Рачунарска интелигенција

Оптимизација

Александар Картељ

kartelj@matf.bg.ac.rs

Датум последње измене: 23.10.2019.

Оптимизација

Основни појмови

Оптимизација

- Оптимизациони алгоритми припадају групи **алгоритама претраге**
- Циљ је пронаћи решење проблема такво да је:
  1. Нека циљна функција (**функција циља**) максимална/минимална
  2. Неки **скуп ограничења** задовољен (опционо)
- Изазови:
  - Решење може бити представљено као комбинација вредности из различитих домена
  - Ограничења могу бити нелинеарна
  - Карактеристике проблема могу варирати током времена
  - Функција циља може бити у „конфликту“ са ограничењима

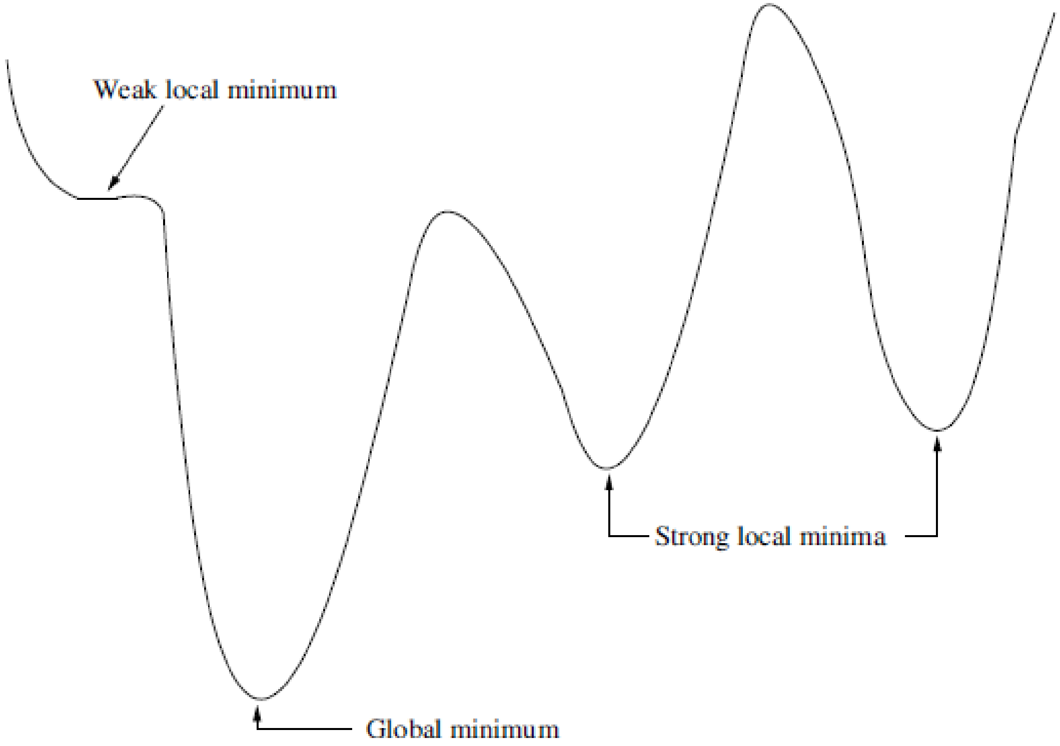
Кључни појмови у оптимизацији

- Нека је са $f: S \rightarrow R$, означена **функција циља**
- S је домен, док је кодомен скуп реалних бројева
- Приметити да је минимум f исто што и максимум од $-f$
- Скуп $\boldsymbol{x}$ представља **независне променљиве** које утичу на вредност f и за дате вредности променљивих функција има вредност $f(\boldsymbol{x})$
- Скуп ограничења најчешће поставља зависности између независних променљивих
- Такође омогућава и ограничавање самих независних променљивих, нпр. на неки интервал или скуп вредности

Програмирање ограничења

- Постоје област која се зове **прорамирање ограничења** (енг. Constraint programming)
- Она се бави проблемима без функције циља
- Овде постоје само ограничења која треба задовољити
- Оптимизација се не бави оваквим проблемима!
- Иако се неким оптимизационим методама могу решавати и овакви проблеми

Типови оптимума



- **Глобални оптимум** – најбоље решење на читавом допустивом скупу решења S

$$f(\mathbf{x}^*) < f(\mathbf{x}),\ \forall \mathbf{x} \in S$$

- Јак локални оптимум - најбоље решење у некој околини $N \subseteq S$

$$f(\mathbf{x}^*_N) < f(\mathbf{x}),\ \forall \mathbf{x} \in N$$

- Слаб локални оптимум – једно од најбољих решења у околини $N \subseteq S$

$$f(\mathbf{x}^*_N) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in N$$

Методе оптимизације

- Оптимизационе методе траже оптимум у простору допустивих решења
- Решење је допустиво ако испуњава ограничења проблема
- Подела метода према фокусу претраге:
    - Локалне
    - Глобалне
- Подела метода према приступу претраге:
    - Стохастичке
    - Детерминистичке

