišta.



Slika 12.9: Slika durmitorskog vrha Zupci, slika koja je od nje dobijena smanjenjem palete na 10 nasumično izabranih od 166075 boja sa polazne slike i slika koja je od polazne dobijena smanjenjem palete pomoću algoritma k sredina.

Glava 13

Učenje potkrepljivanjem

Mnogi problemi prirodno se rešavaju nizom akcija koje svojim zajedničkim efektom dovode do cilja. Očigledan primer je igranje strateških igara poput šaha, ali i mnogi problemi robotike, planiranja, upravljanja i slično. U takvim problemima, često za neku akciju u nizu nije lako reći da li je bila ispravna ili nije. U šahu, na primer, gubitak figure često je loš, ali se u nekim situacijama figure namerno žrtvuju kako bi na duže staze bio postignut neki cilj. U takvim problemima obično je lako reći samo da li je problem uspešno rešen čitavim nizom akcija, a netrivijalna je preraspodela zasluga za uspeh ili neuspeh na pojedinačne akcije. Dodatno, ishod neke konkretne akcije zavisi od konteksta u kojem je preduzeta. U šahu, uzimanje protivnikove figure može imati različite posledice zavisno od toga da li je ta figura bila čuvana ili nije. Ovo je tipičan scenario u kojem se koristi takozvano *učenje potkrepljivanjem* (eng. *reinforcement learning*), koje predstavlja pristup mašinskom učenju inspirisan učenjem pomoću pokušaja i grešaka, često prisutnim kod životinja. Cilj ove vrste učenja je da se, na osnovu iskustava iz mnoštva pokušaja rešavanja problema, nauči kako da se biraju adekvatne akcije u različitim situacijama u toku rešavanja problema.

Učenje potkrepljivanjem pretpostavlja postojanje *agenta* koji svojim *akcijama* može da interaguje sa *okruženjem*. Pri svakoj akciji menja se *stanje* okruženja, a agent dobija određenu *nagradu* predstavljenu realnim brojem. *Politika* je preslikavanje iz skupa stanja u skup akcija i nju agent koristi za odlučivanje, tj. za izbor akcije. U cilju rešavanja zadatog problema, agent treba da nauči politiku koja maksimizuje sumu svih nagrada koje dobija u procesu interagovanja. Kako bi se to ostvarilo, algoritmi učenja potkrepljivanjem tragajući za optimalnom politikom preraspodeljuju ukupnu dobijenu nagradu na akcije koje su bile zaslužne za njeno dobijanje, iako ta informacija nije dostupna eksplicitno. Pomenuti pojmovi biće definisani precizno u nastavku.

U nekim problemima akcije se preduzimaju u diskretnim vremenskim koracima, dok je u nekim problemima vreme neprekidno i postoji mogućnost preduzimanja akcija u bilo kom trenutku. U skladu sa tim, razlikujemo diskretne i neprekidne probleme učenja potkrepljivanjem. Ova glava odnosi se na diskretne probleme.

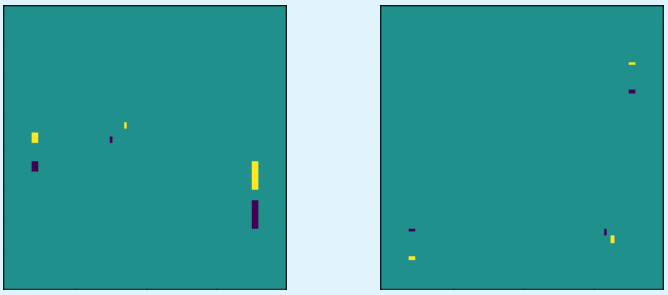
Rešavanje problema pomoću učenja potkrepljivanjem zahteva najpre uočavanje pomenutih elemenata u datom problemu. Skupovi akcija i stanja nekad su određeni sâmom definicijom problema, ali ih je nekada moguće i pogodno definisati ili aproksimirati u fazi modelovanja a u cilju efikasnijeg rešavanja problema. U rešavanju problema ovim pristupom, jedan od najvažnijih izbora je način na koji se definiše nagrada. Potrebno je definisati i formu politike, ali to ne ilustrujemo u početnim primerima.

Veoma su važni koncept stanja i odnos agenta i okruženja. Razmotrimo primer robota čiji je cilj da podigne neki predmet sa poda ispred sebe. Verovatno je prva intuitivna pretpostavka da je pogodno da se robot modeluje kao agent, a da prostor u kojem se kreće i objekti u njemu, uključujući predmet od interesa, predstavljaju okruženje. Međutim, pažljivijim razmatranjem problema zaključuje se da je ključni problem naučiti kako pomoću električnih impulsa koji se šalju motorima robotskih ruku i nogu postići njihovo adekvatno pokretanje. Zato je pogodnije da se i ti motori, odnosno ruke i noge robota, smatraju delom okruženja, a ne agenta. Obično je pogodno da granica između agenta i okruženja ne koincidira sa nama intuitivnom fizičkom granicom, već da bude postavljena bliže agentu.

Primer 13.1. *U slučaju šaha, agent je igrač. Okruženje čine igrač-protivnik i tabla sa figurama (pretpostavljamo da se ne koristi sat). Stanja su uređene 64-orke čiji je svaki element jedna šahovska figura ili prazno polje. Akcije koje agent može da preduzme su legalni šahovski potezi u datoj poziciji. Akcije se preduzimaju u diskretnim koracima, pa je problem očito diskretan. Nagrada može biti* 1 *za potez kojim agent neposredno dobija partiju,* -1 *za potez neposredno nakon kojeg ga drugi igrač matira, a* 0 *u svim ostalim potezima. Tipična greška u modelovanju ovog problema bila bi da se nagrade definišu kao pozitivne vrednosti pri svakom potezu kojim agent uzima protivničku figuru. Na ovaj način agent bi mogao da nauči da uzme veliki*

broj protivničkih figura, čak i po cenu da ubrzo izgubi partiju. Zbog toga je prilikom definisanja nagrada vrlo važno voditi računa o tome da politika koja vodi maksimalnoj nagradi zaista vodi rešenju problema. U slučaju šaha, kako je cilj igre matirati protivnika, a ne imati što više figura, najbezbednije je nagradu definisati strogo u skladu sa tim ciljem, kao što je predloženo u ovom primeru.

Primer 13.2. *U računarskoj igri* Pong *dva igrača se reketima dobacuju lopticom sa ciljem da suprotni igrač ne uspe da vrati lopticu. Igra se završava kada se loptica nađe iza reketa jednog od igrača. Taj igrač je izgubio igru, a drugi je pobedio. Oba igrača imaju po jedan reket i oni se kreću duž vertikalnih ivica ekrana. Agent je ponovo igrač. Moguće akcije su pomeranje reketa ili nagore ili nadole i preduzimaju se više puta u sekundi, ali u diskretnim koracima. Dakle, problem je diskretan. Okruženje čine loptica i drugi igrač. Stanja su u idealnom slučaju određena lokacijom loptice, njenim vektorom brzine, lokacijama reketa oba igrača i njihovim vektorima brzine. Ovde se javlja jedan od ključnih problema vezanih za predstavljanje stanja. Naime, igra* Pong *ne daje neposredno informacije o vektorima kretanja i brzini objekata, već samo generiše slike jednu za drugom u skladu sa kretanjem objekata. Stoga ove veličine nisu neposredno dostupne i ne mogu se direktno koristiti prilikom učenja. Ove informacije mogle bi se aproksimirati tako da se stanje definiše kao razlika dve slike dobijene u susednim trenucima. Sve pomenute veličine vidljive su u takvoj reprezentaciji, kao što je ilustrovano narednim slikama koja prikazuje dva takva stanja. Pretpostavlja se da su boje na slikama u igri kodirane brojevima* 0 *i* 1*. Prilikom oduzimanja tako kodiranih slika, dobijaju se nove slike sa pikselima čije su vrednosti* -1, 0 *ili* 1*. Na sledećoj slici koja to ilustruje, tamno-ljubičasta boja označava vrednost* -1*, zelena vrednost* 0*, a žuta vrednost* 1*. U prvom stanju se vidi da se loptica kreće gore desno, a oba reketa nagore, pri čemu desni značajno brže nego levi. Na drugoj se loptica kreće dole desno, levi reket sporo nadole, a desni sporo nagore.*



Problem je pogodno modelovati tako da se daje nagrada 1 *u koraku u kom je agent neposredno dobio partiju (što je korak u kojem se loptica nađe iza protivnikovog reketa),* -1 *kada ju je izgubio (odnosno kad se loptica nađe iza njegovog reketa) i* 0 *u svakom drugom koraku. Na prvi pogled, možda bi imalo smisla dati agentu nagradu* 1 *kad god uspešno vrati lopticu. Međutim, u tom slučaju bi tražena politika (ona koja maksimizuje ukupnu ostvarenu nagradu) bila takva da agent svaki put vrati lopticu, ali tako da i protivnik takođe može lako da je vrati. Naime, što se igra duže igra, ovakva politika vodi većoj ukupnoj nagradi.*

U prethodnim primerima, nagrade različite od nule dobijaju se vrlo retko – na kraju partije. Takve nagrade nazivaju se *proređenim*. Opravdana je bojazan da na osnovu tako definisanih nagrada agent ne može lako naučiti da rešava složene probleme, što može biti motivacija da se nagrade definišu drugačije i da se agent nagrađuje češće (recimo prilikom ostvarivanja nekih potciljeva). Ipak, kao što je ilustrovano prethodnim primerima, ljudska procena može biti pogrešna te agent može da nauči *neželjeno* ponašanje koje zaista maksimizuje loše definisanu nagradu. Zato je korišćenje proređenih nagrada ipak standardna praksa — obično se agentu daje pozitivna nagrada samo kada ostvari konačni cilj, odnosno da se nagradom agentu ukazuje na to koji cilj treba da ostvari, a ne kako treba da ga ostvari.

Primer 13.3. *Neka je potrebno upravljati helikopterom tako da se kreće unapred zadatom putanjom od jedne do druge tačke u praznom prostoru. Poželjno je zadati put preći što brže. Kako su helikopteri skupi, pretpostavka je da se učenje primarno obavlja u simulatoru. Agent je pilot helikoptera, a okruženje je simulator fizičkog prostora u kojem se nalazi helikopter. Stanje bi moglo biti definisano vektorom položaja težišta helikoptera u odnosu na neki fiksirani koordinatni sistem, izvodom tog vektora u odnosu na vreme (što je vektor brzine kretanja), vektorom orijentacije helikoptera (na primer, vektorom težište-nos) i njegovim izvodom u odnosu na vreme. Ovakva definicija stanja možda ne sadrži sve relevantne informacije. Na primer, drugi izvodi pomenutih vektora su takođe relevantni jer predstavljaju ubrzanja. Ukoliko se ovakva definicija stanja ispostavi manjkavom u praksi, može se probati sa drugačijom. Akcije se mogu definisati u*

terminima komandi koje pilot ima na raspolaganju. I takve akcije su neprekidne jer uključuju pomeranje upravljačke palice u neprekidnom prostoru. Ove akcije mogu se diskretizovati, ali postoje i algoritmi učenja koji to ne zahtevaju.

U ovom primeru i vreme je neprekidno, pa se u ovom slučaju radi o kontinualnom problemu učenja potkrepljivanjem. To znači da je potrebno ili diskretizovati vreme ili koristiti naprednije algoritme učenja koji bi, umesto sumiranja, vršili integraljenje nagrade. U ostatku primera pretpostavićemo da se vrši diskretizacija vremena.

Potrebno je definisati i nagrade. Helikopter treba da ostvari dva cilja. Prvi je da se drži zadate putanje, a drugi je da brzo pređe put. Nagrada koja se daje u svakom koraku mogla bi biti definisana kao zbir sledeće dve komponente. Jedna bi bila negativno rastojanje od definisane putanje, čime se favorizuje let po putanji ili u njenoj bliskoj okolini. Druga komponenta bila bi -1 čime bi bilo favorizovano brže kretanje. Naime, ako svaki korak nosi negativnu nagradu, kako bi ukupna nagrada bila veća, potrebno je problem rešiti u manjem broju koraka. Ove komponente mogu imati različite težinske faktore pri sumiranju. Ti težinski faktori treba da odgovaraju relativnom značaju navedenih ciljeva.

U navedenim primerima, od prvog ka poslednjem raste sloboda u modelovanju relevantnih elemenata problema učenja, pa čak i u definisanju samog problema, kao i kompleksnost rešenja. U praksi, situacija je često slična situaciji opisanoj u primeru 13.3. Ipak, primer 13.1 sakriva pravu složenost problema koji se rešava. To što je ponekad, kao u tom primeru, lako definisati osnovne elemente problema učenja, ne znači da je tako kreiran model problema pogodan za uspešno učenje.

Problem učenja potkrepljivanjem može se prirodno povezati sa pretragom. I u pretrazi figurišu određena stanja i mogu se preduzimati akcije (na primer, izbor grada u koji će se otići). Pretraga se obično sprovodi primenom računski zahtevnih algoritama na instancu problema koju je potrebno rešiti kako bi se našlo optimalno ili približno optimalno rešenje. U slučaju učenja potkrepljivanjem, kako bi se dobila politika, računski zahtevan proces učenja obavlja se na mnoštvu podataka proisteklih iz interakcija sa okruženjem u fazi treninga. Potom se naučena politika primenjuje za (obično) vrlo brzo aproksimativno rešavanje nove instance.

Naredna poglavlja bave se trima važnim temama. Prva je postavljanje formalnog okvira, Markovljevog procesa odlučivanja, koji opisuje okruženje i u kojem se razmatra interakcija agenta sa okruženjem. Druga je pronalaženje optimalnih politika u odnosu na tako definisano okruženje pri čemu pretpostavljamo da nam je njegov formalni opis (Markovljev proces odlučivanja) potpuno poznat. Ta pretpostavka obično nije ispunjena. Dodatno, nalaženje optimalne politike pod takvom pretpostavkom ne predstavlja učenje jer se ne oslanja na podatke iz iskustva, već predstavlja rešavanje (do na željenu tačnost) na osnovu potpunog i formalnog opisa problema. Ipak, koncepti i algoritmi koji će biti diskutovani pod ovom pretpostavkom (da je okruženje potpuno poznato) predstavljaju važan korak ka trećoj temi – algoritmima učenja u nepoznatom okruženju na osnovu iskustva iz interakcije sa njim.

13.1 Markovljevi procesi odlučivanja

Markovljev proces odlučivanja je uobičajeni formalni okvir u kojem se opisuju problemi koji se rešavaju učenjem potkrepljivanjem. Njegovo razumevanje je prvi korak ka konstrukciji algoritama u ovoj oblasti.

Definicija 13.1 (Markovljev proces odlučivanja)**.** Markovljev proces odlučivanja *(eng.* Markov decision process*) je uređena petorka $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \tau, p)$ gde je $\mathcal{S}$* skup stanja*, $\mathcal{A}$ je* skup akcija*, $\mathcal{R} \subseteq \mathbb{R}$ je* skup nagrada*, τ je* trajektorija*, odnosno niz slučajnih promenljivih*

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \ldots$$

takvih da važi $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$ i $R_t \in \mathcal{R}$ za svako $t \geq 0$, $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ je funkcija prelaska *takva da za svako $t \geq 0$ važi*

$$p(s', r|s, a) = P(S_{t+1} = s', R_t = r|S_t = s, A_t = a).$$

Ukoliko se iz nekog stanja ne može preći ni u jedno drugo, to stanje se naziva završnim stanjem*.*

U ovoj definiciji i u nastavku, P označava verovatnoću. Uloga funkcije prelaska je da definiše verovatnoće sa kojima se iz nekog stanja pri preduzetoj akciji prelazi u neko drugo stanje, a da se pri tome dobija određena nagrada. Primetimo da su funkcijom prelaska, između ostalog, određene nagrade u konkretnim situacijama, kao i da li se iz jednog stanja uopšte može doći u drugo stanje. Ukoliko se iz s ne može doći u s', onda je $p(s', r|s, a) = 0$ za svaku vrednost a i svaku vrednost r. Ukoliko se iz s može doći u s', onda je $p(s', r|s, a)$ različito od nule makar za jednu vrednost a i jednu vrednost r.

