# Problem trgovskega potnika

## 1 Motivacija in opis problema

Problem trgovskega potnika (v nadaljevanju PTP, angl. *Traveling salesman problem (TSP)*) je v 19. stoletju prvi matematično obravnaval irski matematik W. Hamilton, danes pa je verjetno najbolj proučevan problem kombinatorične optimizacije. Imamo $n$ mest in množico povezav $i \to j$ z neko utežjo (dolžno, ceno) $d_{ij}$. Zanima nas, po kateri poti moramo iti, da bomo obiskali vsako mesto natanko enkrat in se vrnili v izhodišče. Cilj je, da bo skupna cena (razdalja) poti najmanjša (najkrajša).

Lahko predpostavimo, da imamo opravka s polnim grafom. To pomeni, da med poljubnima mestoma obstaja usmerjena povezava. V primeru, da med mestoma $i$ in $j$ ni usmerjene povezave nastavimo $d_{ij} = \infty$. Iščemo najkrajši Hamiltonov obhod tega grafa. Hamiltonov obhod je pot v grafu, ki gre skozi vse točke grafa natanko enkrat in se na koncu vrne v začetno.

Problem spada med NP - polne probleme, kar med drugim pomeni, da zanj še vedno ne poznamo algoritma, ki problem reši v polinomskem času. Oznaka NP prihaja iz angleščine (angl. *nondeterministic polynomial*). V splošnem se časovna zahtevnost za ta problem veča eksponentno; če imamo $n$ mest, potem je trajanje izvajanja algoritma sorazmerno s $C^n$ , kjer je $C$ neka konstanta.

## 2 ILP formulacija

Rešujemo problem z $n$ mesti. Definirajmo spremenljivko

$$x_{ij} = \begin{cases} 1, & \text{uporabimo pot } i \to j \\ 0, & \text{sicer} \end{cases}$$

in naj $d_{ij}$ označuje ceno poti $i \to j$ in naj bo $d_{i,i} = \infty$ za $i = 1, \ldots, n$. Zapišimo problem v jeziku celoštevilskega linearnega programa.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} d_{ij}$$

**p.p**

(1) $\sum_{j=1}^{n} x_{ij} = 1 \quad$ za $i \in [n]$

(2) $\sum_{i=1}^{n} x_{ij} = 1 \quad$ za $j \in [n]$

(3) $x_{ij} = \{0, 1\} \quad$ za $i, j \in [n]$

(4) Rešitev je natanko en cikel

Upoštevajoč le omejitve (1), (2) in (3) se problem prevede na problem najmnajšega prirejanja opravil, ki ga lahko rešimo npr. z madžarsko metodo. Pogoj (4) zapišimo drugače

$$u_i - u_j + n x_{ij} \leq n - 1, \quad 2 \leq i, j \leq n,$$
$$u_i \in \mathbb{Z} \quad i \in [n],$$

kjer $u_i = t$ označuje korak, v katerem obiščemo $i$-to mesto ($i, t \in [n]$). Torej, če je $x_{ij} = 1$ (*izključujoč povezavo, s katero sklenemo cikel*) sledi $u_j > u_i$, saj mesto $i$ obiščemo pred mestom $j$ in ker jih obiščemo zaporedoma, smemo definirati $u_j = u_i + 1$, spet tega ne definiramo za povezavo, s katero cikel sklenemo.

Potrebno je preveriti, da nam zgornja omejitev da natanko en cikel in ne več podpoti. Dokazati moramo torej dvoje. Prvič, da pogoj prepreči pojav podpoti, in drugič, da je pogoj dober, če imamo natanko en cikel.

**Dokaz.**
Omejitev velja za $2 \leq i, j \leq n$. Mi bomo brez škode za splošnost predpostavili, da se pot trgovskega potnika začne v prvem ogljišču, v nadaljevanju pa je razloženo, zakaj omejitev ne velja, če $i, j = 1$.