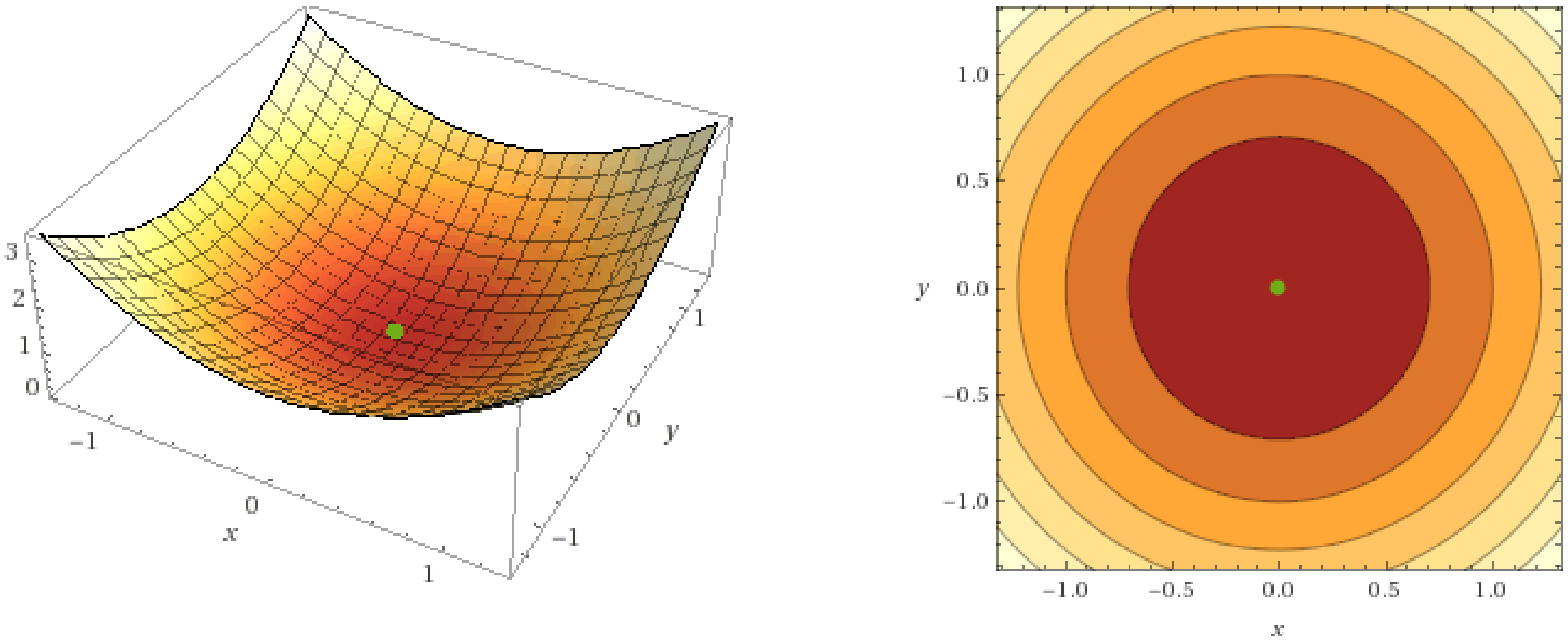
Оптимизација без ограничења

- Формулација проблема је следећа:

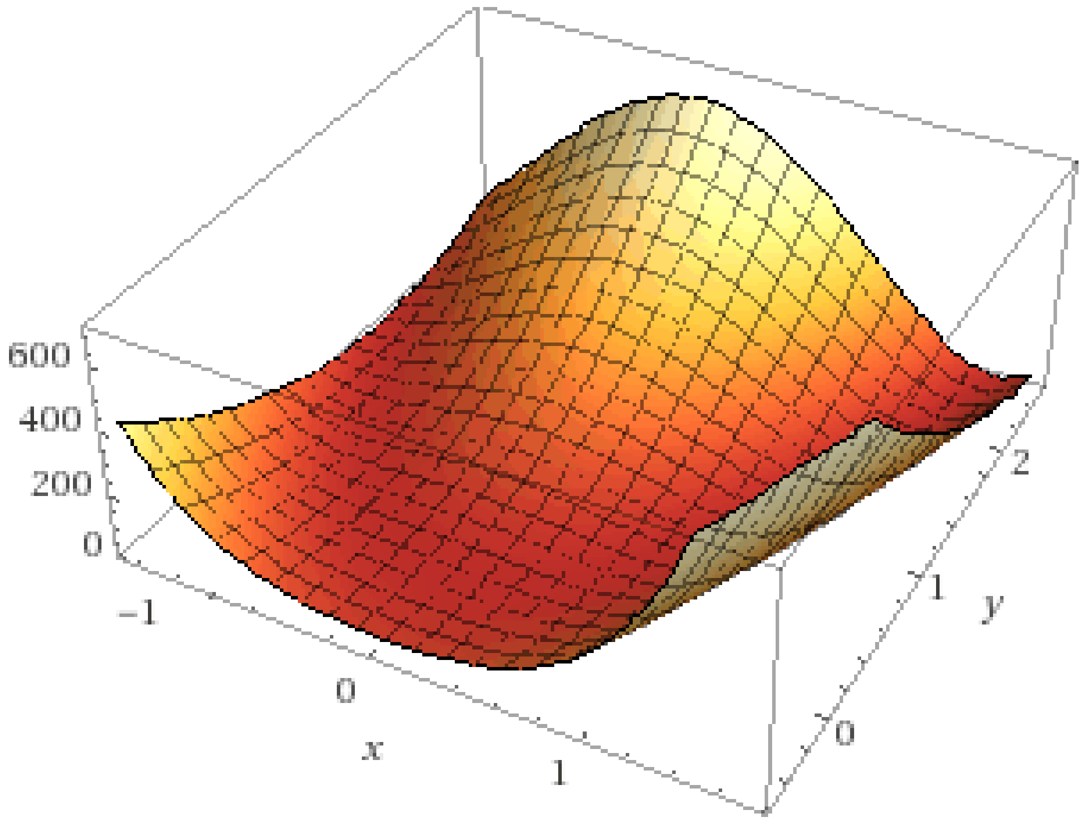
  минимизовати $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_{nx})$, $\boldsymbol{x} \in S$