Ukoliko je za neki problem Markovljev proces odlučivanja poznat, govorimo o problemu sa poznatim okruženjem. U suprotnom, govorimo o problemu sa nepoznatim okruženjem. Isti problem može biti problem sa poznatim ili nepoznatim okruženjem, u zavisnosti od toga da li je Markovljev proces odlučivanja koji ga opisuje poznat ili ne. Jedan isti problem može da modeluje više različitih Markovljevih procesa odlučivanja.

Definicija 13.2 (Determinističko i stohastičko okruženje)**.** *Ako u datom Markovljevom procesu odlučivanja za svako stanje s i svaku akciju a, za tačno jedan par vrednosti s' i r važi $p(s', r|s, a) = 1$, dok za ostale parove važi $p(s', r|s, a) = 0$, onda se za funkciju prelaska i za okruženje kaže da su* deterministički*. U suprotnom, kaže se da su* stohastički*.*

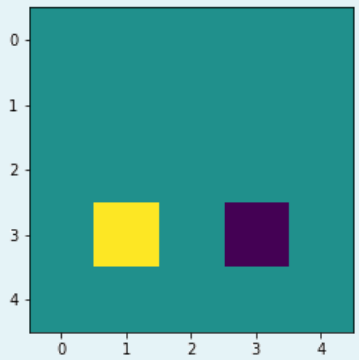
Primer 13.4. *Prethodni primeri predstavljaju Markovljeve procese i već ilustruju koncepte stanja, akcija i nagrada, pa samim tim i koncepta trajektorije koja predstavlja niz tih veličina. Funkcija prelaska u slučaju šaha i igre Pong je deterministička. Na primer, u slučaju šaha, potezom a iz datog stanja s prelazi se u jedinstveno određeno stanje s', pri čemu se dobija (jednoznačno određena) nagrada r u skladu sa opisanom definicijom nagrada. Verovatnoća da se odigravanjem istog poteza u istom polaznom stanju pređe u neko stanje $s'' \neq s'$ ili da se pri tome dobije neka nagrada $r' \neq r$ jednaka je* 0*.*

U slučaju vožnje helikoptera, praktično upotrebljiv opis stanja ne može uzeti u obzir sve zamislive faktore koji utiču na kretanje helikoptera. Zato, u zavisnosti od faktora koji nisu uzeti u obzir, helikopter iz jednog stanja pri datoj akciji može preći u različita, iako bliska, nova stanja. Odatle sledi da funkcija prelaska ne mora biti deterministička.

Razmotrimo dva jednostavna primera na kojima ćemo ilustrovati glavne koncepte koji se izlažu u ovoj glavi, kako ih budemo uvodili.

Primer 13.5. *Pas ima vlasnika. Pas predstavlja agenta, a vlasnik okruženje. Pas može da preduzme akcije „ustati“ i „sesti“, po svom izboru. Vlasnik može psu izdati komandu „sedi!“ ili „stoj!“ (tj. može tražiti psu da sedi ili da stoji) — to su moguća stanja okruženja. Pored njih, smatraćemo da postoji još samo završno stanje „kraj“. Početno stanje može biti jedno od postojeća dva, sa jednakim verovatnoćama. Ukoliko pas uradi kako mu vlasnik kaže, dobija keks. U suprotnom, ne dobija ništa. Dakle, nagrade mogu biti* 1 *ili* 0*. Pri bilo kojoj preduzetoj akciji, prelazi se u završno stanje.*

Primer 13.6. *Razmotrimo igru koja se odvija u jednostavnom svetu – na tabli dimenzija 5×5 polja, koja ima rupu umesto polja $(3, 3)$ i cilj na polju $(3, 1)$. Indeksiranje počinje od 0 i prvi indeks označava vrstu, a drugi kolonu. Tabla je prikazana narednom slikom.*



Agent se može kretati po tabli, a zadatak mu je da dođe do ciljnog polja. On može da se kreće po tabli po jedno polje levo, desno, gore ili dole. Skup stanja prirodno je skup polja table, a skup akcija je skup smerova kretanja $\{\leftarrow, \rightarrow, \uparrow, \downarrow\}$. Preduzimanje jedne takve akcije ne mora uvek da vodi do polja koje se nalazi u smeru koji je određen preduzetom akcijom, već se to dešava sa verovatnoćom β, a sa verovatnoćom $1 - \beta$ agent ide nasumice u jednom od preostala tri smera. Pad sa table (u rupu ili preko ivice) rezultuje nagradom -1 i krajem igre, dostizanje cilja nagradom 1 *i krajem igre, a ostali ishodi nagradom* 0*.*

Definicija Markovljevog procesa odlučivanja podrazumeva da, pri preduzimanju neke akcije, verovatnoća prelaska iz jednog stanja u drugo uz neku prateću nagradu ne zavisi ni od čega drugog osim od neposredno

prethodnog stanja i preduzete akcije. Drugim rečima, za Markovljev proces odlučivanja važi:

$$P(S_t, R_{t-1}|S_0, A_0, R_0, \ldots, S_{t-1}, A_{t-1}) = P(S_t, R_{t-1}|S_{t-1}, A_{t-1})$$

Ovo svojstvo se naziva *Markovljevim svojstvom* i govori da ako je poznato trenutno stanje procesa, za zaključivanje o njegovoj budućnosti nije potrebno poznavanje njegove prošlosti. Drugim rečima, ako znamo trenutno stanje, nije bitno kako se do njega stiglo. Na primer, u slučaju šaha, stanje table i informacija o tome koji je igrač na potezu daju dovoljno podataka za analizu daljeg toka igre. Kako se do tog stanja stiglo, za potrebe budućeg razmatranja potpuno je nebitno.[1]

Trajektorija izražava interakcije agenta sa okruženjem. Otud, ona zavisi delom od funkcije prelaska koja definiše okruženje, a delom od ponašanja agenta. Ponašanje agenta formalizuje se pomoću pojma *politike*.

Definicija 13.3 (Politika). *Neka je dat Markovljev proces odlučivanja sa skupom akcija $\mathcal{A}$ i skupom stanja $\mathcal{S}$. Uslovna raspodela verovatnoće nad akcijama $a \in \mathcal{A}$ za dato stanje $s \in \mathcal{S}$ naziva se* politika *(eng.* policy*) i označava $\pi(a|s)$. Politika je* deterministička *ukoliko je za svako stanje verovatnoća jedne od akcija jednaka 1, dok su verovatnoće ostalih akcija jednake 0. U suprotnom, politika je* stohastička.

Agent u stanju s preduzima akciju a sa verovatnoćom $\pi(a|s)$. Kako je ponašanje agenta opisano isključivo politikom, pojmovi agenta i njegove politike mogu se poistovetiti.

Primetimo da stohastičnost politike nema veze sa stohastičnošću okruženja. I u determinističkom i u stohastičkom okruženju ponašanje agenta može se modelovati i determinističkom i stohastičkom politikom.

Primer 13.7. *Pas iz primera 13.5, može se voditi različitim politikama. Na primer, može uvek uraditi ono što vlasnik traži. Može uvek da uradi suprotno. Ili može uvek stajati ili uvek sedeti, nezavisno od toga šta vlasnik kaže. Ovo su sve moguće determinističke politike. Jedna stohastička politika bila bi da sa jednakom verovatnoćom bira jednu od dve moguće akcije.*

Primer 13.8. *U slučaju problema iz primera 13.6, možemo formulisati različite politike. Na primer, jedna deterministička politika za svako stanje pridružuje verovatnoću 1 akciji kretanja udesno, a verovatnoću 0 svim ostalim akcijama. Možemo formulisati takve jednostavne politike i za druge smerove. Pored ovako jednostavnih konstantnih politika, mogli bismo u svakom polju izabrati tačno jedan smer kretanja koji bi imao verovatnoću 1, dok bi svi ostali imali verovatnoću 0. Determinističkih politika ima 4^{23}, jer za svako od 23 polja osim rupe i cilja možemo izabrati jedan smer na 4 načina. Jednostavan primer stohastičke politike je politika koja u svakom stanju svakoj akciji pridružuje verovatnoću 0.25.*

Kada je poznata politika π na osnovu koje agent donosi odluke, moguće je definisati zajedničku raspodelu promenljivih koje čine trajektoriju

$$\tau = S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \ldots$$

Ako je $P(S_0)$ raspodela mogućih polaznih stanja agenta, π politika prema kojoj agent donosi odluke i p funkcija prelaska, ta zajednička raspodela jednaka je:

$$P(\tau) = P(S_0) \prod_{t=0}^{\infty} \pi(A_t|S_t) p(S_{t+1}, R_t|S_t, A_t) \tag{13.1}$$

Niz konkretnih vrednosti slučajnih promenljivih koje čine trajektoriju, nazivamo *epizodom*. U slučaju nekih problema ima smisla govoriti o konačnim epizodama, kao u slučaju šaha ili u primeru sa psom, dok u nekim, kao u slučaju sveta iz primera 13.6, ima smisla govoriti o beskonačnim epizodama. Kako bi se izbegla diskusija po ovim slučajevima prilikom razvijanja teorije učenja potkrepljivanjem, pretpostavlja se da je trajektorija beskonačna uz dodatni trik za konačne epizode: u takvim problemima, uvodi se dodatno završno stanje u koje se prelazi bilo kojom akcijom iz svakog stanja u kojem se epizoda prirodno završava. U takvom dodatnom završnom stanju, svaka akcija ponovo vodi u to isto stanje. Dodatno, nagrada pri bilo kom prelasku u takvo završno stanje jednaka je nuli. Ova konvencija ne menja suštinski razmatrane probleme i u daljem tekstu ćemo je povremeno koristiti, ali bez eksplicitnog naglašavanja kako bismo pojednostavili izlaganje. Konkretnije, pri definisanju teorijskih koncepata konvenciju ćemo primenjivati, dok u primerima i algoritmima nećemo. Time je zaokružen formalni okvir u kojem se obično razmatra ponašanje agenta u nekom okruženju.

[1] Ovo nije sasvim tačno, zbog nekoliko izuzetaka: na primer, nekad se iz same pozicije ne može zaključiti da li je još uvek dozvoljeno sprovesti rokadu.

Primer 13.9. *U slučaju primera sa psom jedna epizoda bi mogla biti*

$$\textit{sedi!}, \textit{ustati}, 0, \textit{kraj}.$$

Ova epizoda se u skladu sa prethodnom konvencijom može beskonačno produžiti, ali uvek prelazeći u dodatno završno stanje sa nagradom 0*. U slučaju problema iz primera 13.6, epizoda bi mogla biti*

$$(0,0), \downarrow, 0, (1,0), \downarrow, 0, (2,0), \downarrow, 0, (3,0), \rightarrow, 1, (3,1).$$

I ona bi se mogla beskonačno produžiti.

13.2 Rešavanje Markovljevih procesa odlučivanja

Pod rešavanjem Markovljevih procesa odlučivanja podrazumeva se pronalaženje optimalne politike za agenta koji dela u okruženju definisanom tim Markovljevim procesom odlučivanja. Potrebno je najpre ustanoviti kako se meri kvalitet politika i šta znači da je politika optimalna. U tom cilju se uvodi nekoliko važnih pomoćnih koncepata. Potom je potrebno razmatrati algoritamske aspekte njenog izračunavanja. Kao što je već najavljeno, ovo poglavlje ne bavi se učenjem jer se u njemu ne vrši ocenjivanje vrednosti parametara nekog modela na osnovu podataka iz iskustva, već se Markovljev proces odlučivanja rešava (do na željenu tačnost) na osnovu potpunog opisa problema i poznatih verovatnoća prelaska. O učenju, kada te verovatnoće nisu poznate, biće reči u narednom poglavlju.

Naredna definicija uvodi pojam ukupnog *dobitka* koji agent ostvaruje u toku jedne epizode.

Definicija 13.4 (Dobitak)**.** *Za dati Markovljev proces odlučivanja,* dobitak *(eng.* gain*) počev od koraka t je slučajna promenljiva definisana kao red*

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

gde je γ realan broj iz intervala $[0,1]$ koji se naziva faktorom umanjenja *(eng.* discount factor*). Podrazumeva se da važi $0^0 = 1$.*

Na osnovu prethodne definicije i konvencije o predstavljanju svih epizoda kao beskonačnih, dobitak u završnom stanju je uvek 0. Razmotrimo ulogu hiperparametra γ. U slučaju da su nagrade ograničene, što je prihvatljiva pretpostavka, ako važi $\gamma < 1$, onda dobitak konvergira (za konačne epizode, sve nagrade počev od nekog stanja jednake su nuli, pa tada dobitak konvergira i ako je $\gamma = 1$). Pored toga, faktor umanjenja ima i intuitivni smisao — omogućava nam da vagamo značaj kratkoročnih i dugoročnih nagrada. Ukoliko važi $\gamma = 0$, politika koja maksimizuje dobitak bila bi pohlepna – u svakom koraku bila bi izabrana akcija koja nudi najveću nagradu koja se može dobiti u tom koraku. Što su vrednosti γ veće, to dobitak sa većom težinom uzima u obzir buduće nagrade.

Primer 13.10. *Ukoliko pas iz primera 13.5, kada mu vlasnik kaže „sedi!“, zaista sedne, dobiće nagradu* 1*. Kako se epizoda time završava i vrednost dobitka (od početnog stanja) će biti jednak* 1 *(bez obzira na to kolika je vrednost hiperparametra γ). Slično je i ako stoji kada mu vlasnik kaže „stoj!“. Ukoliko uradi suprotno, vrednost dobitka u takvoj epizodi će biti* 0*.*

Primer 13.11. *U slučaju sveta iz primera 13.6, agent se može kretati od gornjeg levog ugla tri koraka udesno, pa tri koraka nadole. Podsetimo se, pad sa table rezultuje nagradom -1, dostizanje cilja nagradom* 1*, a ostali ishodi nagradom* 0*. Ako je faktor umanjenja γ jednak* 0.9*, vrednost dobitka od polaznog trenutka je:*

$$\underbrace{1 \cdot 0 + 0.9 \cdot 0 + 0.9^2 \cdot 0}_{udesno} + \underbrace{0.9^3 \cdot 0 + 0.9^4 \cdot 0 + 0.9^5 \cdot (-1)}_{nadole} = -0.59$$

U slučaju da je faktor umanjenja jednak 0.99*, vrednost dobitka jednaka je* -0.95*.*

Kao i kod igranja igara, u učenju potkrepljivanjem važan je pojam *vrednosti stanja* (analogan vrednosti pozicije). Vrednost stanja je u vezi sa dobitkom koji agent može da ostvari od tog stanja. Kako i okruženje i politika mogu biti stohastički, dobitak od datog stanja može biti različit u zavisnosti od akcija koje agent

ubuduće preduzme. Zbog toga se vrednost stanja definiše pomoću očekivanja dobitka, pri čemu se misli na očekivanje u odnosu na zajedničku raspodelu $P(\tau)$ svih promenljivih u trajektoriji, to jest, na očekivanje po svim epizodama.

Definicija 13.5 (Funkcije vrednosti)**.** *Za dati Markovljev proces odlučivanja, očekivani dobitak koji agent dobija prateći politiku π iz nekog stanja s jednak je*

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

i naziva se funkcija vrednosti stanja *(eng.* state value function*). Očekivani dobitak koji agent dobija prateći politiku π nakon što u stanju s preduzme akciju a jednaka je*

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

i naziva se funkcija vrednosti akcije u stanju *(eng.* state-action value function*). U oba slučaja, očekivanje dobitka G_t računa se u odnosu na raspodelu datu formulom (13.1, strana 221).*

Naglasimo dva detalja u definiciji 13.5. Prvo, zbog Markovljevog svojstva, nebitno je o kom trenutku t se radi. Drugo, pošto je dobitak u završnom stanju jednak 0, i vrednost završnog stanja ili neke akcije u završnom stanju mora biti jednaka 0.