1.) Da pogoj prepreči pojav podpoti, bomo dokazali s protislovjem. Recimo, da torej obstaja taka rešitev, ki vsebuje podcikel, ki ne obišče prvega ogljišča. Naj bo ta podcikel dolžine $k < n$, Seštejemo levo in desno stran omejitve (4), kjer $x_{ij} = 1$, in sicer za povezave $i \to j$, ki so v podciklu, ki ne vsebuje prvega vozlišča. Dobimo

$$\begin{aligned}
&(-1+n)(k-1) + (k-1) + n = (k-1)(-1+n+1) + n = n(k-1) + n \leq k(n-1) \\
&\Rightarrow nk - n + n \leq nk - k \\
&\Rightarrow 0 \leq -k
\end{aligned}$$

Dobimo protislovje, torej ne obstaja rešitev, ki vsebuje podpoti.

2.) Dokažimo še, da omejitev dopušča (4) natanko en cikel. Spet brez škode za splošnost začnemo v prvem vozlišču. Iz definicije $u_i$ razberemo, da je vedno res $u_i - u_j \leq n - 1$, saj je $u_i$ največ $n$ in najmanj 1.

- Če $x_{ij} = 0$ dobimo $u_i - u_j \leq n - 1$ in je to po zgornji trditvi vedno res.
- Če $x_{ij} = 1$ za $2 \leq i, j \leq n$ (to so tiste povezave, s katerimi cikla ne sklenemo) velja $u_j = u_i + 1 = t + 1$. Sledi:

$$u_i - u_j + nx_{ij} = t - (t+1) + n = n + 1 \leq n - 1$$

Razlog, da omejitev (4) ni definirana za $i, j \in [n]$ je sledeča. Recimo, da je $k$ zadnje vozlišče, ki ga obiščemo. Da se vrnemo v prvo vozlišče, kjer smo začeli, mora veljati $x_{k1} = 1$. Če bi četrti pogoj veljal za vse $i, j \in [n]$, to seveda ne bi držalo. Potrebna je previdnost pri povezavi, ki sklene cikel. Zato smo rekli, da začnemo v prvem vozlišlu (kar BSŠ lahko naredimo) in $2 \leq i, j \leq n$, kar nam tudi dopušča sklenjen cikel.
Pomankljivost ILP formulacije pa je, da število spremenljivk raste eksponentno s številom mest, tako da je težko dobiti numerično rešitev. To je tudi razlog, da smo se odločili problem eksaktno rešiti s pomočjo B&B algoritma.

# 3 Posebni tipi TSP

- **Simetrični in asimetrični problem trgovskega potnika**

  Problem trgovskega potnika na grafu je simetrični PTP, saj se cene potovanje med točkami v obe smeri enake. Asimetrični PTP pa definiramo na digrafu, kar pomeni, da sta lahko ceni potovanja med dvema točkama različni, odvisno v katero smer potujemo. Prometne nesreče, enosmerne ulice, cene letalskih kart so samo nekateri primeri, ki pokažejo, da simetrični problem PTP ni vedno pravi model. Kljub temu, se bomo v nadaljevanju omejili na simetrične PTP.

- **Metrični problem trgovskega potnika**

  V metričnem PTP cenovna funkcija zadošča trikotniški neenakosti, tj. za poljubno trojico vozlišč $(A, B, C)$ velja $d(A, C) \leq d(A, B) + d(B, C)$, kjer je z $d$ dana razdalja na metričnem

prostoru. Uvedba metrike pomeni, da je direktna povezava od $A$ do $B$ vedno najkrajša. Tako kot običajen PTP je tudi metrični PTP NP- težek problem.

V posebnem za $d$ vzamemo evklidsko metriko. Evklidski PTP je PTP, pri katerem so točke grafa točke v ravnini, cene pa so kar razdalje v običajni evklidski metriki. Evklidski PTP je poseben primer metričnega PTP, saj za poljubno metriko velja trikotniška neenakost. Izkaže se, da je tudi evklidski NP-težak problem.