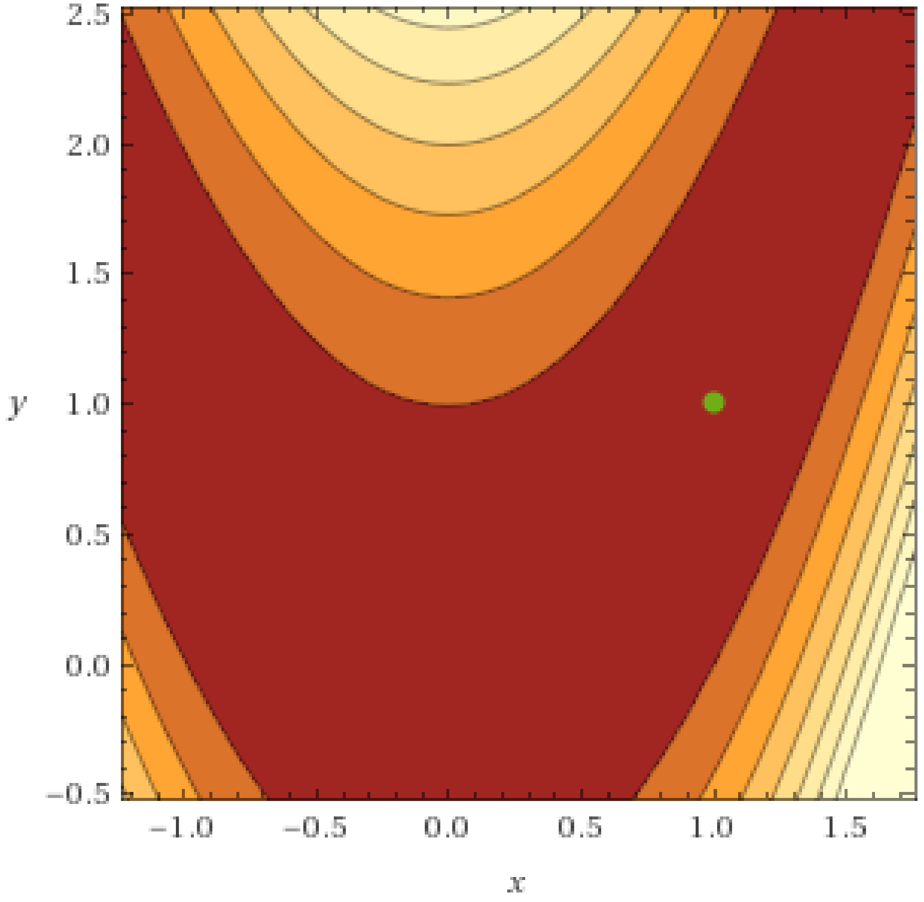
- Пример:
  пронаћи минимум функције $f(x,y)=x^2+y^2$ на домену реалних бројева

Оптимизација без ограничења (2)



Оптимизација без ограничења (3)





Rosenbrock функција – $f(x,y)=(1-x)^2+100(y-x^2)^2$

Оптимизација са ограничењима

- Општа форма у случају ограничења:

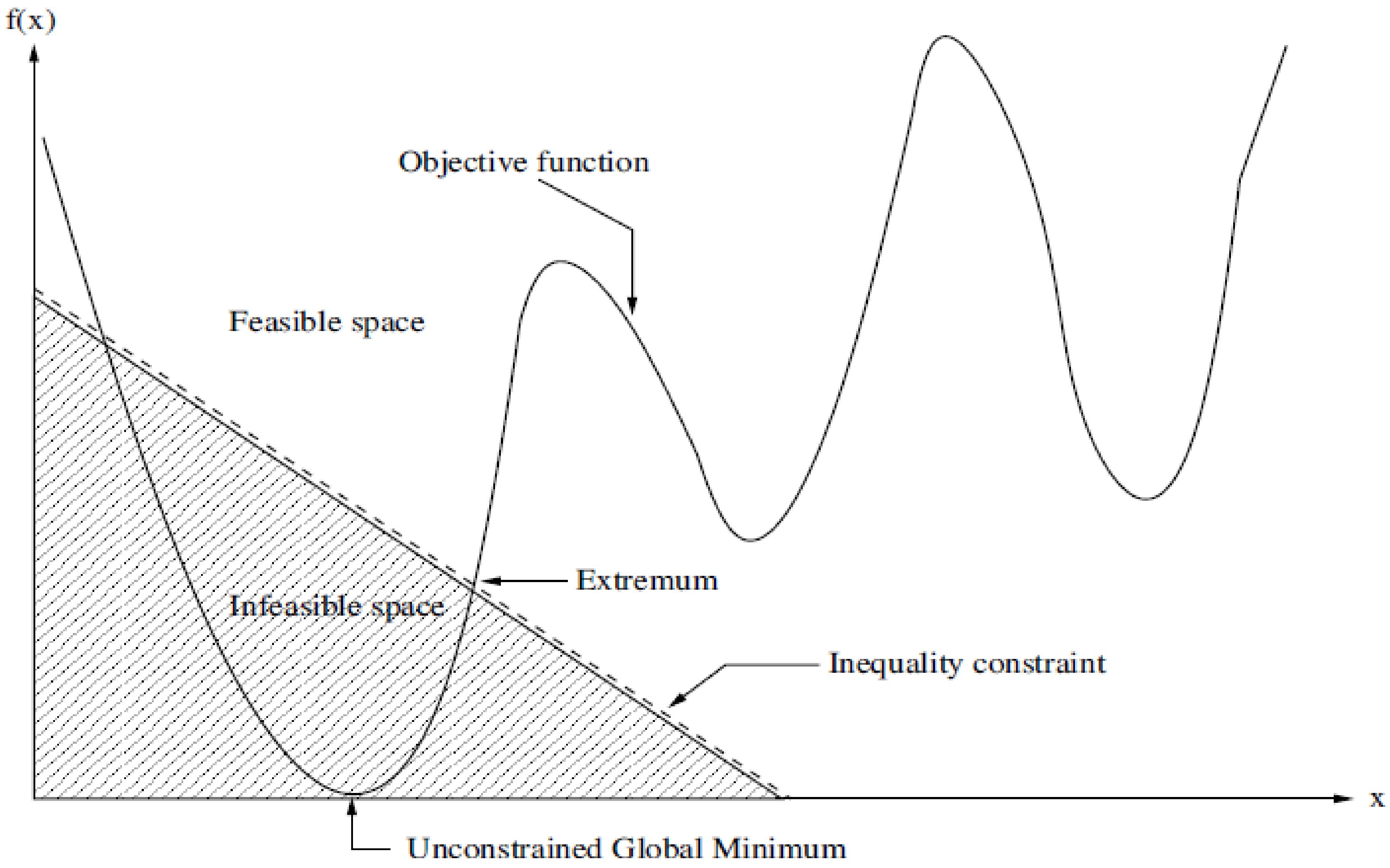
минимизовати $f(\mathbf{x})$, $\mathbf{x} = (x_1, \dots, x_{nx})$, $\boldsymbol{x} \in S$
при ограничењима:

$g_m(\mathbf{x}) \leq 0,\ m = 1, \dots, n_g$

$h_m(\mathbf{x}) = 0,\ m = n_g + 1, \dots, n_g + n_h$

- Видимо да поред доменских ограничења овде постоје и ограничења засновала на једнакости и/или неједнакости
- Форма функција g_m и h_m је у општем случају нелинеарна

Оптимизација са ограничењима (2)



Оптимизација са ограничењима (3)

- Поставља се питање, шта радити са недопустивим решењима?

1. **Одбацивати** их;
2. **Додељивати пенал** – нпр. негативни фактор у случају максимизације или позитиван у случају минимизације;
3. **Сводити** на решење **без ограничења** па га после конвертовати у решење које поштује ограничења;
4. **Одржавати допустивост** самим дизајном метода;
5. **Уређивати недопустива** решења према степену недопустивости;
6. **Поправљати** недопустива решења.

Напомена

- Постоје још неки видови поставки оптимизационих проблема:

1. Проблеми са **вишеструким оптимумима**:
   циљ је пронађи сва решења која су оптимална или довољно близу оптимума
2. **Вишециљна** оптимизација:
   проблем има више функција циља
   па је сложеније уредити решења
3. **Динамичка** оптимизација:
   функција циља се мења током времена

Домен простора решења

- Према типу домена над којим се врши, оптимизација се дели на:

1. **Комбинаторну** (дискретну) оптимизацију:
   допустиви скуп вредности променљивих је из коначног
   или бесконачног скупа целих бројева
   - Специјални случај су проблеми **бинарне оптимизације**
2. **Глобалну** (континуалну) оптимизацију:
   допустиви скуп вредности из домена реалних бројева