Primer 13.12. *Razmotrimo tri politike kojima se može voditi pas iz primera 13.5. Politika π_1 pretpostavlja da će pas sa verovatnoćom 1 poslušati vlasnika. Politika π_2 pretpostavlja da će sa verovatnoćom 1 uraditi suprotno. Politika π_3 pretpostavlja da će akcija biti izabrana nasumice između dve moguće akcije sa jednakom verovatnoćom.*

Izračunajmo funkcije vrednosti stanja u odnosu na ove politike. Kako će epizode sadržati samo jednu nagradu, vrednost hiperparametra γ nije bitna. U svim primerima pretpostavljamo da se prvom akcijom smatra akcija „sesti“, a drugom akcija „ustati“. Vrednost stanja kraj će biti 0, imajući u vidu da je to završno stanje i da nema daljih nagrada.

$$\begin{aligned} v_{\pi_1}(sedi!) &= \mathbb{E}_{\pi_1}[G_0 | S_0 = sedi!] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1 | S_0 = sedi!] \\ &= 1 \cdot (1 + \gamma \mathbb{E}_{\pi_1}[G_1 | S_1 = kraj]) + 0 \cdot (0 + \gamma \mathbb{E}_{\pi_1}[G_1 | S_1 = kraj]) \\ &= 1 \cdot 1 + 0 \cdot 0 = 1 \end{aligned}$$

Pri prelasku sa drugog na treći red, upotrebljena je definicija matematičkog očekivanja (u odnosu na raspodelu definisanu politikom), njegova linearnost i poznavanje funkcije prelaska.

$$\begin{aligned} v_{\pi_1}(stoj!) &= \mathbb{E}_{\pi_1}[G_0 | S_0 = stoj!] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1 | S_0 = stoj!] \\ &= 0 \cdot (0 + \gamma \mathbb{E}_{\pi_1}[G_1 | S_1 = kraj]) + 1 \cdot (1 + \gamma \mathbb{E}_{\pi_1}[G_1 | S_1 = kraj]) \\ &= 0 \cdot 0 + 1 \cdot 1 = 1 \end{aligned}$$

$$\begin{aligned} v_{\pi_2}(sedi!) &= \mathbb{E}_{\pi_2}[G_0 | S_0 = sedi!] \\ &= \mathbb{E}_{\pi_2}[R_0 + \gamma G_1 | S_0 = sedi!] \\ &= 0 \cdot (1 + \gamma \mathbb{E}_{\pi_2}[G_1 | S_1 = kraj]) + 1 \cdot (0 + \gamma \mathbb{E}_{\pi_2}[G_1 | S_1 = kraj]) \\ &= 0 \cdot 1 + 1 \cdot 0 = 0 \end{aligned}$$

$$\begin{aligned} v_{\pi_2}(stoj!) &= \mathbb{E}_{\pi_2}[G_0 | S_0 = stoj!] \\ &= \mathbb{E}_{\pi_2}[R_0 + \gamma G_1 | S_0 = stoj!] \\ &= 1 \cdot (0 + \gamma \mathbb{E}_{\pi_2}[G_1 | S_1 = kraj]) + 0 \cdot (1 + \gamma \mathbb{E}_{\pi_2}[G_1 | S_1 = kraj]) \\ &= 1 \cdot 0 + 0 \cdot 1 = 0 \end{aligned}$$

$$\begin{aligned} v_{\pi_3}(\textit{sedi!}) &= \mathbb{E}_{\pi_3}[G_0|S_0 = \textit{sedi!}] \\ &= \mathbb{E}_{\pi_3}[R_0 + \gamma G_1|S_0 = \textit{sedi!}] \\ &= 0.5 \cdot (1 + \gamma\mathbb{E}_{\pi_3}[G_1|S_1 = \textit{kraj}]) + 0.5 \cdot (0 + \gamma\mathbb{E}_{\pi_3}[G_1|S_1 = \textit{kraj}]) \\ &= 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5 \end{aligned}$$

$$\begin{aligned} v_{\pi_3}(\textit{stoj!}) &= \mathbb{E}_{\pi_3}[G_0|S_0 = \textit{stoj!}] \\ &= \mathbb{E}_{\pi_3}[R_0 + \gamma G_1|S_0 = \textit{stoj!}] \\ &= 0.5 \cdot (0 + \gamma\mathbb{E}_{\pi_3}[G_1|S_1 = \textit{kraj}]) + 0.5 \cdot (1 + \gamma\mathbb{E}_{\pi_3}[G_1|S_1 = \textit{kraj}]) \\ &= 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5 \end{aligned}$$

Možemo vrlo jednostavno izračunati i odgovarajuće funkcije vrednosti akcije u stanju. Uradićemo to samo za politiku π_1.

$$\begin{aligned} q_{\pi_1}(\textit{sedi!}, \textit{sesti}) &= \mathbb{E}_{\pi_1}[G_0|S_0 = \textit{sedi!}, A_0 = \textit{sesti}] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1|S_0 = \textit{sedi!}, A_0 = \textit{sesti}] \\ &= 1 + \gamma\mathbb{E}_{\pi_1}[G_1|S_1 = \textit{kraj}]) \\ &= 1 \end{aligned}$$

$$\begin{aligned} q_{\pi_1}(\textit{sedi!}, \textit{ustati}) &= \mathbb{E}_{\pi_1}[G_0|S_0 = \textit{sedi!}, A_0 = \textit{ustati}] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1|S_0 = \textit{sedi!}, A_0 = \textit{ustati}] \\ &= 0 + \gamma\mathbb{E}_{\pi_1}[G_1|S_1 = \textit{kraj}]) \\ &= 0 \end{aligned}$$

$$\begin{aligned} q_{\pi_1}(\textit{stoj!}, \textit{sesti}) &= \mathbb{E}_{\pi_1}[G_0|S_0 = \textit{stoj!}, A_0 = \textit{sesti}] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1|S_0 = \textit{stoj!}, A_0 = \textit{sesti}] \\ &= 0 + \gamma\mathbb{E}_{\pi_1}[G_1|S_1 = \textit{kraj}]) \\ &= 0 \end{aligned}$$

$$\begin{aligned} q_{\pi_1}(\textit{stoj!}, \textit{ustati}) &= \mathbb{E}_{\pi_1}[G_0|S_0 = \textit{stoj!}, A_0 = \textit{ustati}] \\ &= \mathbb{E}_{\pi_1}[R_0 + \gamma G_1|S_0 = \textit{stoj!}, A_0 = \textit{ustati}] \\ &= 1 + \gamma\mathbb{E}_{\pi_1}[G_1|S_1 = \textit{kraj}]) \\ &= 1 \end{aligned}$$

Funkcija vrednosti stanja i funkcija vrednosti akcije u stanju mogu se jednostavno povezati:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \sum_a \pi(a|s)\mathbb{E}[G_t|S_t = s, A_t = a] = \sum_a \pi(a|s)q_\pi(s, a)$$

Moguće je i izraziti funkciju q preko funkcije v:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}[R_t + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)\,(r + \gamma\mathbb{E}[G_{t+1}|S_{t+1} = s']) \\ &= \sum_{s',r} p(s', r|s, a)\,(r + \gamma v_\pi(s')) \end{aligned}$$

Primetimo da je lakše u terminima funkcije q izraziti funkciju v nego obratno. Naime, prvi smer podrazumeva da je, pored vrednosti funkcije q, poznata i politika π što je u praksi obično slučaj, pošto se politika koristi za upravljanje agentom. Drugi smer podrazumeva da je, pored vrednosti funkcije v, poznata i funkcija prelaska, to jest, da je poznato funkcionisanje okruženja, što često nije tačno, jer je okruženje teško precizno modelovati (osim u retkim slučajevima, poput nekih igara). Na primer, u slučaju vožnje helikoptera teško je precizno modelovati kretanja vazduha koja bi mogla uticati na ishode akcija pilota u određenim stanjima. Kako je u svakom stanju potrebno birati akciju, pa stoga i poznavati njenu vrednost u datom stanju, algoritmi učenja potkrepljivanjem u nepoznatom okruženju (koji će biti prikazani u narednim poglavljima) biće zasnovani na aproksimaciji funkcije q_π, a ne na aproksimaciji funkcije v_π.

Funkcije vrednosti omogućavaju merenje kvaliteta politika i definisanje parcijalnog poretka nad njima.

Definicija 13.6 (Parcijalni poredak nad politikama)**.** *Ukoliko u nekom Markovljevom procesu odlučivanja za sva stanja $s \in \mathcal{S}$ važi*

$$v_{\pi_1}(s) \geq v_{\pi_2}(s)$$

onda pišemo $\pi_1 \geq \pi_2$ i kažemo da je politika π_1 bolja ili jednaka od politike π_2 za taj Markovljev proces odlučivanja.

Sledeća teorema izražava fundamentalno svojstvo Markovljevih procesa odlučivanja.

Teorema 13.1. *Za svaki Markovljev proces odlučivanja postoji bar jedna politika π_* takva da za svaku politiku π važi $\pi_* \geq \pi$. Bilo koja takva politika naziva se* optimalnom politikom.

Primer 13.13. *Za psa iz primera 13.5, politika π_1 po kojoj pas sluša gazdu bolja je od politike π_2 po kojoj radi suprotno ili politike π_3 pri kojoj se ponaša nasumice. Štaviše, politika π_1 je optimalna.*

Sve optimalne politike dele istu funkciju vrednosti stanja i istu funkciju vrednosti akcije u stanju:

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

za sve $s \in \mathcal{S}$ i $a \in \mathcal{A}$. Ove funkcije vrednosti nazivamo optimalnim funkcijama vrednosti. Mogućnost da se optimalne politike razlikuju iako imaju iste funkcije vrednosti može delovati neintuitivno. Razmotrimo determinističku varijantu okruženja iz primera 13.6. Da bismo došli do cilja iz polja u gornjem levom uglu, svejedno je da li ćemo prvo krenuti nadole ili udesno. Zbog toga postoje barem dve optimalne politike koje se razlikuju u akciji koju preduzimaju na ovom polju, a vrednost polja je ista ako se agent nadalje ponaša optimalno.

Uočimo da važi prirodna veza između optimalnih funkcija vrednosti:

$$v_*(s) = \max_a q_*(s,a)$$

pošto se najveći očekivani dobitak u nekom stanju dobija preduzimanjem najbolje akcije u tom stanju (naravno, ovo se može i formalno dokazati).

Primetimo da poznavanje funkcije q_* omogućava izračunavanje bar jedne optimalne politike. Kako je u svakom stanju najpoželjnija akcija ona koja daje najveću očekivanu nagradu u datom stanju, sledeća (deterministička) politika je optimalna:

$$\pi_*(a|s) = \begin{cases} 1 & \text{ako važi } a = \arg\max_a q_*(s,a) \\ 0 & \text{inače} \end{cases}$$

Jedan način da se izračuna funkcija q_* pružaju takozvane *Belmanove jednakosti* koje izražavaju rekurentne veze funkcija vrednosti. Konkretno, za funkciju v_* važi:

$$\begin{aligned} v_*(s) &= \max_a q_*(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_t + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma \mathbb{E}_{\pi_*}[G_{t+1}|S_{t+1} = s']) \\ &= \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s')) \end{aligned}$$

Belmanova jednakost za funkciju q_* je:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma \max_{a'} q_*(s',a')\right)$$

Intuitivni smisao poslednje jednakosti je sledeći: najveći dobitak pri preduzimanju akcije a u stanju s zavisi od neposredno dobijene nagrade i najvećeg dobitka koji se može dobiti iz stanja u koje se dospe nakon preduzimanja akcije. Pritom, kako različiti prelasci u naredna stanja mogu biti različito verovatni, potrebno je neposredno dobijene nagrade i buduće dobitke množiti verovatnoćama tih prelaza kao težinskim faktorima.

Ove jednačine omogućavaju izračunavanje funkcija vrednosti narednim iterativnim postupcima, poznatim kao *iteriranje vrednosti* (eng. *value iteration*):

$$v(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v(s'))$$

za svako s, kao i

$$q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)\left(r + \gamma \max_{a'} q(s',a')\right)$$

za svako s i a. U slučaju konačnih skupova $\mathcal{S}$ i $\mathcal{A}$, iteriranje vrednosti konvergira ka v_* i q_* za proizvoljne inicijalne vrednosti funkcija v i q.

Kada su funkcije vrednosti aproksimirane, moguće je na osnovu njih aproksimirati optimalnu politiku, čime je Markovljev proces odlučivanja približno (a često i tačno) rešen. Ipak, ako su skupovi stanja i akcija velike kardinalnosti, algoritam iteriranja vrednosti će biti previše spor za praktičnu upotrebu. Ako su beskonačni, onda ovaj algoritam nije ni primenljiv. Takođe, kao što je rečeno ranije, ovaj postupak ne smatramo učenjem jer se radi o egzaktnom rešavanju problema (do na željenu tačnost) na osnovu formalnog opisa, a ne o ocenjivanju parametara na osnovu podataka dobijenih iz iskustva.

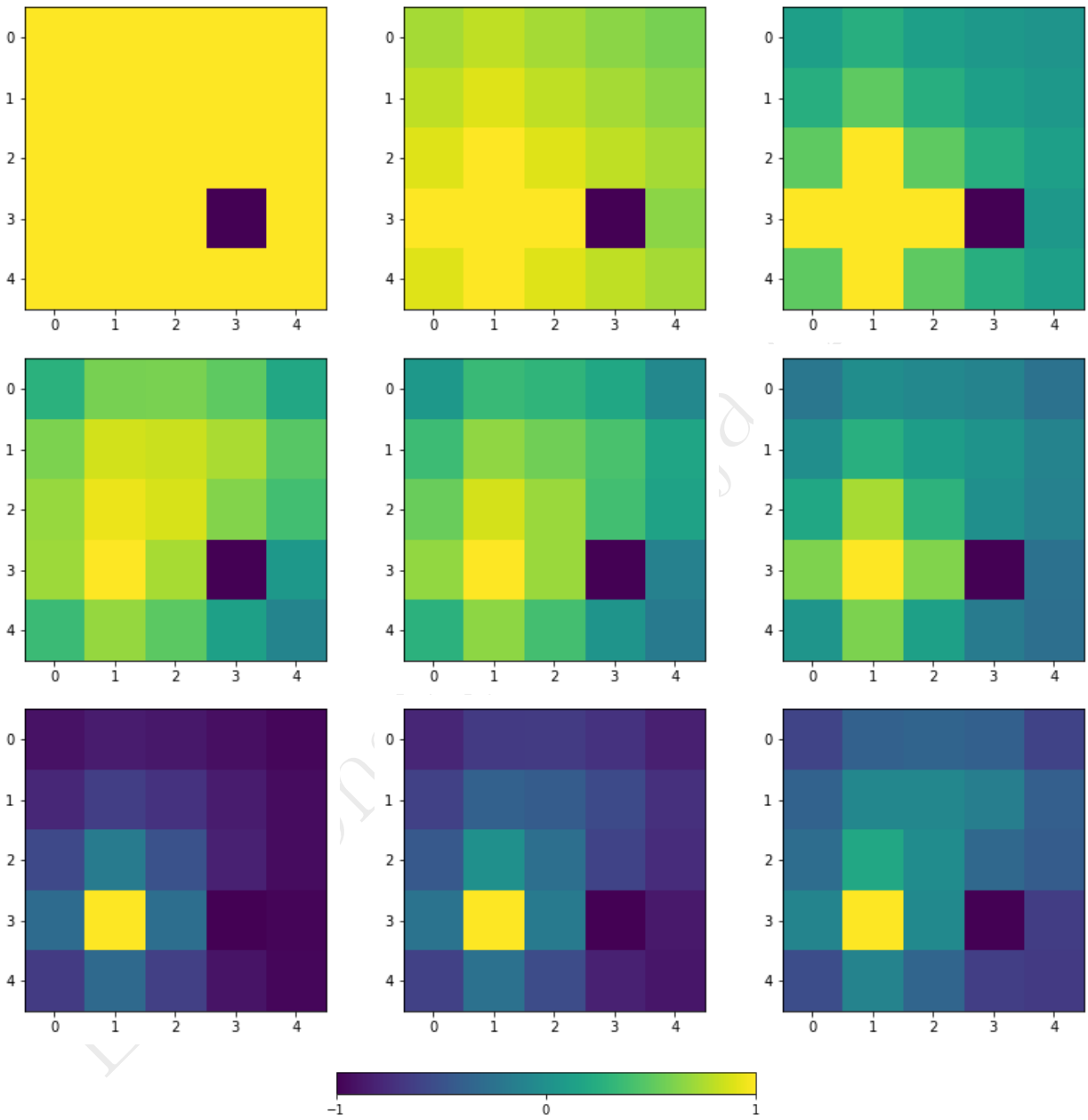
Primer 13.14. *U kontekstu primera 13.6, razmotrimo pitanje kakva je optimalna funkcija vrednosti stanja v_* za neke konkretne vrednosti verovatnoće β i hiperparametra γ. Funkcija v_*, izračunata iteriranjem vrednosti, ilustrovana je na slici 13.1 za različite njihove vrednosti: optimalna politika je da se za svako polje izabere akcija ka najintenzivnije žutom susednom polju.*