# 4 Načini reševanja

Reševanja PTP se lahko lotimo s pomočjo eksaktnih algoritmov, ki nam najdejo točno (optimalno) rešitev, vendar pa so lahko neprimerni za velike probleme. V to skupino spadajo algoritmi, ki delujejo po načelu razveji in omeji. To pomeni, da za iskanje rešitve problema, ki ga skušamo rešiti s pomočjo tovrstnih algoritmov tvorimo drevo rešitev, ki jih nato sistematično pregledamo in poiščemo najboljšo rešitev. Lahko pa uporabimo hitrejše, hevristične algoritme, ki nam lahko najdejo zelo dober približek poti v sprejemljivem času, ne moremo pa zagotoviti, da je rezultat res optimalen. Najbolj direktna možna rešitev bi bila sledeča: preizkusimo vse možne permutacije (načine povezovanja oglišč v sklenjeno celoto) in izberemo najboljšo, a je časovna zahtevnost take metode $O(n!)$.

V nadaljevanju bomo izmed hevrističnih algoritmov predstavili

- **Metoda najbližjega soseda** Hevristika najbližjega soseda ali t.i. požrešni algoritem deluje tako, da pri izdelavi poti vedno izbere najbližje mesto, ki ga še nismo obiskali kot točko premika iz trenutnega mesta. Poti, ki jih najdemo na ta način, so v povprečju za 25 % daljše od optimalnih, je pa precej hitra. Ima to omejitev, da je najkrajša pot odvisna od ogljišča v katerem začnemo.
- **Metoda obratnih podpoti**

Pri eksaktnih algoritmih se bomo osredotočili na **B&B algoritem**.

# 5 Hevristični algoritmi

V nadaljevanju bomo poleg psevdokod naše algoritme ilustrirali na primeru. Generirali smo naslednjo simetrično matriko

$$A = \begin{pmatrix} \infty & 3 & 7 & 5 & 8 & 5 & 4 \\ 3 & \infty & 6 & 3 & 1 & 9 & 5 \\ 7 & 6 & \infty & 5 & 8 & 10 & 5 \\ 5 & 3 & 5 & \infty & 3 & 4 & 5 \\ 8 & 1 & 8 & 3 & \infty & 2 & 1 \\ 5 & 9 & 10 & 4 & 2 & \infty & 9 \\ 4 & 5 & 5 & 5 & 1 & 9 & \infty \end{pmatrix}.$$

**Algoritem najbližjega soseda** (imenovan tudi požrešna metoda) je bil eden izmed prvih algoritmov za reševanje problema trgovskega potnika in je zelo preprost. Deluje tako, da pri izdelavi poti vedno izbere najbližje mesto, ki ga še nismo obiskali kot točko premika iz trenutnega mesta. Obiskovanje najbližjih točk ponavalja, dokler niso obiskana vsa mesta. Algoritem najbližjega soseda je precej hiter (časovna zahtevnost algoritma je namreč $O(n^2)$, vendar ni prav natančen, saj so poti, ki jih najdemo s to metodo v povprečju za 25 % daljše od optimalnih. Ima to omejitev, da je najkrajša pot odvisna od ogljišča v katerem začnemo.
Potek algoritma po točkah:

- Začni v poljubnem vozlišču.
- Poišči najbližje (najcenejše) neobiskano vozlišče.
- Označi trenutno vozlišče kot obiskano.
- Če so vsa vozlišča označena kot obiskana, končaj, sicer nadaljuj z drugim korakom.