- Методе којима ћемо се бавити у даљем току курса ће бити у стању да решавају често проблеме и једног и другог типа
- Нешто већа пажња ће бити посвећена **комбинаторној оптимизацији**

Оптимизација

Комбинаторна оптимизација

Комбинаторна оптимизација

- Формулација проблема комбинаторне оптимизације је:

$$\textit{минимизовати } f(\mathbf{x}), \mathbf{x} = (x_1, x_2, \ldots, x_{nx}), \mathbf{x} \in S$$

при чему је *S* коначан или бесконачан и дискретан

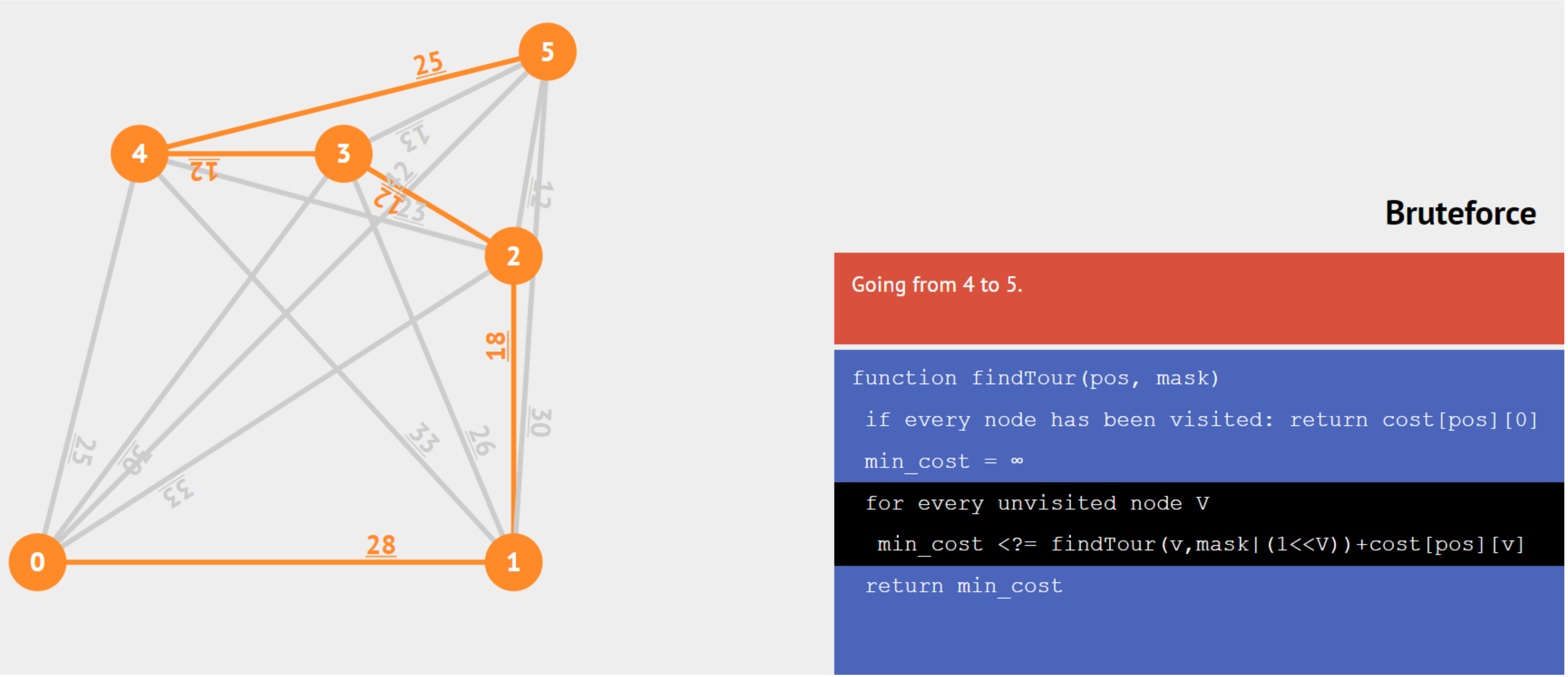
- **Пример: Трговачки путник (TSP) -** Нека је задат скуп *C* од *m* градова и функција удаљености $d(c_i, c_j) \in N$ за сваки пар градова. Пронаћи пермутацију $p: [1..m] \rightarrow [1..m]$ такву да је укупна сума удаљености грана које се пролазе обиласком минимална.