Opšti trend je da su vrednosti stanja veće ako je polje bliže cilju, a dalje od rupe i ivica, a da su manje ako je polje dalje od cilja, a bliže rupi i ivicama. U slučaju $\beta = 1$ i $\gamma = 1$, svako polje osim rupe ima vrednost 1. To je tako jer je sa svakog drugog polja moguće bez greške doći do cilja. Za $\beta = 1$, smanjivanjem faktora umanjenja, očekivano, polja bliža cilju postaju vrednija. Preciznije, ako je $\beta = 1$, vrednost svakog polja osim cilja i rupe je γ^{k-1} gde je k minimalan broj poteza potrebnih za dolazak do cilja. Ipak, to nije nužno tako i za druge vrednosti verovatnoće β. Za $\beta = 0.25$, tj. u slučaju nasumičnog kretanja, polja iznad ciljnog postaju vrednija sa smanjenjem hiperparametra γ. Naime, zbog eksponencijalnog umanjivanja budućih nagrada, nagrade na susednim poljima više doprinose vrednosti polja nego nagrade na daljim poljima. Stoga, mogućnost ostvarivanja nagrade 1 jednim potezom vodi povećanju ocenjene vrednosti tog polja u odnosu na polja dalja od cilja. Povećanje vrednosti tog polja dalje pogoduje povećanju vrednosti njemu okolnih polja.

Pri smanjivanju verovatnoće β, ponašanje agenta postaje sve nasumičnije i šanse da naiđe na polje sa nagradom se smanjuju, što se vidi kroz vrednosti polja.

13.3 Učenje u nepoznatom okruženju

Algoritam iteriranja vrednosti primenljiv je samo ako je okruženje poznato, odnosno ako je poznat Markovljev proces odlučivanja (skupovi stanja i akcija i funkcija prelaska). U realnim primenama to je retko ispunjeno. Iako nam funkcionisanje različitih okruženja može biti razumljivo na nekom nivou intuicije, to nije dovoljno za primenu prethodnog algoritma. U takvim situacijama, potrebno je paralelno sa određivanjem optimalne politike upoznavati i okruženje u kojem agent dela, što sugeriše da agent mora istraživati svoje okruženje preduzimajući akcije koje će ga odvesti u različita stanja. Agent bi to mogao postići preduzimanjem nasumičnih akcija u različitim stanjima, ali bi takav algoritam bio previše spor jer ne bi koristio nikakvu informaciju iz okruženja kao smernicu za efikasnije učenje. Postoji više principa na kojima se konstruišu upotrebljivi algoritmi učenja u nepoznatom okruženju i u ovom poglavlju ćemo razmotriti dva. Pritom, razmotićemo najpre učenje u konačnim prostorima stanja i akcija, a potom i učenje u beskonačnim prostorima koje nosi nove izazove.



Slika 13.1: Prikaz optimalne funkcije vrednosti stanja za različite vrednosti verovatnoće β i hiperparametra γ. Žuta boja označava vrednost 1, a tamno ljubičasta vrednost -1. Po redovima, verovatnoće β su 1, 0.7 i 0.25. Po kolonama, vrednosti hiperparametra γ su 1, 0.9 i 0.5.

13.3.1 Učenje u konačnim prostorima stanja i akcija

Kao što je već rečeno, algoritmi učenja potkrepljivanjem u nepoznatom okruženju obično su zasnovani na aproksimaciji funkcije q, a ne na aproksimaciji funkcije v. Primetimo da u Belmanovim jednakostima figuriše očekivanje zbira tekuće nagrade i vrednosti budućeg stanja:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma \max_{a'} q_*(s',a')\right)$$

Očekivanja se često ocenjuju uzoračkim prosekom, pri čemu se uzorak može dobiti tako što će agent interagovati sa okruženjem. U ekstremnom slučaju, taj uzorak može sadržati samo jedno opažanje koje se dobija preduzimanjem neke akcije a u nekom stanju s, pri čemu se prelazi u stanje s' i dobija nagrada r:

$$q_*(s,a) \approx r + \gamma \max_{a'} q_*(s',a')$$

Očigledan problem u ovom pristupu je to što funkcija q_* nije poznata – upravo ona se ocenjuje. Ključni korak u izvođenju novog algoritma je da se, umesto korišćenja stvarne vrednosti funkcije q_*, koristi funkcija kojom je aproksimiramo. Tu funkciju označićemo jednostavno sa q. Funkciju q ocenjujemo (tj. učimo) tako što zasebno ocenjujemo vrednosti $q(a,s)$ za svaki par stanje-akcija. Kako su, na osnovu pretpostavke, skupovi stanja i akcija konačni, poznavanje ovih vrednosti daje traženu reprezentaciju funkcije q. Kako se funkcija q menja u toku procesa njenog ocenjivanja, ovaj postupak potrebno je sprovoditi iterativno. Iterativni postupak omogućava i sakupljanje većeg broja opažanja. Ona se usrednjavaju pokretnim prosekom ocena funkcije vrednosti koje se dobijaju u različitim koracima. Preciznije, algoritam ocene funkcije vrednosti akcije u stanju zasniva se na sledećem pravilu ažuriranja koje se sprovodi pri prelazu iz stanja s u stanje s' preduzimanjem akcije a uz nagradu r:

$$q(s,a) \leftarrow (1-\alpha_t)q(s,a) + \alpha_t(r + \gamma \max_{a'} q(s',a'))$$

gde je $\alpha_t \in [0,1]$, za $t = 1,2,3,\ldots$ hiperparametar koji i ovde zovemo *brzinom učenja*, odnosno hiperparametar koji vaga značaj tekuće i novodobijene ocene u iteraciji t. Kao i u slučaju gradijentnog spusta (poglavlje 4.1), za konvergenciju je dovoljno da niz α_t zadovoljava uslove:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty\,.$$

Iako je definisano pravilo ažuriranja ocene pri jednoj akciji, algoritam nije do kraja definisan. Naime, pitanje je koju akciju agent preduzima u nekom stanju kako bi generisao prelaz. Može se pokazati da je za konvergenciju dovoljno preduzimati akcije tako da se svi parovi stanja i akcija posećuju beskonačno puta. Naravno, u praktičnim primenama će se taj proces odvijati u samo konačno mnogo koraka. Način na koji se ovo realizuje u praksi je korišćenje ε-pohlepne politike. ε-pohlepna politika u odnosu na datu funkciju q je politika koja sa verovatnoćom $1-\varepsilon$ bira najbolju akciju u odnosu na (tekuću) funkciju q, a preostale akcije sa međusobno jednakim verovatnoćama, odnosno:

$$\pi_{q,\varepsilon}(a|s) = \begin{cases} 1-\varepsilon & \text{ako } a = \arg\max_{a'} q(s,a') \\ \frac{\varepsilon}{|\mathcal{A}|-1} & \text{inače} \end{cases}$$

Hiperparametar ε treba da bude strogo veći od 0, ali se tipično smanjuje tokom treninga. Kada se završi jedna epizoda, tj. kada se u nekoj iteraciji dostigne završno stanje s', pokreće se nova epizoda, kako bi se nastavilo učenje. Kako je vrednost $q(s',a')$ za svako završno stanje s' i svaku akciju a' jednaka 0, izraz $\max_{a'} q(s',a')$ definisan je i u završnim stanjima s' i u njima ima vrednost 0. Jedna varijanta ovakvog postupka precizirana je algoritmom 13.2 i naziva se *q-učenje* (eng. *q-learning*).

Algoritam q-učenja je, naravno, moguće primeniti i na probleme u kojima je okruženje poznato. Na primer, za probleme sa velikim brojem stanja i akcija često se koriste varijante algoritama za učenje u nepoznatom okruženju, jer je za njih algoritam iteriranja vrednosti praktično neprimenljiv.

Primer 13.15. *Primer 13.5 je izrazito jednostavan i potpuno nezanimljiv za rešavanje ako je okruženje poznato. U slučaju da pretpostavimo da pas ne poznaje okruženje, tj. ne razume smisao komandi svog vlasnika, primer postaje ilustrativniji.*

U ovom primeru primenićemo algoritam q-učenja (algoritam 13.2) i sprovesti evaluaciju koja prati tok njegovog izvršavanja. Brzina učenja α_t je konstantna i jednaka 0.1 za svako t. Vrednost ε_t jednaka je $1/t$ ukoliko je taj količnik veći od 0.1, a u suprotnom je jednaka 0.1. Vrednost γ nije bitna pošto su sve epizode dužine 1 (jer se u ovom okruženju od polaznog stanja uvek jednom akcijom dolazi do završnog stanja). Koristićemo 100 epizoda, što u ovom konkretnom okruženju znači 100 iteracija, tj. važi $N = 100$. Funkcija

Algoritam: Algoritam q-učenja.

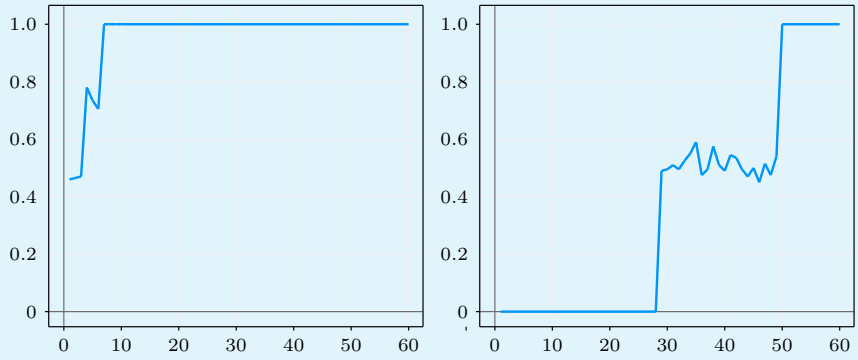
Ulaz: Broj iteracija N, niz vrednosti $\alpha_1, \alpha_2, \ldots, \alpha_N$ i niz vrednosti $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N$

Izlaz: Aproksimacija funkcije q_*

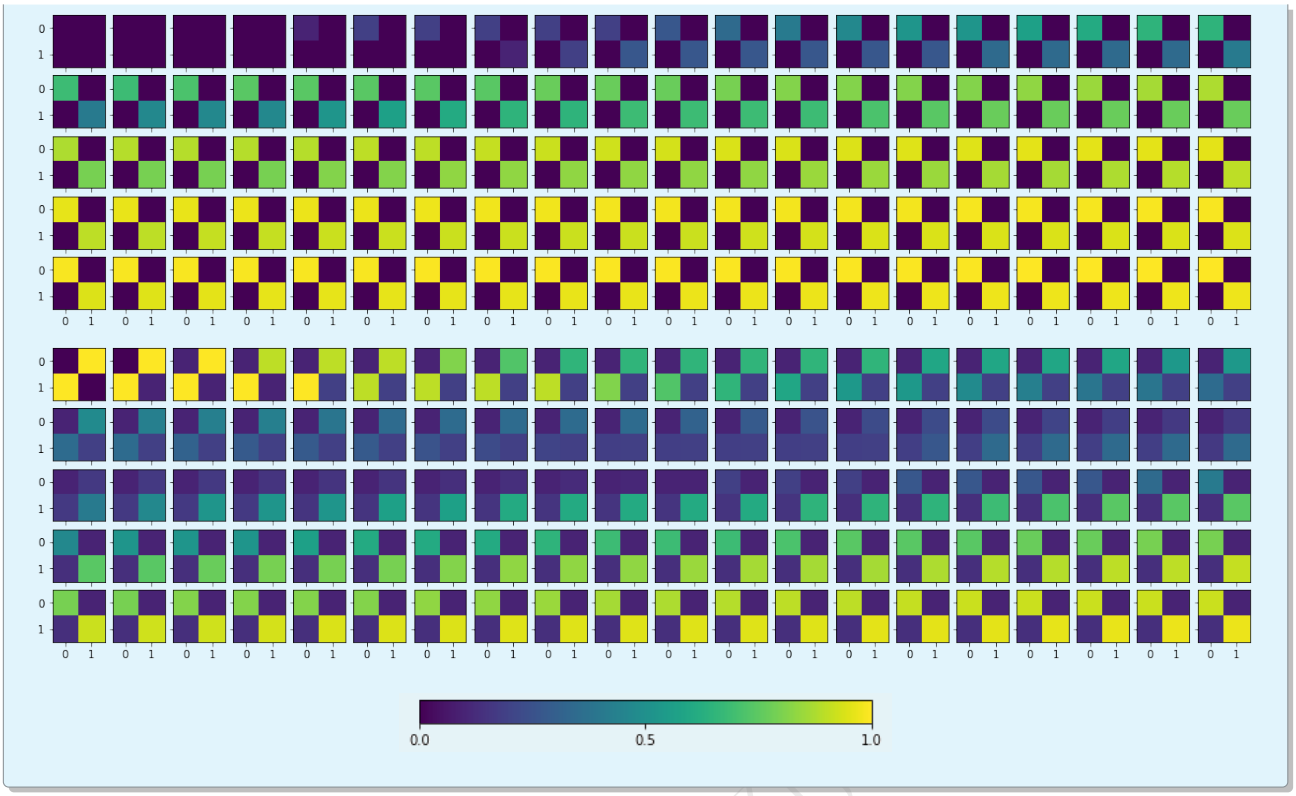
1: inicijalizuj q na 0 za sve parove s i a;
2: inicijalizuj stanje s na polazno stanje;
3: $t \leftarrow 1$;
4: **ponavljaj**
5: u stanju s preduzmi akciju a izabranu u skladu sa raspodelom $\pi_{q,\varepsilon_t}(\cdot|s)$ i opazi nagradu r i novo stanje s';
6: $q(s,a) \leftarrow (1-\alpha_t)q(s,a) + \alpha_t(r + \gamma \max_{a'} q(s',a'))$;
7: **ako** je stanje s' završno **onda**
8: postavi stanje s na polazno stanje;
9: **inače**
10: postavi stanje s na s';
11: $t \leftarrow t+1$;
12: **dok nije ispunjen** uslov $t > N$;
13: vrati q kao rešenje.