***Psevdokoda hevristike najbližjega soseda***
Vhod: Matrika A, začetno vozlišče
Izhod: pot, teža

**def** najbližji_sosed(A, začetek)
$Q$ = množica vseh vozlišč
$pot$ = [ ]
$teža = 0$
$u$ = začetek
while $Q$ ni prazna do:
    odstrani $u$ iz $Q$
    pot.append(u)
    izberi tak $v$ iz $Q$, da $A[u, v]$ minimalna
    $teža = teža + A[u, v]$
    u=v
return (pot, teža)

Na matriki A, ki je zapisana zgoraj, se lahko prepričamo, da je dolžina poti pri metodi najbližjega soseda res odvisna od vozlišča, kjer začnemo. Če vemo, kje začeti, nam metoda najbližjih sosedov da dober približek teže optimalne poti, to bomo pokazali v nadaljevanju.

| Začetek | Pot | Teža |
|---|---|---|
| 1 | 1-2-5-7-3-4-6-1 | 24 |
| 2 | 2-5-7-1-4-6-3-2 | 31 |
| 3 | 3-4-2-5-7-1-6-3 | 29 |
| 4 | 4-2-5-7-1-6-3-4 | 29 |
| 5 | 5-2-1-7-3-4-6-5 | 24 |
| 6 | 6-5-2-1-7-3-4-6 | 24 |
| 7 | 7-5-2-1-4-6-3-7 | 29 |

**Metoda obratnih podpoti** deluje tako, da poljubno začetno pot, ki obišče vsa mesta, izboljšuje z rotacijami. Najprej zamenjuje podpoti, ki vključujejo dve mesti, to naredimo za vsa mesta, ki ju v začetni poti obiščemo zaporedoma. Če z menjavo najdemo pot, ki ima manjšo ceno, je nova pot, kjer smo naredili rotacijo, nova optimalna pot in jo uporabimo za primerjavo pri nadaljnjih rotacijah. Nato nadaljujemo z zamenjavo podpoti s tremi mesti, vse dokler ne pride do menjave podpoti, ki vključuje $n-1$ mest. Če začnemo s potjo $1-2-3-4-5-6-7-1$, bomo najprej pogledali, če je pot $1-3-2-4-5-6-7-1$ boljša in če je, jo uporabimo za primerjavo pri naslednjih rotacijah.

***Psevdokoda hevristike obratnih podpoti***
Vhod: matrika A, začetna pot
Izhod: pot,teža

**def** obratne_podpoti (A, začetna)
n = $lenght(pot)$
opt_pot = začetna
opt_teža = $teža(A, začetna)$
for (i in 2:(n-1)):
    for (j in 2:(n-i)):
        nova_pot = (opt_pot[1:(j-1)],$rev$(opt_pot[j:(j+i-1)]),opt_pot[(j+i):n])
        nova_teza = $teža(A, nova\_pot)$
        if nova_teza < opt_teža then
            opt_pot = nova_pot
            opt_teža = nova_teža
return(opt_pot, opt_teža)

Prva *for* zanka nam pove, koliko vozlišč obrnemo, torej v prvem koraku prve $for$ zanke obračamo dve zaporedni vozlišči, v drugem koraku tri itd. . Druga *for* se sprehaja drsi po vektorju trenutne optimalne poti in obrača željene elemente. Znotraj obeh zank pa definiramo nov vektor, ki je zlepek in del optimalne poti rotiramo. V psevdokodi smo si pomagali tudi s funkcijo teža, ki jo prej nismo definirali, ta ima za vhod matriko $A$ in neko pot ter izračuna težo te poti. Psevdokoda za tak algoritem ni težka, zato jo izpuščamo. Poglejmo si rotacije na primeru, ko na matriki $A$ uporabimo začetno pot $7-6-3-5-1-4-2$. Začetna pot nam da težo 48, z rotacijami pa jo popravljamo.

| | Rotacija | Pot | Teža |
|---|---|---|---|
| | | 7-6-3-5-1-4-2-7 | 48 |
| t | 6-3 | 7-3-6-5-1-4-2-7 | 38 |
| | 6-5 | 7-3-5-6-1-4-2-7 | 33 |
| | 3-5-6-1 | 7-1-6-5-3-4-2-7 | 32 |

Najboljša hevristična metoda je hibrid med opisanima. Za začetno pot pri metodi obratnih podpoti vzamemo tisto pot, ki nam da optimalno rešitev pri metodi najbližjih sosedov. Izkaže se, da je ta rešitev blizu optimalne, a ne ponudi bistvene izboljšave poti, ki jo dobimo z metodo najbližjih sosedov, če začnemo v vozlišču, ki nam da najlažjo pot.