Решавање TSP

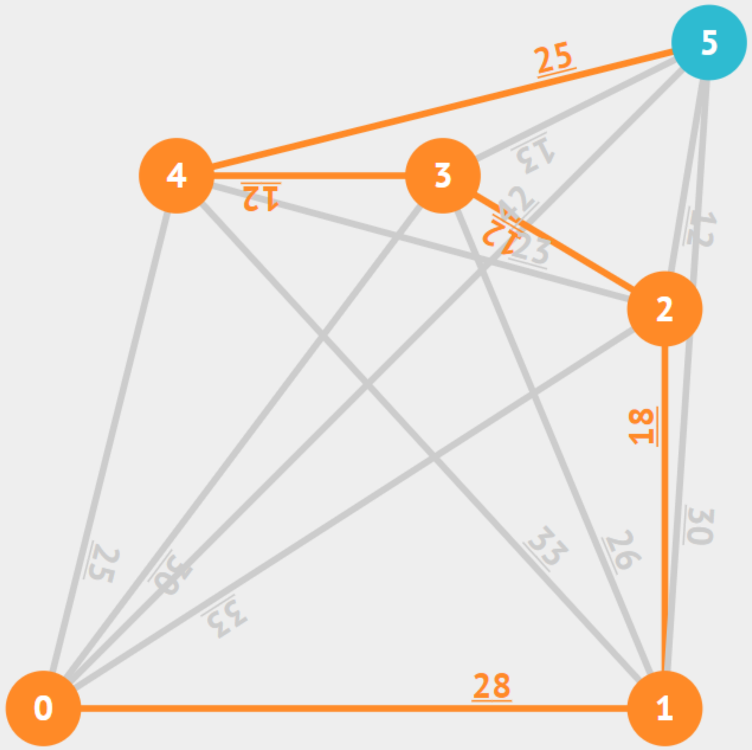
- Колика је величина допустивог скупа?
- Величина је m!
    - Помоћу Стирлингове формуле може се уочити да је ово експоненцијално
- Који алгоритам препоручујете у општем случају и шта је општи случај?
    - Општи случај би био TSP над произвољним графом
    - У општем случају, најсигурније решење је **тотална енумерација** (енг. Brute-force)
- Шта ако претпоставимо да се ради у Еуклидском простору?
    - Може се мало боље због важења неједнакости троугла, али је проблем и даље тежак

Решавање TSP (2)

https://visualgo.net/en/tsp



Решавање TSP (3)

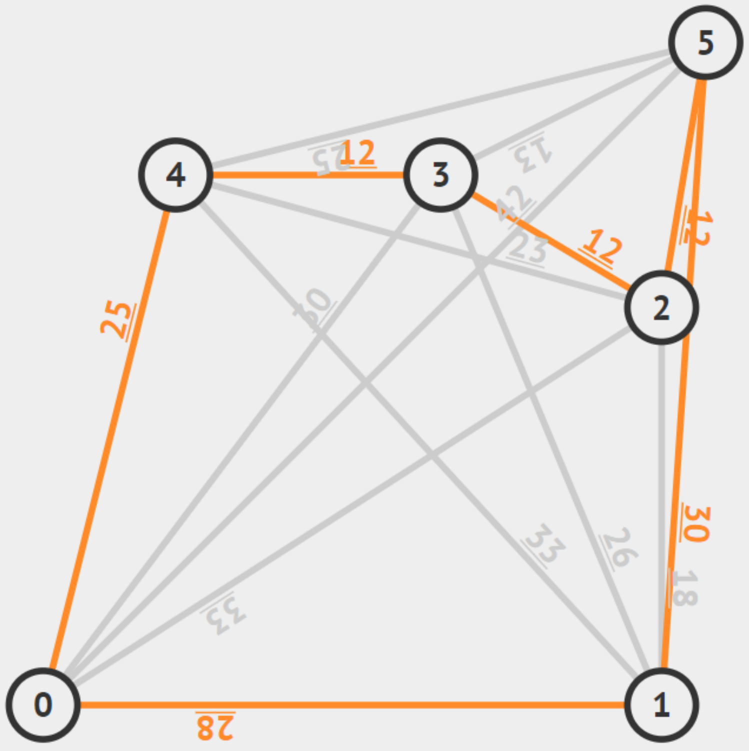


Dynamic Programming

Position = 5, bitmask = 111111.

```
function findTour(pos, mask)
  if every node has been visited: return cost[pos][0]
  if (pos, mask) in memo: return memo[(pos, mask)]
  min_cost = ∞
  for every unvisited node V
   min_cost <?= findTour(v,mask|(1<<V))+cost[pos][v]
  return memo[(pos, mask)] = min_cost
```

Решавање TSP (4)



2-approximation TSP

Found 2-approximation TSP tour [0,4,3,2,5,1] with cost = 119. While the cost of the optimal solution = 108.

```
Get the MST of the (complete) graph
Duplicate each edge in the MST and obtain a eulerian t
Obtain a TSP tour from the obtained eulerian tour and
skip repeated vertices
```

Поређење различитих приступа

- Прва два приступа
  - Су неадекватни када димензија проблема нарасте – нпр. већ за 50-ак градова
  - Међутим, предност је што дају оптимална решења ако заврше
- Апроксимативни приступ
  - Је ефикасан – полиномијалан
  - Међутим, квалитет решења може бити значајно нарушен, понекад нам добијање до два пута лошијег решења не одговара
  - Предност је што знамо да решење не може да буде више од 2 пута лошије
- Приступи којима ћемо се бавити у даљем току курса, метахеуристике, ће омогућити:
  1. Да квалитет решења буде „очекивано“ близак оптималном (не и гарантовано)
  2. Да време извршавања буде разумно – можемо да га контролишемо