Slika 13.2: Algoritam q-učenja.

q je na početku inicijalizovana vrednošću 0 za sve parove stanja i akcija. Nakon svake epizode, privremeno se prekida izvršavanje algoritma q-učenja i evaluira se pohlepna politika izvedena iz tekuće funkcije q tako što se izračunava prosečna nagrada na osnovu 20 epizoda (koje počinju nasumično izabranim stanjima). Na narednoj slici (levo) prikazan je grafik takvih nagrada u odnosu na broj epizoda iz kojih je agent učio. Vidimo da je nakon 6 epizoda pas naučio optimalnu politiku (odnosno da uvek treba da posluša komande vlasnika). Pretpostavimo da pas na početku iz nekog razloga ima potpuno suprotno razumevanje vlasnikovih komandi. U tom slučaju, učenje je značajno sporije. Sa naredne slike (desno), vidi se da je potrebno mnogo više epizoda da bi se naučilo optimalno ponašanje.



Zanimljivo je prikazati i funkciju q u toku učenja. Na narednoj slici (gore), prikazane su vrednosti funkcije q kroz 100 epizoda pod pretpostavkom da je funkcija q inicijalizovana nulama. Svaka sličica predstavlja matricu dimenzija 2×2 gde vrste odgovaraju stanjima (0 – „sedi!“, 1 – „stoj!“), a kolone akcijama (0 – „sesti“, 1 – „ustati“). Boja odražava njihovu vrednost. Vidi se da algoritam još dugo konvergira ka pravim vrednostima funkcije q i nakon što je naučena optimalna politika. Ocenjivanje vrednosti q funkcije i jeste zahtevniji problem od učenja optimalne politike, ali u praksi najčešće nije neophodno. Prikaz funkcije q u toku učenja kada je inicijalizovana tako da pas bira suprotnu akciju od gazdine komande takođe je prikazan na narednoj slici (dole).



Prethodni primer demonstrira da epizoda može biti vrlo kratka i u tom slučaju, učenje se vrši iz potencijalno velikog broja epizoda. Takođe, procedura evaluacije upotrebljena u tom primeru, pa i u narednim, ne bi se koristila u praksi jer generiše veliki broj epizoda koje se ne koriste za učenje, ali je za potrebe ilustracije taj pristup dobar jer daje nepristrasnu ocenu kvaliteta tekuće politike.

Primer 13.16. *Demonstrirajmo ponašanje algoritma q-učenja i na okruženju iz primera 13.6. I u ovom okruženju Markovljev proces odlučivanja je poznat, što omogućava poređenje sa algoritmom iteriranja vrednosti. Za početak, pretpostavimo da je okruženje determinističko, odnosno da će se agent sigurno kretati u smeru koji odabere.*

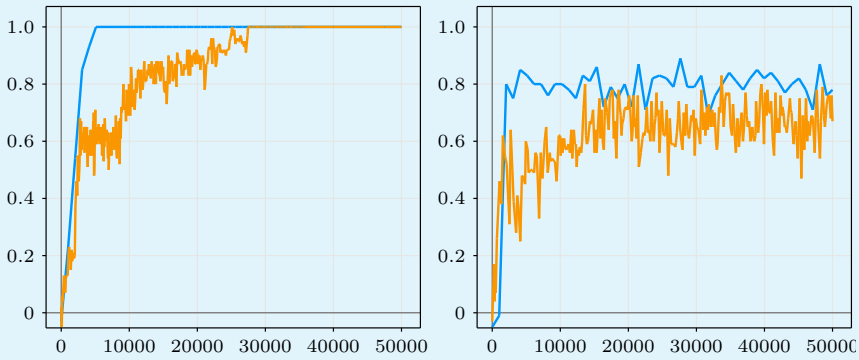
Algoritam q-učenja u ovom primeru koristi brzinu učenja $\alpha_t = 0.01$ *za svako* t. *Kao i u primeru 13.15, vrednost* ε_t *jednaka je* $1/t$ *ukoliko je taj količnik veći od* 0.1, *a u suprotnom je jednaka* 0.1. *Vrednost* γ *jednaka je* 0.99. *Svaka epizoda počinje na nasumično odabranom polju koje nije ni rupa ni ciljno polje. Broj iteracija je* $N = 15000$. *Algoritam uspeva da nauči politiku koja uvek vodi do ciljnog stanja za oko* 3000 *epizoda (ovaj broj može da varira zbog slučajnog izbora akcija pri učenju). Naučena politika ilustrovana je u narednoj tabeli. Strelice predstavljaju akciju sa najvećom vrednošću u datom stanju. Ciljno polje i rupa su prazni.*

↓	↓	←	↓	←
→	↓	↓	←	↑
→	↓	←	←	←
→		←		↓
↑	↑	←	←	←

Primetimo da akcija kretanja nagore na polju $(1, 4)$ *nije optimalna u smislu najvećeg umanjenog dobitka. Naime, finalna nagrada bila bi nešto veća ako bi put do cilja bio kraći, što bi se moglo postići kretanjem ulevo ili nadole. To ukazuje da je bilo prostora za dalje učenje. Ipak, i ova politika uvek vodi do ciljnog polja.*

Uporedimo ponašanje q-učenja sa algoritmom iteriranja vrednosti. Očekuje se da algoritam iteriranja vrednosti brže dolazi do optimalne politike, imajući u vidu da koristi znanje o poznatom Markovljevom procesu odlučivanja, dok q-učenje ne koristi tu informaciju. Ipak, kako je priroda ova dva algoritma značajno različita nije ih lako uporediti. U slučaju q-učenja mogli bismo pratiti unapređenje naučenog znanja u odnosu na broj epizoda iz kojih algoritam uči, ali kod iteriranja vrednosti ne postoji koncept epizoda. Ono što je zajedničko za oba algoritma je da vrše izračunavanja tekuće aproksimacije funkcije q i broj tih izračunavanja može se uzeti kao mera uloženog računskog truda u odnosu na koju se može izraziti kvalitet naučenog znanja.

Taj kvalitet može se ustanoviti usputnom evaluacijom, kao u primeru 13.15. U ovom slučaju je korišćeno 100 *epizoda za izračunavanje prosečne nagrade. Na narednoj slici (levo) prikazani su grafici prosečne nagrade u odnosu na broj izačunavanja vrednosti funkcije q za iteriranje vrednosti (plavo) i za q-učenje (narandžasto). Vidi se da iteriranje vrednosti brže i stabilnije postiže maksimalnu nagradu, što je i očekivano.*



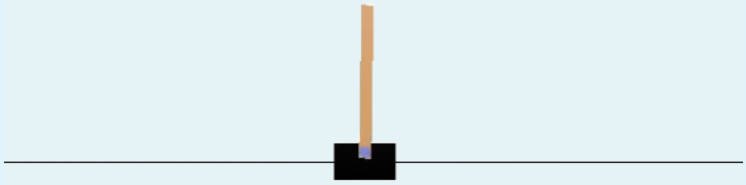
Razmotrimo i slučaj stohastičkog okruženja, pri čemu je verovatnoća kretanja u izabranom smeru 0.7*, a u svakom od preostalih* 0.1*. Grafici prosečne nagrade za oba algoritma prikazani su na prethodnoj slici (desno). U slučaju oba algoritma vidljiva je određena nestabilnost performansi. U slučaju iteriranja vrednosti, ona potiče samo od stohastičnosti okruženja prilikom evaluacije. U toku učenja, iteriranje vrednosti ima potpunu informaciju o okruženju i koristi tačne verovatnoće prelaska. S druge strane, q-učenje je podložno i stohastičnosti u vreme učenja i zbog toga dostiže nešto niže vrednosti prosečne nagrade.*

Algoritam q-učenja rešava problem nepoznavanja okruženja, ali ne i problem velikih ili beskonačnih skupova stanja i akcija, kakav je slučaj u diskutovanom primeru igranja šaha. Naime, u takvom slučaju nije jasno kako čuvati informacije o aproksimaciji $q(s,a)$ u računaru i kako dovoljno puta posetiti sve parove stanja i akcija tako da aproksimacija bude dovoljno dobra za praktične potrebe. U slučaju šaha, nezamislivo je da će sva moguća stanja biti viđena prilikom treninga. Dodatno, ovako treniran agent se ne može snaći ni u jednom stanju koje ranije nije video, čak i ako je to stanje slično već viđenim. Drugim rečima, algoritam ne generalizuje na nepoznate situacije te je i njega teško smatrati algoritmom učenja. Ovi problemi se otklanjaju pristupima kakve diskutujemo u narednom potpoglavlju.

13.3.2 Učenje u beskonačnom prostoru stanja i konačnom prostoru akcija

U velikom broju praktično relevantnih primena, prostor stanja je beskonačan ili je konačan ali previše veliki za prethodno izložene metode. Nekada nije ni diskretan kao što te metode pretpostavljaju, već kontinualan. Zbog toga nije moguće koristiti algoritam q-učenja (slika 13.2) u kojem se funkcija q reprezentuje kao skup vrednosti za sve pojedinačne parove stanje-akcija, te je potrebno definisati nove pristupe. Rešenje ovog problema diskutovaćemo najpre uz pretpostavku o konačnosti skupa akcija, što je dovoljno dobro za mnoštvo primena.

Primer 13.17. *Problem balansiranja štapa na kolicima dobro je poznat primer problema upravljanja u neprekidnim a samim tim i beskonačnim prostorima stanja. Pretpostavlja se da kolica mogu da se kreću levo-desno po podlozi bez trenja i da je štap za njih zakačen pomoću zgloba oko kojeg može slobodno da rotira. Na narednom prikazu iz kolekcije okruženja OpenAI Gym, štap je prikazan braon bojom, a kolica crnom.*



U polaznom stanju, štap je skoro uspravan, stoji pod malim uglom u odnosu na vertikalu. Na kolica je moguće

primenjivati jediničnu silu ulevo ili udesno, što dovodi do njihovog kretanja u datom smeru. Interakcija se odvija u diskretnim trenucima, sa razmakom od 0.02 *sekunde. Epizoda se završava ako se kolica udalje* 2.4 *jedinice od polaznog stanja, ako je ugao štapa u odnosu na vertikalu veći od* 12 *stepeni ili ako je prošlo* 200 *koraka. U ovom problemu, skup akcija je očigledan – primeniti silu na kolica ili ulevo ili udesno. Skup stanja je skup četvorki realnih brojeva, koje uključuju koordinatu kolica duž ose levo-desno, brzinu kolica, ugao štapa i ugaonu brzinu štapa. Ovo su očito neprekidne veličine. Nagrada je* 1 *za svaki korak do kraja epizode. Kao i uvek, cilj je maksimizovati dobitak od početnog trenutka.*

Ključno zapažanje učenja potkrepljivanjem u beskonačnim prostorima stanja je da se funkcija q_* može učiti, kao i u drugim pristupima mašinskog učenja, uvođenjem parametrizovanog modela $q_{\mathbf{w}}$ nad nekom reprezentacijom stanja i akcija. Kada je skup akcija konačan (što je tekuća pretpostavka), nije neophodno tražiti posebnu reprezentaciju za njih – mogu se tretirati kao bilo koje kategoričke promenljive. Što se tiče stanja, ili je potrebno predstaviti ih unapred definisanim skupom svojstava ili je potrebno upotrebiti model koji je u stanju da sam izabere takva svojstva i time kreira pogodnu reprezentaciju (kao što mogu duboke neuronske mreže ako se stanja predstavljaju slikama kao, na primer, u video igrama).

Razmotrićemo jedan algoritam inspirisan algoritmom q-učenja. Pravilo ažuriranja aproksimacije funkcije q u q-učenju je:

$$q(s,a) \leftarrow (1-\alpha_t)q(s,a) + \alpha_t(r + \gamma \max_{a'} q(s',a'))$$

gde je s' stanje u koje se prelazi preduzimanjem akcije a u stanju s i koje je poznato iz konkretne epizode i gde je t redni broj iteracije, a α_t brzina učenja. Jednostavnim preuređivanjem članova dobija se sledeće:

$$q(s,a) \leftarrow q(s,a) + \alpha_t(r + \gamma \max_{a'} q(s',a') - q(s,a))$$

Algoritam inspirisan ovim pravilom, a koji je zasnovan na parametarskoj aproksimaciji $q_{\mathbf{w}}$ funkcije q, koristi sledeće pravilo ažuriranja parametara:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_t(r + \gamma \max_{a'} q_{\mathbf{w}}(s',a') - q_{\mathbf{w}}(s,a))\nabla_{\mathbf{w}} q_{\mathbf{w}}(s,a) \tag{13.2}$$

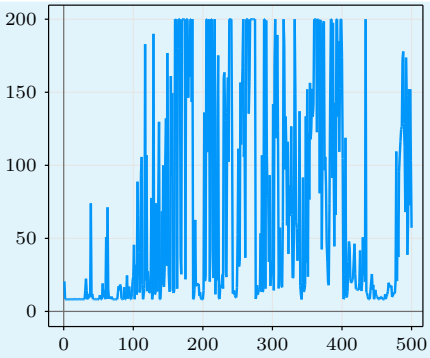
Ostatak algoritma biće isti kao u q-učenju (algoritam 13.2). Dve navedene iterativne operacije imaju jedan analogni element, ali da se razlikuju po tome što se u prvoj ažurira vrednost $q(s,a)$ a u drugoj vrednost $\mathbf{w}$, kao i po tome što se u drugoj operaciji javlja gradijent aproksimacije $q_{\mathbf{w}}$ po parametrima $\mathbf{w}$. Kako je skup akcija konačan, odgovarajuća politika π definiše se prirodno, na sledeći način:

$$\pi(a|s) = \begin{cases} 1 & \text{ako važi } a = \arg\max_a q_{\mathbf{w}}(s,a) \\ 0 & \text{inače} \end{cases}$$

Ukoliko funkcija $q_{\mathbf{w}}(s,a)$ predstavlja duboku neuronsku mrežu, prethodni algoritam naziva se *dubokim q-učenjem* (eng. *deep q-learning*). Ključni problem u primeni ovakvog algoritma je to što on ne mora konvergirati. Postoji niz poboljšanja koja vode ka pouzdanijoj i bržoj konvergenciji, ali po pravilu su algoritmi učenja potkrepljivanjem, zbog nestabilnosti optimizacije, teži za primenu od algoritama nadgledanog učenja.

Primer 13.18. *Pri primeni algoritma dubokog q-učenja u problemu balansiranja štapa, za aproksimaciju $q_{\mathbf{w}}$ funkcije q_*, koristićemo vrlo jednostavnu neuronsku mrežu. Mreža ima* 4 *ulaza, pošto je stanje opisano pomoću* 4 *realna broja. Ima dva izlaza koji aproksimiraju vrednosti funkcije q za dve akcije (levo i desno) u datom ulaznom stanju. Vrednost aproksimacije $q_{\mathbf{w}}(s,a)$ dobija se tako što se mreži na ulazu dâ stanje s i očita se vrednost na izlazu koji odgovara akciji a. Arhitektura je mogla biti i drugačije definisana (na primer, tako što bi i s i a bili dati na ulazu mreže, a vrednost $q_{\mathbf{w}}(s,a)$ na izlazu). Koristićemo* 3 *sloja sa po* 20 *neurona i ReLU aktivacione funkcije. Vrednost ε ε-pohlepne politike je $1/t$ gde je t redni broj epizode. Za brzinu učenja α korišćena je vrednost* 0.005.