# 6 B&B algoritem

B&B algoritem (angl. Branch and bound) sodi med eksaktne algoritme ter tako poišče optimalno rešitev problema trgovskega potnika. Različni postopki metode B&B rešijo PTP s štirideset do šestdeset mest, ob nadgradnji postopka z različnimi tehnikami lineranega programiranja pa vse do dvesto mest.

Opišimo B&B algoritem na primeru matrike A. Na vsakem koraku torej rešujemo problem minimalnega priranja opravil, če dobimo cikel, končamo, sicer izberemo najmanjši podcikel v rešitvi in v njem vsako povezavo posebej nastavimo na neskončno. Prvi korak na matriki A nam da rešitev $x_{1,2} = x_{2,1} = x_{4,3} = x_{3,7} = x7,5 = x_{5,6} = x_{6,4} = 1$, kar nam da dva cikla, in sicer $(2-1-2)$ in $(4-3-7-5-6-4)$, ki nam da težo 23, kar je spodnja meja za optimalno pot. Spomnimo, problem trgovskega potnika je podoben problemu najmanjšega prirejanja opravil, le da imamo dodatno omejitev, in sicer zahtevo natanko enega cikla, torej je rešitev začetnega problema tudi minimum za optimalno težo poti.

Ker rezultat ni natanko en cikel, vzamemo najmnajšega, in ga razbijemo. Najmanjši cikel (2-1-2) razbijemo tako, da naredimo dve veji, na eni nastavimo $A_{2,1} = \infty$, na drugi pa $A_{1,2} = \infty$. Sedaj se spustimo nivo nižje in ponovno rešujemo ti. *assignment problem*. Na tem koraku je v obeh primerih rešitev natanko en cikel. Veja, kjer smo nastavili $A_{2,1} = \infty$ nam da cikel $(6-1-2-4-3-7-5-6)$ in težo 24, druga veja pa rezultat $(2-1-6-5-7-3-4-2)$ in enako težo. Z razvejanjem smo zaključili in lahko sklenemo, da ima optimalna pot težo 24.
V primeru, da cikla še ne bi dobili, bi vsako vejo spet delili in nove povezave nastavljali na neskončno. Pozorni moramo biti, na kateri veji smo, saj velja, da tiste povezave, ki smo jih že nastavili na neskočno, take tudi ostanejo. Pri B&B algoritmu ni potrebno, da vsako vejo do konca razvejamo (ni potrebno da v vsako vejo zaključimo tako, da dobimo natanko en cikel in pripadajočo težo). Enkrat, ko dobimo nek cikel, lahko pripadajočo težo nastavimo za zgornjo mejo optimalne poti. Ko na neki veji to vrednost presežemo, lahko vejo *zapremo*, saj v vsakem naslednjem nivoju dobimo le večjo vrednost (logično, saj vedno več povezav nastavimo na neskočno, oziroma preprečimo pot po njih).

Prednost te metode je ta, da lahko na vsakem koraku povemo, na kakšnem intervalu leži optimalna vrednost rešitve, slaba pa ta, da pride pri velikih problemih do ogromnega razvejanja.