Алгоритми претраге

- Метахеуристике припадају широј групи алгоритама претраге
- У наставку је дата општа форма алгоритма локалне претраге

Започни од полазног решења $\mathbf{x}(0) \in S$;

$t = 0$;

repeat

Израчунај вредност $f(\mathbf{x}(t))$;

Израчунај правац и смер претраге, $\mathbf{q}(t)$;

Израчунај дужину корака претраге $\eta(t)$;

Пређи у наредно решење $\mathbf{x}(t + 1) \rightarrow \mathbf{x}(t) + \eta(t)\mathbf{q}(t)$;

$t = t + 1$;

until *није задовољен критеријум завршетка*;

Врати $\mathbf{x}(t)$ као коначно решење;

Убацивање случајности у претрагу

- Други екстрем би била потпуно случајна претрага – како се она реализује?
- Добри алгоритми претраге који не „упадају“ у локалне оптимуме користе и **случајност** и **детерминизам** (симулирано каљење →)

Започни од полазног решења, **x**(0);
Постави почетну температуру, *T*(0);
t = 0;
repeat
  Формирај ново решење, **x**;
  Израчунај вредност, *f*(**x**);
  Израчунај вероватноћу прихватања решења;
  if *U*(0, 1) ≤ *вероватноћа прихватања* **then**
    x(*t*) = **x**;
  end
until *није задовољен критеријум завршетка*;
Врати **x**(*t*) као коначно решење;

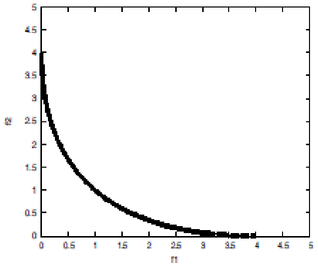
Вишециљна оптимизација

- Често у пракси ситуација да је потребно задовољити више функција циља (критеријума):
  - Економија – наћи портфолио акција са максималним приходом и минималним ризиком
  - Транспортни проблеми – максимизација искоришћености улица уз минимизацију загушења, трошкова рутирања и слично
- Дефиниција отпималности је компликованија
  - Често побољшање једне функције циља изазива погоршање друге, нпр. побољшање структуралне стабилности моста изазива погоршање (повећавање) трошкова изградње
  - Тада се говори о прављењу баланса, односно компромиса концепт **недоминираних решења**

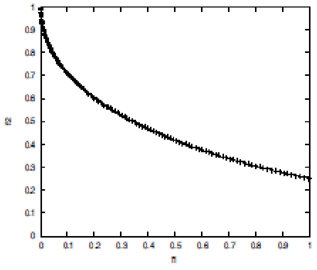
Приступи решавању

- Прављење пондерисаних просека односно агрегација
  - За сваку функцију циља се одређује тежина (значај)
  - Ово се касније своди на класичну једноциљну оптимизацију
  - Проблем: како одредити пондере/тежине?
- Прављење скупа Парето-оптималних решења
  - Решење **x** доминира над решењем **y** ако ниједна вредност функције циља од решења **y** није боља од одговарајуће вредности функције циља **x**
  - Решење **x** је Парето-оптимално ако не постоји ниједно друго решење које доминира над њим
  - Скуп свих Парето-оптималних решења се назива **Парето-оптималан скуп**
  - **Парето-оптимална површ** предстаља површ коју формирају функције циља када се примене над Парето-оптималним скупом решења

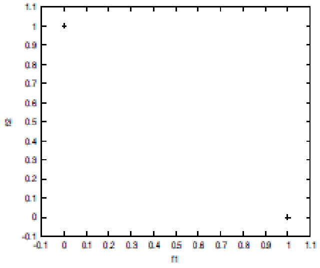
Пример Парето-оптималних површи



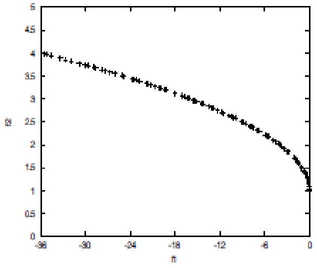
(a) Convex, Uniform Front for Equation (A.49)



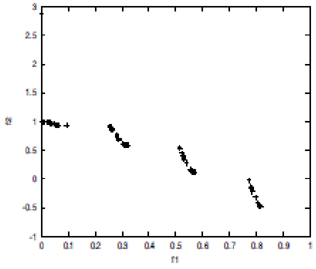
(b) Convex, Non-uniform Front for Equation (A.50)



(c) Concave Front for Equation (A.51)



(d) Partially Convex, Partially Concave Front for Equation (A.52)



(e) Discrete Convex Front for Equation (A.53)

- Приступи решавању:
  - Алгоритми претраге који могу ефикасно да претражују Парето-оптималну површ
  - Нпр. популационе стратегије које итеративно побољшавају скуп добрих решења употребом својства доминације

Питања за размишљање

- Шта би се добило применом алгоритма локалне претраге над проблемом трговачког путника?
- А шта применом случајне претраге?
- Размислите о неком Вашем алгоритму претраге који би користио и детерминизам, али и случајност?