Evaluacija se sprovodi kao u primeru 13.15. Nakon svake epizode učenja, računa se prosečna nagrada (uprosečena na 100 *epizoda testiranja) za tekuću politiku. Naredna slika prikazuje prosečnu nagradu u odnosu na do tada upotrebljeni broj epizoda za treniranje.*



Sa slike se vidi da je algoritam u stanju da reši problem (odnosno da dostigne ukupnu nagradu 200*) u prvih* 200 *epizoda učenja, ali se takođe vidi i nestabilnost usled koje model može da izgubi na kvalitetu, ali i da se potom ponovo popravlja. Ukoliko algoritam u nekoj iteraciji uspe da nađe dobar model, nije ni bitno da li će posle toga oscilovati. Naime moguće je zapamtiti tekući model kad god on po prosečnoj nagradi prevaziđe prethodno zapamćeni. Kada se trening završi, u primeni se može koristiti poslednji zapamćeni model.*

13.3.3 Učenje u beskonačnom prostoru stanja i beskonačnom prostoru akcija

U slučaju beskonačnih prostora akcija, algoritam dubokog q-učenja nije lako upotrebljiv. Naime, čak i ako bi uspešno bila aproksimirana funkcija $q_*(s,a)$, za zadato stanje s, najbolja akcija a (jedna od beskonačno mnogo) bi se uvek morala tražiti nekim optimizacionim algoritmom nad funkcijom $q_{\mathbf{w}}(s,a)$. Funkcija vrednosti q, u slučaju beskonačnog skupa akcija, jednostavno ne predstavlja pogodnu reprezentaciju za određivanje tražene politike. Umesto nje, alternativni pristup, koji se često koristi u učenju potkrepljivanjem (a može biti korišćen i u slučaju konačnog prostora akcija), zasniva se na direktnom učenju politike $\pi_{\mathbf{w}}$ (definisane parametrima $\mathbf{w}$), umesto učenja funkcije q_*. Primetimo da je u slučaju neprekidnih akcija koje odgovaraju realnom intervalu, politiku $\pi_{\mathbf{w}}(a|s)$ moguće modelovati, na primer, normalnom raspodelom $\mathcal{N}(f_{\mathbf{w}}(s),\sigma^2)$ za neko σ^2, gde $f_{\mathbf{w}}(s)$ može da predstavlja neuronsku mrežu koja vrši regresiju. U tom slučaju, ta mreža za konkretno, dato stanje s direktno izračunava preporučene akcije. Naredni pristup može se koristiti i u slučaju neprekidnih i u slučaju diskretnih skupova akcija.

Kao što je rečeno, cilj učenja potkrepljivanjem je maksimizovanje očekivanog dobitka:

$$J(\mathbf{w}) = \mathbb{E}_{P_{\mathbf{w}}}[G_0]$$

pri čemu je $P_{\mathbf{w}}$ zajednička raspodela promenljivih u trajektoriji u slučaju korišćenja politike $\pi_{\mathbf{w}}$:

$$P_{\mathbf{w}}(\tau) = P(S_0)\prod_{t=0}^{\infty}\pi_{\mathbf{w}}(A_t|S_t)p(S_{t+1},R_t|S_t,A_t)$$

Za maksimizaciju očekivanja možemo, kao i u drugim sličnim situacijama, da koristimo metod gradijentnog uspona. Gradijent $\nabla_{\mathbf{w}}J(\mathbf{w})$ se može izvesti kao u nastavku (nisu navedena obrazloženja izvođenja). U izvođenju, $\mathbf{e}$ označava konkretnu epizodu trajektorije τ, odnosno niz konkretnih vrednosti slučajnih promenljivih koje čine niz τ. Konkretna vrednost dobitka G_0 za epizodu $\mathbf{e}$, tj. vrednost $\sum_{t=0}^{\infty}\gamma^t r_t$ označena je sa $r(\mathbf{e})$. Očekivani dobitak u odnosu na raspodelu $P_{\mathbf{w}}(\tau)$, izračunava se uobičajeno, tako što se integrale vrednosti $r(\mathbf{e})$ pomnožene sa njihovim verovatnoćama $P_{\mathbf{w}}(\mathbf{e})$.

$$\begin{aligned}
\nabla_{\mathbf{w}}J(\mathbf{w}) &= \nabla_{\mathbf{w}}\mathbb{E}_{P_{\mathbf{w}}}[G_0]\\
&= \nabla_{\mathbf{w}}\int r(\mathbf{e})P_{\mathbf{w}}(\mathbf{e})d\mathbf{e}\\
&= \int r(\mathbf{e})\nabla_{\mathbf{w}}P_{\mathbf{w}}(\mathbf{e})d\mathbf{e}\\
&= \int r(\mathbf{e})\frac{\nabla_{\mathbf{w}}P_{\mathbf{w}}(\mathbf{e})}{P_{\mathbf{w}}(\mathbf{e})}P_{\mathbf{w}}(\mathbf{e})d\mathbf{e}\\
&= \int r(\mathbf{e})\left(\nabla_{\mathbf{w}}\log P_{\mathbf{w}}(\mathbf{e})\right)P_{\mathbf{w}}(\mathbf{e})d\mathbf{e}\\
&= \mathbb{E}_{P_{\mathbf{w}}}[G_0\nabla_{\mathbf{w}}\log P_{\mathbf{w}}(\tau)]
\end{aligned}$$

Dalje, važi

$$\nabla_{\mathbf{w}} \log P_{\mathbf{w}}(\mathbf{e}) = \nabla_{\mathbf{w}} \left(\log P(s_0) + \sum_{t=0}^{\infty} (\log \pi_{\mathbf{w}}(a_t|s_t) + \log p(s_{t+1}, r_t|s_t, a_t)) \right)$$
$$= \sum_{t=0}^{\infty} \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t)$$

odnosno

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{P_{\mathbf{w}}} \left[G_0 \sum_{t=0}^{\infty} \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t) \right]$$

Navedeni gradijent, koji je izražen očekivanjem, uobičajeno se aproksimira uzoračkim prosekom:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \approx \frac{1}{N} \sum_{i=1}^{N} r(\mathbf{e}_i) \sum_{t=0}^{T_i} \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_{it}|s_{it}).$$

gde je T_i dužina i-te epizode. Na ovoj činjenici zasnovan je algoritam Reinforce prikazan u svom najjednostavnijem obliku algoritmom 13.3. U ovoj formulaciji pretpostavlja se da se gradijent aproksimira na osnovu uzorka koji čini samo jedna epizoda:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) \approx r(\mathbf{e}) \sum_{t=0}^{T} \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t).$$

Algoritam: Algoritam Reinforce.

Ulaz: Broj iteracija N

Izlaz: Aproksimacija $\pi_{\mathbf{w}}$ optimalne politike

1: nasumično inicijalizuj $\mathbf{w}$;
2: $t \leftarrow 1$;
3: **ponavljaj**
4: generisati epizodu $\mathbf{e} = s_0, a_0, r_0, s_1, \ldots, s_T, a_T, r_T$ korišćenjem politike $\pi_{\mathbf{w}}$;
5: izračunati nagradu $r(\mathbf{e})$;
6: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(r(\mathbf{e}) \sum_{t=0}^{T} \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(a_t|s_t) \right)$;
7: $t \leftarrow t + 1$;
8: **dok nije ispunjen** uslov $t = N$;
9: vrati $\pi_{\mathbf{w}}$ kao rešenje.

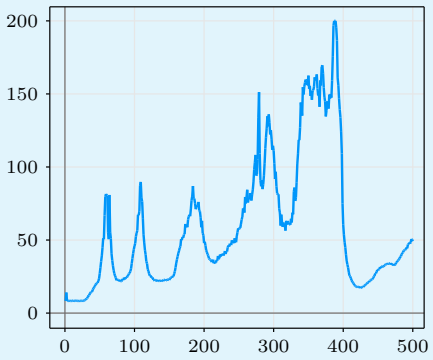
Slika 13.3: Algoritam Reinforce.

U algoritmu Reinforce, ne koriste se determinističke politike. Jedan razlog je to što je izvod takve politike u odnosu na parametre $\mathbf{w}$ ili jednak nuli ili nedefinisan, te stoga nije upotrebljiv za ažuriranje parametara. Drugi razlog je vezan za diverzitet epizoda koje se pomoću njih u ovom algoritmu mogu generisati. Ukoliko je polazno stanje uvek isto i ukoliko je okruženje determinističko (što je slučaj, na primer, u mnogim igrama), za datu politiku, uvek će biti generisana ista epizoda. Zato se u algoritmu Reinforce obično koristi neka diferencijabilna reprezentacija politike koja daje pozitivnu (iako možda malu) verovatnoću svim akcijama da budu izabrane u svakom stanju. Na primer, ako je skup akcija konačan, pogodna reprezentacija je neuronska mreža sa onoliko izlaza koliko postoji akcija i softmax aktivacionom funkcijom na izlazu.

Algoritam Reinforce osnova je više algoritama učenja potkrepljivanjem korišćenih u praksi. Njegova mana u odnosu na algoritme zasnovane na q funkciji je veća nestabilnost treninga koja se često ublažava dodatnim tehnikama koje izlaze iz okvira ove knjige.

Primer 13.19. *Algoritam Reinforce se takođe može primeniti na problem balansiranja štapa. Za mrežu koja modeluje funkciju $\pi_{\mathbf{w}}$ korišćena je ista arhitektura i ista brzina učenja kao za modelovanje funkcije $q_{\mathbf{w}}$ u slučaju algoritma dubokog q-učenja (videti primer 13.18), uz korišćenje softmax funkcije na izlazima, kako bi se modelovala politika koja predstavlja raspodelu nad dve moguće akcije. Sa naredne slike vidi se da je i algoritam Reinforce uspešan u rešavanju ovog problema, ali da do rešenja dolazi tek nakon skoro* 400

epizoda.



Sprovedeni eksperimenti sugerišu da se algoritam dubokog q-učenja pokazao znatno boljim pri rešavanju ovog problema kako po brzini rešavanja problema, tako i po obimu eksperimenata koji je bio dovoljan da se dođe do zadovoljavajućih rezultata. Ipak, taj zaključak se ne može lako uopštiti na druge probleme, pa je u praksi često potrebno razmotriti više različitih algoritama.

13.4 Primene učenja potkrepljivanjem

U nastavku prikazujemo nekoliko primera primena učenja potkrepljivanjem. Naglasimo da do sada diskutovani algoritmi, iako se mogu primeniti na date probleme, ne bi na njima dali rezultate upotrebljive u praksi. Algoritmi stvarno korišćeni u praksi su, iako izgrađeni na sličnim osnovima, ipak znatno komplikovaniji.

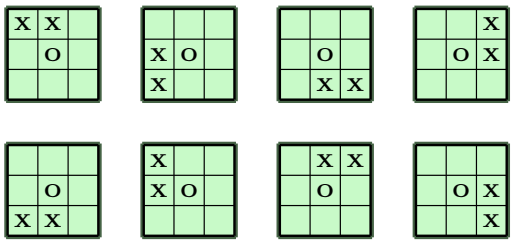
13.4.1 Učenje potkrepljivanjem u strateškim igrama za dva igrača na primeru igre iks-oks

Igranje igara jedan je od tradicionalnih izazova veštačke inteligencije. Jedna zanimljiva klasa igara su strateške igre u kojima dva igrača igraju jedan protiv drugog i najviše jedan od njih pobeđuje (videti glavu 5). Takve igre su šah, iks-oks, go i druge. Neki od najimpresivnijih uspeha mašinskog učenja su uspesi baš na ovom polju. Konkretno, učenjem potkrepljivanjem trenirani su agenti koji u šahu, gou i drugim srodnim igrama nadmašuju najbolje ljudske igrače. Jedan od najpoznatijih ovakvih sistema (čiji opis izlazi iz okvira ove knjige) je AlphaZero koji kombinuje učenje potkrepljivanjem, neuronske mreže (videti poglavlje 11.5) i Monte Karlo pretragu stabla igre (videti poglavlje 5.4.9).

Specifičnost problema učenja u ovom kontekstu je postojanje dva igrača. Ukoliko se igrači modeluju po jednim agentom, oba agenta mogu se smatrati delom okruženja drugog agenta. Postavlja se pitanje na koji način onda jedan agent treba da igra dok obučavamo drugog. Ukoliko bi igrao po nekoj predefinisanoj strategiji, drugi agent bi možda naučio da efikasno igra samo protiv te strategije. Ukoliko bi igrao nasumice, možda bi bio potreban ogroman broj odigranih partija kako bi drugi agent naučio da dobro igra jer bi agent koji igra nasumice retko igrao dobro. Pristup koji se za učenje u ovakvom kontekstu često koristi poznat je kao *trening igranjem protiv samog sebe* (eng. *self-play*). U ovom pristupu, postoji samo jedan agent, on bira poteze za oba igrača i učenje se vrši na osnovu poteza oba igrača (umesto da jedan od igrača i njegovi potezi budu tretirani samo kao deo okruženja). U nastavku ćemo ovaj pristup ilustrovati samo na primeru igre iks-oks, ali sa osvrtima i na neke druge igre.

U igri iks-oks, okruženje čini tabla dimenzija 3×3 na kojoj igrači naizmenično upisuje simbole **X** i **O** na prazna polja. Inicijalno su sva polja prazna. Partija se završava nakon prvog poteza nakon kojeg se na tabli vertikalno, horizontalno ili dijagonalno nalaze tri ista simbola (i tada je pobedio igrač čiji je to simbol) ili kad su sva polja popunjena (i tada je ishod nerešen). Svaka konfiguracija table sa različitim upisanim simbolima predstavlja po jedno stanje. Partija počinje potezom igrača **X** (pa opis stanja implicitno sadrži i informaciju o tome koji igrač je na potezu). Igra se može modelovati Markovljevim procesom odlučivanja na sledeći način. Agent igra naizmenično u ime igrača **X** pa u ime igrača **O**. Nakon svakog poteza agent dobija nagradu 0, osim ukoliko su nakon poslednjeg poteza na tabli duž nekog pravca upisana tri simbola **X** kada dobija nagradu 1, i ukoliko su na tabli duž nekog pravca upisana tri simbola **O**, kada dobija nagradu -1.

Algoritam q-učenja se na ovako definisan problem ne može primeniti direktno, jer takva primena ne bi odgovarala igri dva suprotstavljena igrača (već bi oba igrača stremila da maksimizuju nagradu, a ona je najveća kada završno stanje sadrži niz od tri simbola **X**). Jedno moguće rešenje sastoji se u promeni perspektive tj. u transformaciji stanja igre tako da ga agent uvek razmatra iz perspektive jednog istog igrača. Zahvaljujući tome, i funkcija q biće definisana samo iz perspektive tog igrača. Primera radi, u šahu nije ključno da li su figure bele ili crne, već da li pripadaju igraču koji je na potezu (i u čije ime agent bira potez) ili pripadaju njegovom



Slika 13.4: Osam simetričnih stanja u igri iks-oks. Ako je poredak relevantnih simbola **X** < **O** < ␣, onda je kanonsko stanje za ovaj skup prvo prikazano stanje: (**X**, **X**, ␣, ␣, **O**, ␣, ␣, ␣, ␣).

protivniku. Ukoliko se ustanovi dogovor da bele figure uvek pripadaju agentu, a da su crne uvek protivničke, u slučaju da je na potezu crni, promena perspektive se vrši rotacijom table za 180° i promenom boja svih figura: bele postaju crne, a crne postaju bele. U slučaju igre iks-oks, agentu uvek pripada simbol **X**, a u slučaju da je na potezu **O**, promena perspektive vrši se zamenom svih simbola **X** simbolima **O**, a svih simbola **O** simbolima **X**. U fazi primene, na osnovu funkcije q mogu se dobiti potezi za oba igrača – za drugog igrača primenom promene perspektive, nalaženjem najboljeg poteza i onda primenom promene perspektive unazad. Naravno, podrazumeva se da se sve ove izmene vrše nad reprezentacijom koja je u implementaciji odabrana za predstavljanje stanja. Ako je s stanje igre, označimo sa s^1 baš stanje s, a sa s^{-1} stanje dobijeno promenom perspektive u skladu sa datom igrom. Neka promenljiva p uzima vrednosti 1 i -1 u zavisnosti od toga koji igrač je na potezu (igraču **X** odgovara vrednost 1, a igraču **O** vrednost -1). Ova operacija analogno se definiše i za akcije. I nagradu je potrebno modifikovati tako da odgovara perspektivi igrača. Kako igrač svojim potezom nikad ne može izgubiti (barem je tako u igri iks-oks), iz perspektive igrača nagrada može biti samo 0 ili 1. Ovo se jednostavno postiže promenom znaka nagrade u zavisnosti od toga koji igrač je na potezu, odnosno izračunavanjem izraza $p \cdot r$. Konačno, treba imati u vidu da ono što je dobro za jednog igrača, loše je za drugog, i obratno. Stoga, pri oceni nove vrednosti izabrane akcije u polaznom stanju, maksimalnu vrednost akcije u novom stanju ne treba dodavati već je treba oduzimati od tekuće nagrade. Na osnovu navedenih razmatranja, pravilo učenja pri igranju protiv samog sebe formulišemo na sledeći način:

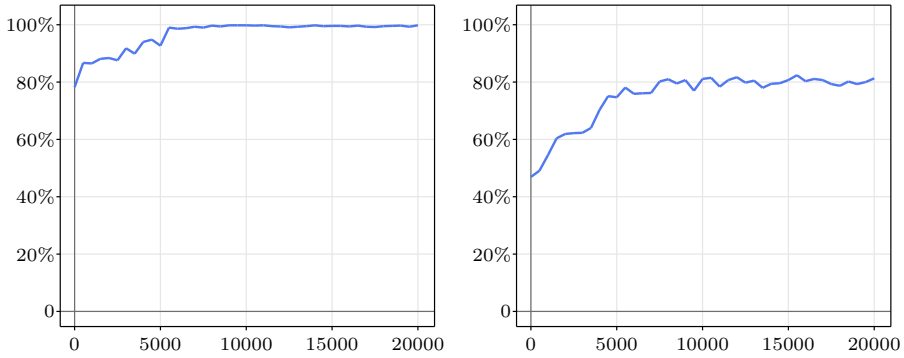
$$q(s^p, a^p) \leftarrow (1 - \alpha_t) q(s^p, a^p) + \alpha_t (p \cdot r - \gamma \max_{a'} q(s'^{-p}, a'^{-p})).$$

Primetimo da se u ovom pravilu funkcija q zaista uvek izračunava za stanja i akcije iz perspektive jednog igrača, što je obezbeđeno promenljivom p i znakovima ispred nje.