# 7 Primerjava metod

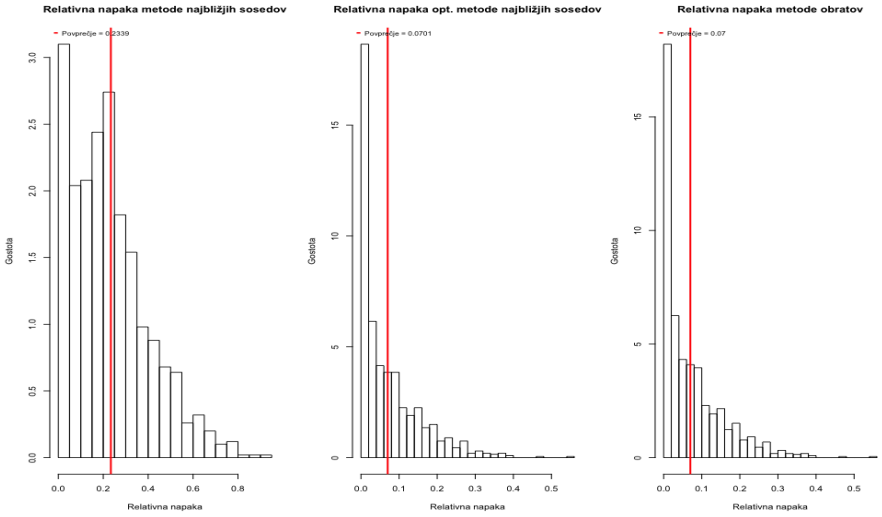
Metode smo primerjali na z dveh zornih kotov - najprej nas je zanimala natančnost hevrističnih metod, nato pa smo s simulacijami primerjali časovno zahtevnost vseh treh opisanih.

B&B algoritem nam da eksaktno vrednost problema trgovskega potnika. Sedaj lahko s simulacijami preverimo, kako dobre so hevristične metode. Simulacija je potekala na naslednji način. Vsakič znova (naredili smo 1000 simulacij) smo simulirali matriko velikosti $10 \times 10$ in

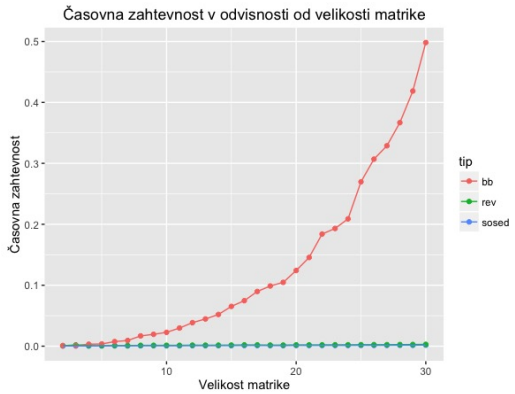
a. izračunali težo poti, če začnemo v **naključnem** vozlišču in uporabimo metodo najbližjih sosedov. Izračunali smo še eksaktno rešitev ter poračunali relativno napako, ki nam jo da opisan postopek

b. izračunali smo težo poti, ki nam jo da metoda najbližjih sosedov, če začnemo v **optimalnem** vozlišču. Spet izračunamo relativno napako.

c. Uporabimo *hibridno* metodo. Izračunamo težo pri metodi obratov, če je vhod optimalna pot, ki jo dobimo z metodo najbližjih sosedov (začnemo v vozlišču, ki nam da najlažjo pot).




Opazimo, da, če začnemo v naključnem vozlišču in naredimo obhod po metodi najbližjih sosedov, lahko v povprečju pričakujemo relativno napako 0.23. Če začnemo optimalno, pa relativna napaka v povprečju znaša le 0.7 in jo z dodatnim poizukšanjem rotacij ne bistveno zmanjšamo.




Simulacija časovne zahtevnosti je potekala na sledeč način. Simulirali smo matrike velikosti od $1 \times 1$ do $30 \times 30$. Pri vsaki velikosti smo 100-krat simulirali matriko in optimalno pot izračunali

na tri načine - z metodo najbližjega soseda, z metodo obratov in s pomočjo B&B algoritma. Na 100 simulacijah, kjer je matrika iste velikosti, izračunamo povprečne čase metod, nato se pomaknemo naprej, kjer je dimenzija matrike za ena večja od prejšnjega koraka. Opazimo, da nam da problem trgovskega potnika, če ga rešujemo z B&B algoritmom, eksponentno časovno zahtevnost, kar ni presenetljivo.