Оптимизација

Класе сложености израчунавања

Проблеми одлучивања

- Алгоритам за решавање неког проблема је:
  - Коначан списак правила чијим праћењем долазимо до решења било којег партикуларног проблема (инстанце проблема) из задате класе
  - Праћење правила траје коначно много корака
- Проблеми одлучивања:
  - Решење се састоји у потврђивању или оповргавању неке особине
- Свођење на проблем одлучивања:
  - Различити оптимизациони проблеми попут TSP се могу свести на проблем одлучивања
  - Можемо се нпр. питати да ли за конкретан TSP проблем постоји решење са трошком мањим од C
  - Даље, можемо на основу одговара мењати границу трошка и тако итеративно

Класа полиномско решивих проблема

- Проблем одлучивања припада класи *P* (полиномско решивих)
  - ако **постоји алгоритам А** за решавање тог проблема
  - и полином *p(n)* такав да А завршава извршавање за **не више од *p(n)*** корака за сваку инстанцу тог проблема
  - при чему је ***n* димензија проблема**
- Напомена: подсетити се анализе најгорег случаја и нотације О
- Полиномијалне алгоритме сматрамо „добрим“ алгоритмима
- Алгоритме чије време не можемо да ограничимо полиномом подразумевано ограничавамо експоненцијалном функцијом c^n ($c>1$)

Експоненцијално решиви проблеми

- Постоје проблеми за које је доказано да не могу бити решени алгоритмом бржим од експоненцијалног
  - Подразумева се да су ови проблеми постављени као проблеми одлучивања (класа **EXPTIME**)
  - Нпр. проблеми евалуације потеза у уопштеном шаху, игри GO и слично
  - Уоптешна варијанта игре подразумева да је променљиве димензије

- Пример: пронаћи скуп свих разапињућих стабала у потпуном (комплетном) графу са n чворова.
  Коментар: зна се да је број разапињућих стабала n^{n-2}.
  Дакле, алгоритам утроши експоненцијално време само за приказ резултата рада, а да не говоримо о времену потребном за њихово налажење.

Недетерминистички полиномски проблеми

- Међутим, постоје проблеми за које се не зна да ли постоји полиномски алгоритам за њихово решавање
- Ако се за конкретно понуђено решење проблема одлучивања
  - може утврдити да ли је одговор потврдан у полиномијалном броју корака
  - онда је у питању **недтерминистички полиномски проблем (NP проблем)**
- На примеру TSP се може приметити да за се за:
  - било коју дату пермутацију градова (решење)
  - и број C који представља трошак
  - може дати потврдан или одричан одговор у полиномском времену

Редукција (свођење) проблема

- Нека су дата два проблема одлучивања A_1 и A_2
- Претпоставимо да се за A_1 може конструисати полиномски алгоритам у којем се као један од корака појављује алгоритам за решавање проблема A_2
- Ако је притом алгоритам за решавање проблема A_2 полиномски онда је јасно да и за A_1 постоји полиномски алгоритам
- Каже се да се A_1 **редукује** на A_2
  - ако се за сваки специјалан X проблема A_1
  - може у полиномском времену пронаћи специјалан случај Y проблема A_2
  - такав да је за проблем X одговор потврдан акко је и за Y одговор потврдан

NP потпуни (комплетни) проблеми

- Проблем је NP потпун ако за свођење било ког NP проблема на посматрани проблем постоји полиномски алгоритам
- Последица:
  ако се за било који NP потпун проблем пронађе полиномски алгоритам, тиме се доказује постојање полиномског алгоритма за сваки NP проблем! Односно показало би се да је N=PN!
- У контексту оптимизације спомињу се NP тешки проблеми
- То су проблеми чије су одлучиве варијанте NP потпуни проблеми

Како решавати проблеме?

- Два општа приступа:
  1. **Егзактно** решавање проблема:
     поступци који доводе до гарантовано оптималног решења ако заврше
  2. **Приближно** (апроксимативно) решавање:
     постпуци који чак и када заврше не гарантују оптималност
     - Овде убрајамо и (мета)хеуристике и алгоритме са гаранцијом квалитета
- Полиномске проблеме решавати егзактно!
- Приближно решавати само ако не постоји полиномски поступак, а ово значи да приближно треба решавати NP тешке проблеме!

Материјали

- На адреси испод се налази преглед NP тешких проблема: https://www.nada.kth.se/~viggo/problemlist/compendium.html
- Када нисте сигурни да ли је проблем NP тежак, погледајте да ли се налази у овом списку
- Може се десити да проблем није у списку, али да јако подсећа на неки од постојећих
  - У том случају је кандидат за додавање у списак
  - Да би се за неки нови проблем показало да је NP тежак (комплетан) довољно је:
    1. показати да је NP
    2. редуковати неки од постојећих NP комплетних проблема на њега (то значи да је нови проблем тежак бар колико и тај који је редукован на њега)

Додатна литература

- Čangalović, M., et al.
  "Kombinatorna optimizacija: matematička teorija i algoritmi." (1996).
- Garey, Michael R. "
  A Guide to the Theory of NP-Completeness."
  Computers and intractability (1979).