Bitno svojstvo ovog algoritma je da ukoliko se dva igrača mogu naći u analognim stanjima (tj. u istom stanju, gledano iz svoje perspektive), iskustva oba igrača mogu doprineti ažuriranju istog para stanja i akcije. Ilustracije radi, ukoliko u šahu beli nauči da matira crnog kralja pomoću dva topa, crni može matirati belog kralja pomoću dva topa na isti način, bez obzira na to što je konkretna strategija naučena iz iskustva belog. Ipak, u nekim igrama, poput igre iks-oks, igrači se nikad ne mogu naći u istim konfiguracijama table (pa ovako unapređeno učenje ne dolazi do izražaja). Naime, tabla uvek ima paran broj upisanih simbola kada je na potezu igrač **X**, a neparan broj upisanih simbola kada je na potezu igrač **O**.

Igra iks-oks ima relativno veliki broj stanja. Ipak, mnoga od tih stanja su simetrična i informacija o vrednosti jednog od njih, odnosi se i na simetrična stanja. Simetrije su određene rotacijom table za 90° i jednom refleksijom table (tj. diedarskom grupom od 8 elemenata). Za bilo koje polazno stanje, svako njemu simetrično stanje može se dobiti primenom najviše četiri navedene transformacije. Primetimo da neka od simetričnih stanja mogu biti jednaka, pa stoga različitih simetričnih stanja može biti i manje od 8. Na primer, rotacijama i refleksijama se od prazne table dobija ista tabla. Svaki skup simetričnih stanja može se predstaviti jednim kanonskim stanjem. Ovo nije važno samo zbog smanjenja tabele kojom se predstavlja funkcija q, već i zbog veće efikasnosti učenja. Naime, ukoliko se sva međusobno simetrična stanja poistovete, znanje naučeno o jednom od tih stanja, važi i za sva njemu simetrična stanja, iako ona u toku učenja možda nisu ni posećena. Sva stanja mogu se predstaviti niskom od 9 karaktera (**X**, **O** i ␣ za prazno polje) koji predstavljaju sadržaj polja table. U implementaciji se svaki skup simetričnih stanja predstavlja onim među njima koje ima leksikografski najmanju tekstualnu reprezentaciju (slika 13.4).

Implementacija opisanog algoritma koristi brzinu učenja $\alpha_t = 0.01$. Vrednost ε_t jednaka je $1/(\lfloor \frac{t}{5000} \rfloor + 1)$. Vrednost γ jednaka je 1. Broj iteracija jednak je $N = 140000$, tj. oko 20000 odigranih partija. Kvalitet tekuće politike ustanovljava se usputnom evaluacijom, kao u primeru 13.15. U toj evaluaciji, za izračunavanje procenta dobijenih partija, korišćeno je 1000 partija protiv igrača koji nasumično bira poteze. Na slici 13.5 prikazan je grafik procenta dobijenih partija u odnosu na broj odigranih partija iz kojih je učeno, kada agent igra kao igrač **X** (levo) i kada agent igra kao igrač **O** (desno). Već učeći iz prvih 6000 odigranih partija, agent uspeva da pobedi u skoro svim partijama odigranim u evaluaciji kada igra kao igrač **X**. Ne uspeva u svim, jer protivnik i



Slika 13.5: Procenat dobijenih partija u evaluaciji za igrača **X** (levo) i za igrača **O** (desno) u odnosu na broj odigranih partija iz kojih agent uči.

Slika 13.6: U igri Space Invaders igrač upravlja topom pri dnu slike koji može da se kreće levo-desno i da puca na vanzemaljce koji se spuštaju na Zemlju.

nasumičnom igrom nekada može postići nerešen rezultat. Poređenja radi, igrač koji igra nasumice kao **X**, protiv igrača koji igra nasumice kao **O**, pobeđuje u oko 30% partija. Uspešnost agenta kada igra kao igrač **O** je niža, što je u skladu sa činjenicom da u igri iks-oks igrač **X** ima prednost.

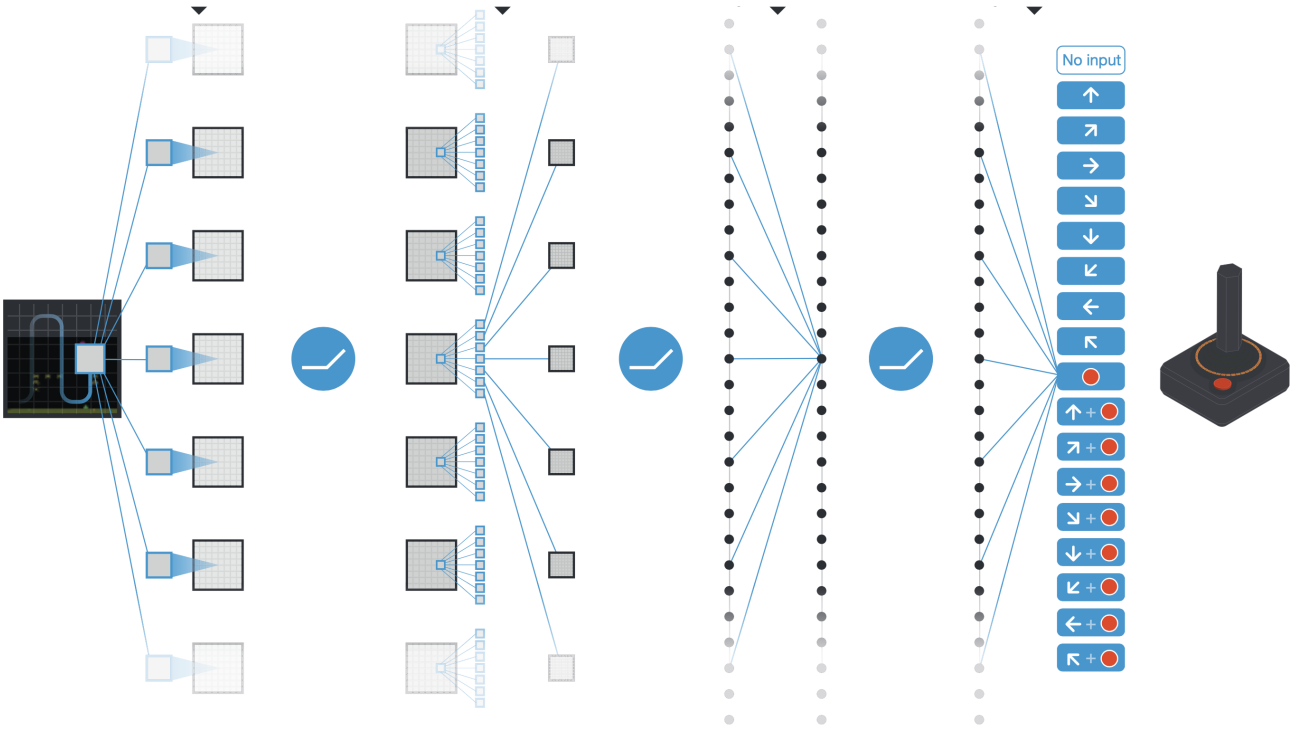
13.4.2 Igranje igara sa Atarija

Iako nije zaista praktična, jedna od poznatijih primena učenja potkrepljivanjem je igranje igara sa Atarija 2600,[2] poput prethodno pomenute igre Pong, Space Invaders i drugih. Jedna situacija iz igre Space Invaders u kojoj je cilj uništiti što više neprijateljskih svemirskih brodova prikazana je na slici 13.6.

Za svaku igru, treniran je zaseban agent zasnovan na konvolutivnoj mreži. Potrebno je da agent igra igru samo na osnovu slika ekrana, isto kao i čovek. Model funkcije vrednosti akcije je konvolutivna mreža čiji je ulaz reprezentacija stanja dobijena na osnovu slika, poput stanja diskutovanog za igru Pong iz primera 13.2. Pored ulaza, algoritam učenja dobija i nagrade na kraju partije. U slučaju Ponga, nagrada je 1 kad god je partija dobijena, a -1 kad je izgubljena. Dakle, ciljna funkcija nije poznata, već su poznate nagrade koje slede nakon nekog niza akcija, te je učenje potkrepljivanjem pogodan okvir za rešavanje ovog problema.

Konvolutivna mreža koja se koristi u svim igrama modeluje funkciju vrednosti stanja i akcije. Ilustrovana je na slici 13.7. Umesto dva ulaza – stanja i akcije, ima samo jedan ulaz – sliku koja predstavlja stanje, dok za svaku akciju ima po jedan izlaz koji daje vrednost te akcije u datom stanju. Ovakva odluka doneta je kako se pri

[2] Igračka konzola iz 1977.



Slika 13.7: Arhitektura mreže za igranje igara sa Atarija.

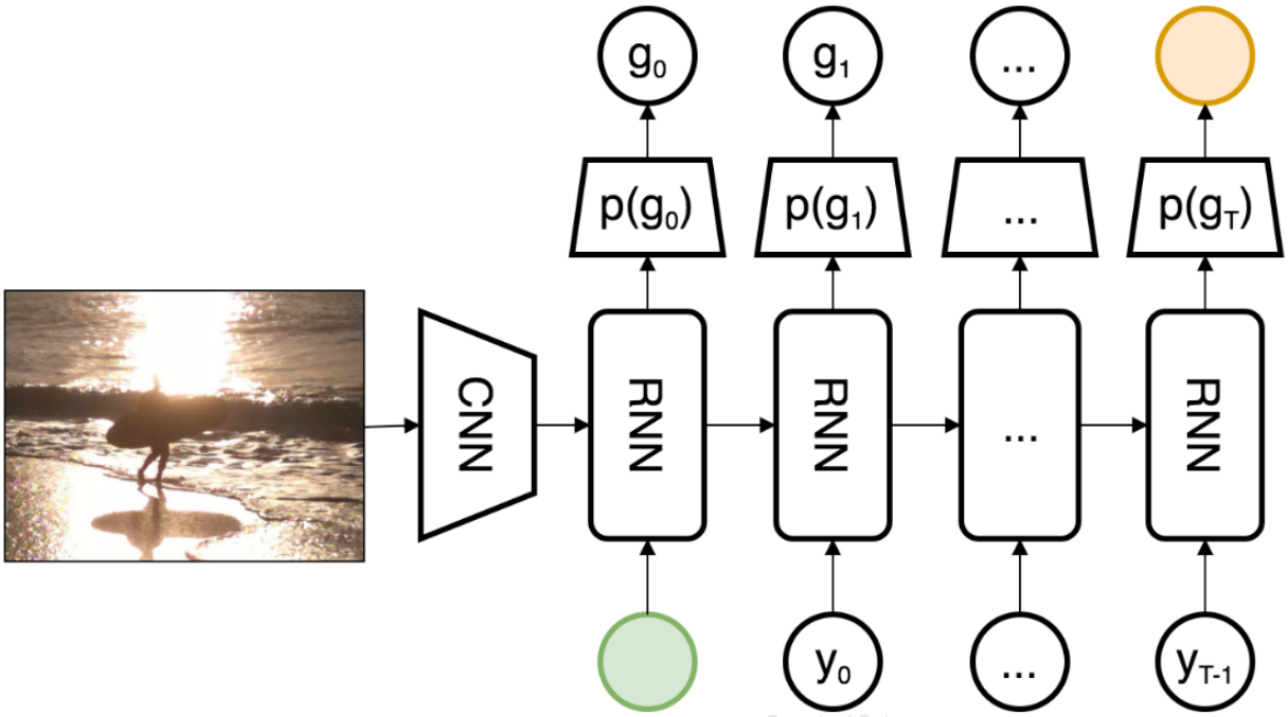
izboru poteza u datom stanju ne bi za svaku akciju iz početka vršilo izračunavanje u mreži, već se izračunavanje vrši samo jednom i odjednom se dobijaju vrednosti svih akcija i preduzima se najbolja.

Algoritam korišćen za treniranje agenta zasnovan je na pravilu (13.2, strana 232) dubokog q-učenja. Sadrži i nekoliko poboljšanja čiji je cilj da stabilizuju proces učenja, što je potvrđeno empirijski. Prvo, trojke stanja akcija i nagrada čuvaju se u jednom baferu unapred definisane veličine kako bi se iz njih moglo ponovo učiti bez potrebe da se ponovo dogode u igri. Mreža se u svakom koraku učenja trenira na osnovu uzorka trojki izabranog iz tog bafera, a ne na osnovu poslednje trojke. Drugo, vrednosti $q_{\mathbf{w}}(s,a)$ i $\nabla_{\mathbf{w}} q_{\mathbf{w}}(s,a)$ izračunavaju se na osnovu tekuće verzije mreže, dok se vrednost $r + \max_{a'} q_{\mathbf{w}}(s',a')$ izračunava na osnovu takozvane „zamrznute“ verzije te mreže čiji se parametri ne ažuriraju u svakom koraku učenja, već povremeno, na primer – na nekoliko hiljada gradijentnih koraka, jednostavno preuzimajući parametre iz tekuće mreže.

Ovaj algoritam postiže rezultate u nivou profesionalnih ljudskih igrača na 49 različitih igara sa Atarija.

13.4.3 Opisivanje slika tekstom

U glavi o nadgledanom učenju (glava 11), pomenut je jedan pristup opisivanju slika koji se zasniva na enkoder-dekoder arhitekturi, nalik prevođenju teksta sa jednog na drugi jezik. Pri tome, slika se kodira pomoću konvolutivne mreže, a potom se generisanje teksta, tj. dekodiranje vrši rekurentnom mrežom, kao što je ilustrovano na slici 13.8. Pretpostavlja se da je za svaku sliku dato nekoliko tačnih rečenica, imajući u vidu da različiti ljudi ne bi istovetno opisali sliku. Kako je izbor reči u svakom koraku generisanja klasifikacioni problem, osnovni pristup podrazumevao bi korišćenje unakrsne entropije kao funkcije greške prilikom treniranja mreže. Međutim, primećeno je da rečenice koje se dobijaju često ne liče dovoljno na rečenice kakve bi formulisali ljudi, što motiviše korićenje drugačijih ciljnih funkcija. Postoji mnoštvo metrika koje se u evaluaciji koriste za poređenje pravih tekstova sa mašinski generisanim tekstovima. Jedna takva mera sličnosti je BLEU. Neka je data jedna generisana rečenica od N reči i nekoliko referentnih rečenica koje se smatraju ispravnim. Tada je za svaku reč generisane rečenice moguće izračunati maksimalan broj m njenih pojavljivanja među referentnim rečenicama i broj njenih pojavljivanja n u generisanoj rečenici. Tada se toj reči može pridružiti ocena $\min(m,n)/N$, a ovakve ocene mogu se uprosečiti po svim generisanim rečima, što predstavlja BLEU metriku. Pored ove, postoji mnoštvo drugih metrika koje odražavaju različita svojstva generisanog teksta, a najnaprednije počivaju na definisanju grafova koji odražavaju semantiku rečenica i čijim uparivanjem se računa poklapanje rečenica po smislu. Ovakve metrike često se linearno kombinuju kako bi ukupna metrika uzela u obzir sve aspekte koje pojedinačne metrike mere. Dodatno, istraživanja sa ljudskim ispitanicima su pokazala da neke od ovih metrika dobro koreliraju sa ljudskim sudom o dobrom prevodu. Zbog toga, bilo bi poželjno koristiti ovakve metrike za definisanje funkcije greške u procesu treniranja. Međutim, one nisu diferencijabilne i čak su deo po deo konstantne u odnosu na parametre



Slika 13.8: Gruba ilustracija arhitekture enkoder-dekoder mreže za opisivanje slika.

modela, što predstavlja veliki problem za gradijentne metode. Jedna od ideja kako bi se učenje zasnovalo na optimizaciji ovakvih metrika počiva na učenju potkrepljivanjem.

Primetimo da ovaj problem predstavlja problem nadgledanog učenja imajući u vidu da su za slike dati tačni opisi. Nameće se pitanje zašto bi se učenje potkrepljivanjem koristilo u ovakvom kontekstu. U uobičajenom kontekstu nadgledanog učenja, očekuje se da funkcija greške ima upotrebljiv gradijent, što ovde, kao što je već konstatovano, nije slučaj. S druge strane, u učenju potkrepljivanjem, informacija o kvalitetu modela može se dati nagradom, za koju nema zahteva da bude diferencijabilna u odnosu na parametre modela. Dakle, ključni razlog za korišćenje učenja potkrepljivanjem je tehnički, a ne suštinski, ali ništa manje važan ukoliko je potrebno optimizovati pomenute metrike.

Problem opisivanja slike tekstom formuliše se kao problem učenja potkrepljivanjem na sledeći način. Agent se modeluje rekurentnom neuronskom mrežom poput one na slici 13.8. Stanje agenta predstavlja se skrivenim stanjem mreže koje se na početku inicijalizuje na izlaz konvolutivnog enkodera. Akcije su reči. Epizoda se sastoji u generisanju rečenice kao niza uzastopno preduzetih akcija – pojedinačnih reči. Nagrada je 0 u svakom koraku, osim u poslednjem, kada je nagrada jednaka vrednosti metrike koja se optimizuje za tako generisanu rečenicu i skup referentnih rečenica. Algoritam koji se koristi u treningu zasnovan je na algoritmu Reinforce, ali koristi niz relativno naprednih trikova koji stabilizuju i ubrzavaju proces optimizacije.

Ovaj primer pruža još važniji nauk od toga kako opisivati slike pomoću mašinskog učenja. On ukazuje da je generalno moguće optimizovati nediferencijabilne funkcije greške ako se problem nadgledanog učenja formuliše kao problem učenja potkrepljivanjem. Pritom se plaća cena sporijeg i nestabilnijeg treninga, kao i kompleksnijeg dizajna algoritma kojim se nastoje ublažiti ti problemi.

13.4.4 Automatski dizajn novih lekova

Automatski dizajn novih jedinjenja predstavlja važan problem farmaceutske industrije i aktivnu oblast istraživanja više od petnaest godina. Raniji pristupi obično su se zasnivali na optimizacionim metodama primenjenim na neku funkciju koja oslikava željena svojstva molekula. Ovaj problem je vrlo težak jer se procenjuje da je broj molekula koji se mogu sintetizovati, a koji predstavljaju potencijalne lekove između 10^{30} i 10^{60}. Ovde će biti prikazan jedan pristup, nazvan ReLeaSE (eng. *reinforcement learning for structural evolution*) zasnovan na učenju potkrepljivanjem.

Arhitektura podrazumeva postojanje dve mreže – jedne generativne, koja predlaže nove hemijski moguće molekule i igra ulogu agenta, odnosno politike i druge prediktivne, koja predviđa različita svojstva od interesa predloženog molekula (poput tačke topljenja, hidrofobnosti, broja benzenovih prstena, broja hemijskih grupa poput -OH, -NH_2, itd) i koja predstavlja pomoćni element u svrhe definisanja nagrade. Obe mreže prvo se

Slika 13.9: Lebdenje helikoptera u mestu u obrnutom položaju.

treniraju nadgledanim učenjem pomoću dostupnih obeleženih podataka (koje čini preko milion jedinjenja), a potom se generativna mreža trenira učenjem potkrepljivanjem.

Molekuli su visoko strukturirani objekti kojima je teže baratati nego slikama i vektorima. Otud se prvo postavlja pitanje njihove reprezentacije u kontekstu neuronskih mreža. Na sreću, postoje tekstualna kodiranja grafova i specifično, hemijskih struktura. Jedna takva reprezentacija naziva se SMILES. U njoj se, na primer, molekul aspirina zapisuje kao [CC(=O)OC1=CC=CC=C1C(=O)O], ali nećemo ulaziti u detalje te sintakse. Onda se skup stanja agenta može predstaviti kao skup validnih SMILES niski do neke unapred predviđene dužine. Akcije odgovaraju slovima SMILES azbuke. Nagrade su 0 u svim nezavršnim stanjima, a u završnom stanju s se definišu kao

$$r(s) = f(P(s))$$

gde je P prediktivna mreža, a f ručno dizajnirana funkcija čiji izbor zavisi od željenih svojstava jedinjenja. Na primer, ako je potrebno maksimizovati tačku topljenja i minimizovati hidrofobnost jedinjenja, funkcija f bi bila razlika predviđene tačke topljenja i predviđene hidrofobnosti, pri čemu su te procene dobijene od strane mreže P. Učenje se sprovodi algoritmom Reinforce, pri čemu treba imati u vidu da se prediktivna mreža ne trenira nakon inicijalnog treninga nadgledanim učenjem.

Sintaksa SMILES niski, odnosno struktura molekula koju predstavljaju, nije jednostavna i obične rekurentne mreže nisu se pokazale dovoljno dobrim u njihovom generisanju, pa generativna mreža uključuje i određene netrivijalne nadogradnje.

Veliki problem ranijih mehanizama za dizajn novih molekula je to što često generišu hemijski neispravne kandidate. Ispostavlja se da generativna mreža trenirana na opisani način generiše validne molekule u 95% slučajeva, od čega manje od 0.1% od milion generisanih struktura predstavljaju molekule iz trening skupa, tj. mreža nije samo naučila trening podatke napamet. Štaviše, prema standardnoj metodologiji procene težine hemijske sinteze predloženih molekula, mreža skoro uvek predlaže molekule koje je lako sintetizovati.

13.4.5 Autonomno upravljanje helikopterom

Autonomno upravljanje helikopterima smatra se težim problemom od upravljanja vazduhoplovima sa fiksiranim krilima. Upravljanje helikopterom pri malim brzinama posebno je teško. Dodatno, pri malim brzinama, jedan od najtežih manevara je lebdenje u mestu u obrnutom položaju — sa elisom naniže, kao što prikazuje slika 13.9. U ovom položaju poseban izazov je stabilnost, pošto je centar mase visoko. Ovaj manevar nije od velikog praktičnog značaja, ali je zanimljiv kao ozbiljan izazov za automatizaciju upravljanja, pa zato diskutujemo kako je rešen učenjem potkrepljivanjem. Navedeni pristup može se upotrebiti za učenje lebdenja i u drugim položajima, pretpostavljajući da su fizički održivi. U eksperimentima je korišćen helikopter na daljinsko upravljanje dužine 150cm, visine 56cm i težine oko 8kg i motorom od 46 kubika.

Opišimo prvo način upravljanja helikopterom. Većinom helikoptera se upravlja pomoću četiri kontrole:

- uzdužni nagib helikoptera, čime se helikopter naginje napred ili nazad, čime se kontroliše ubrzanje u smeru duž helikoptera;
- nagib helikoptera u stranu, čime se helikopter naginje levo ili desno, čime se kontroliše ubrzanje u stranu;
- nagib oštrica glavne elise, čime se kontroliše potisak glavne elise;
- nagib oštrica pomoćne elise, čime se kontroliše rotacija helikoptera oko uspravne ose.

Bitan problem u učenju upravljanja helikopterom je taj što se ne može eksperimentisati sa nasumično izabranim akcijama, kao što učenje potkrepljivanjem pretpostavlja, pošto su helikopteri skupi, a takve akcije mogu voditi njihovom padu. Zato je prvi korak izgradnja verodostojnog simulatora u kojem će potom biti vršeno treniranje. Simulator je modelovan nadgledanim učenjem na osnovu 391 sekunde ljudski kontrolisanog leta u toku kojeg je helikopter leteo u obrnutom položaju. Pritom su pamćene informacije o letu 10 puta u sekundi. Svaki od ovih zapisa sadrži informaciju o stanju helikoptera i preduzetoj akciji pilota. Na osnovu ovih podataka, treniran je model nadgledanog učenja kojim se za dato stanje i akciju predviđa buduće stanje. Simulator na to predviđanje dodaje određeni šum, kako bi modelovao stohastičnost okruženja. Važno pitanje je, naravno, šta su akcije i stanja. Akcije su četiri nabrojane kontrole numerički predstavljene realnim brojevima iz intervala $[-1, 1]$. Reprezentacija stanja je nešto komplikovanija. Ona treba da izrazi poziciju, orijentaciju i brzinu helikoptera. Ako bi se koordinatni sistem vezao za neku fiksiranu tačku spoljnog sveta i imao fiksiranu orijentaciju, modelu simulatora bilo bi teško da nauči invarijantnost zakona fizike u odnosu na translacije i rotacije. Naime, ako je helikopter u jednoj tački prostora za datu akciju imao određenu promenu stanja, model bi usled malog uzorka mogao naučiti da je za tu promenu stanja bila bitna i lokacija na kojoj se helikopter nalazio, iako nije. Otud se spoljni koordinatni sistem koristi za definisanje uglova rotacije helikoptera oko tri ose (eng. *roll*, *pitch*, *yaw*) i ugaone brzine rotacije oko tih osa, ali se za lokaciju i ubrzanje koristi referentni sistem vezan za sam helikopter. Kako je u njemu lokacija konstantna, ona se ni ne čuva eksplicitno, već samo brzina duž osa tog koordinatnog sistema. Stanje se onda definiše vektorom ubrzanja, uglova rotacije i ugaonih brzina. Prilikom modelovanja simulatora, model ne mora da predviđa uglove rotacije, pošto se oni uvek mogu dobiti integracijom na osnovu ugaonih brzina.

Kada je nadgledanim učenjem definisan simulator, koristi se učenje potkrepljivanjem kako bi se trenirao helikopter da leti u nekom (recimo obrnutom) položaju u tom simulatoru. Akcije su očito kontinualne, pa je pogodno koristiti algoritam Reinforce, a politiku predstaviti normalnom raspodelom u odnosu na izlaze mreže koja predviđa vrednosti za 4 akcije. Nagrada se može definisati na sledeći način:

$$-(\alpha_{\mathbf{x}}\|\mathbf{x}-\mathbf{x}^*\|_2^2+\alpha_{\dot{\mathbf{x}}}\|\dot{\mathbf{x}}\|_2^2+\alpha_{\phi}\|\phi-\phi^*\|_2^2+\alpha_{\dot{\phi}}\|\dot{\phi}\|_2^2)$$

pri čemu vektori $\mathbf{x}$ i ϕ označavaju vektore pozicije i uglova rotacije helikoptera, zvezdica označava željene vrednosti, a tačka brzinu, odnosno izvod u odnosu na vreme. Koeficijenti α su empirijski izabrani tako da svi izrazi budu istog reda veličine. Ova nagrada je utoliko veća ukoliko je helikopter bliže željenoj tački i orijentaciji i što je mirniji.

Spisak preuzetih slika

Slika 11.5 napravljena je po uzoru na sliku iz kolekcije [23].

Slika 11.6 napravljena je po uzoru na sliku iz kolekcije [23].

Slika 11.11 preuzeta je sa Vikipedije.

Slika 11.15 preuzeta je uz modifikacije iz izvornog rada [9].

Slika 11.16 preuzeta je iz izvornog rada [17].

Slika 11.17 preuzeta je iz izvornog rada [17].

Slika 11.19 preuzeta je iz izvornog rada [4].

Slika 11.20 preuzeta je iz izvornog rada [4].

Slika 11.21 preuzeta je iz izvornog rada [22].

Slika 11.22 preuzeta je iz izvornog rada [21].

Slika 13.6 preuzeta je iz kolekcije okruženja OpenAI Gym.

Slika 13.7 preuzeta je iz izvornog rada [11].

Slika 13.8 preuzeta je iz izvornog rada [12].

Slika 13.9 preuzeta je iz izvornog rada [13].

Literatura

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR 2015*, 2015.

[2] A. Biere, M. Heule, H. Van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[3] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. Openpose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[5] D. Gabbay, C. J. Hogger, and A. Robinson. *Handbook of Logic in Artificial Intelligence and Logic Programming*. Clarendon Press, 1998.

[6] B. Goertzel, N. Geisweiller, L. Coelho, P. Janičić, and C. Pennachin. *Real-World Reasoning: Toward Scalable, Uncertain Spatiotemporal, Contextual and Causal Inference*. Atlantis Thinking Machines, 2011.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[8] P. Janičić. *Matematička logika u računarstvu*. Matematički fakultet, 2009.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[10] G. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson, 2009.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglu, H. King, D. Kumaran, D. Wiestra, S. Legg, and D. Hassabis. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 2015.

[12] K. Murphy, N. Ye, S. Guadarrama, S. Liu, and Z. Zhu. Improved Image Captioning via Policy Gradient Optimization of SPIDEr. *ICCV 2017*, 2017.

[13] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. *Experimental Robotics IX*, 2006.

[14] M. Nikolić and A. Zečević. *Mašinsko učenje*. U pripremi, 2021.

[15] F. Petroni. Language Distance and Tree Reconstruction. *Journal of Statistical Mechanics Theory and Experiment*, 2008.

[16] M. Popova, O. Isayev, and A. Tropsha. Deep Reinforcement Learning for de Novo Drug Design. *Science Advances*, 2018.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *CVPR 2016*, 2016.

[18] A. Robinson and A. Voronkov. *Handbook of Automated Reasoning*. Elsevier, 2001.

[19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2020.

[20] A. Shi. Visualize How a Neural Network Works from Scratch. Towrads Data Science, https://towardsdatascience.com/visualize-how-a-neural-network-works-from-scratch-3c04918a278, 2020.

[21] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *CVPR 2011*, 2011.

[22] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. *NIPS 2014*, 2014.

[23] P. Veličković. Collection of PGF/TikZ figures. https://github.com/PetarV-/TikZ, 2018.