

Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Finančna matematika – 1. stopnja

Tilen Humar, Urban Rupnik

Iskanje bitonične rešitve problema potujočega trgovca

Projekt OR pri predmetu Finančni praktikum

Ljubljana, 2022

1. PREDSTAVITEV PROBLEMA

Problem potujočega trgovca oziroma **problem trgovskega potnika** je ponavadi zastavljen v naslednji obliki.

Obstaja n mest, za katera poznamo razdalje med poljubnim parom mest. Trгоvec želi obiskati vsa mesta, pri čemer pot začne in konča v istem mestu in vsak kraj obišče natanko enkrat. Katera je najkrajša oziroma najcenejša pot, ki jo lahko izbere trgovec?

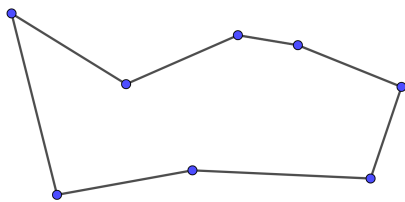
V matematičnem jeziku se problem torej prevede na iskanje najcenejšega Hamiltonovega cikla v polnem grafu K_n v ravnini, kjer ima vsaka povezava e znano utež (ceno) c_e . Ker pa je v osnovi dotični problem “NP-težek”, to je, da bi za iskanje njegove rešitve potrebovali več kot polinomski čas, se omejimo na lažjo nalogo iskanja njegove najkrajše bitonične rešitve.

2. BITONIČNA POT

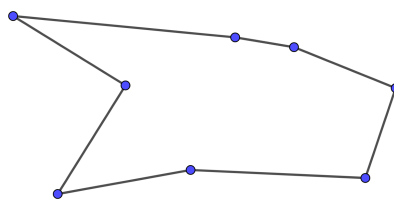
Definicija 2.1. Zaporedje $(x_n)_{n \in \mathbb{N}}$ je bitonično, ko obstaja tak $k, 1 \leq k < n$, da velja

$$x_1 \leq x_2 \leq \dots \leq x_k \geq \dots \geq x_n.$$

Bitonična rešitev problema, bo torej pot, kjer bomo začeli v skrajno desno ležečem vozlišču, nadaljevali strogo levo do najbolj levega vozlišča in še strogo desno nazaj do izhodišča. Bitoničnost poti lahko na grafu preverimo z navpičnicami. Vsaka navpična črta seka pot največ dvakrat.



(A) Bitonična pot



(B) Nebitonična pot

SLIKA 1. Primer bitonične in nebitonične poti na grafu

Iskanje najkrajše bitonične poti je standardna naloga v dinamičnem programiranju, rešljiva v polinomskem času $O(n^2)$, poznamo pa tudi hitrejši algoritem s časovno zahtevnostjo $O(n \log^2 n)$.

3. DINAMIČNO PROGRAMIRANJE

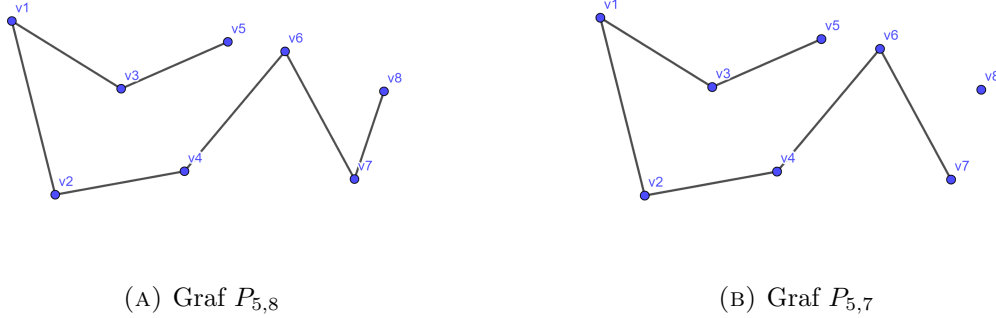
Imamo poln graf K_n z množico n vozlišč $\{v_1, v_2, \dots, v_n\}$, urejenih po naraščajoči x koordinati. Cene povezav so enake (evklidski) razdalji med posameznima vozliščema. Naš problem iskanja najkrajše bitonične poti (po definiciji dinamičnega

programiranja) razdelimo na manjše probleme.

Naj bo $P_{i,j}$ (za $i \leq j$) najkrajša bitonična pot, ki se začne v vozlišču v_i , nadaljuje strogo levo do v_1 in nato strogo desno do v_j , kjer se konča. Slednja pot obišče vozlišča $\{v_1, v_2, \dots, v_j\}$. Rešitev problema potujočega trgovca bo torej pot $P_{n,n}$ oziroma $P_{n-1,n} + e_{n-1,n}$, kjer je $e_{n-1,n}$ povezava med v_{n-1} in v_n . Razčlenimo do zdaj ugotovljeno na nekaj podprimerov, iz česar bomo izpeljali rekurzivno formulo.

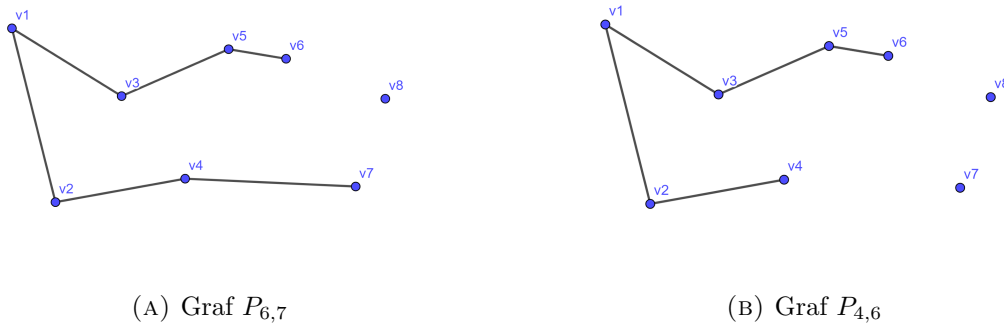
Naj za pot $P_{i,j}$ velja $i < j - 1$. V slednji poti bo tako v_{j-1} predhodnik v_j , zato velja, če iz poti $P_{i,j}$ odstranimo povezavo $e_{j-1,j}$, dobimo rešitev $P_{i,j-1}$.

Na spodnji sliki 2 je izrisan primer poti $P_{5,8}$, kjer omenjena neenakost velja ($5 < 8 - 1$). Ko smo na poti odstranili povezavo $e_{7,8}$, nam je ostala ravno pot $P_{5,7}$ (desna slika).



SLIKA 2. Primer grafov poti $P_{i,j}$ za $i < j - 1$.

Poglejmo še primer, ko za $P_{i,j}$ velja $i = j - 1$. Na poti $P_{i,j}$ bo imela točka v_j predhodnika v_k za $1 \leq k \leq j - 2$. Če sedaj odstranimo povezavo $e_{k,j}$ nam ostane pot $P_{k,j-1}$.



SLIKA 3. Primer grafov poti $P_{i,j}$ za $i = j - 1$.

Slednji primer je prikazan na sliki 3. Za pot $P_{6,7}$ velja enakost ($6 = 7 - 1$), torej ima vozlišče v_7 predhodnika v_k (za $1 \leq k \leq 7 - 2$). V našem primeru je $k = 4$ in ko iz

poti $P_{6,7}$ odstranimo povezavo $e_{4,7}$, dobimo ravno pot $P_{4,6}$.

V zadnji primer ($i = j - 1$) spada tudi skrajni dogodek $i = 1$ in $j = 2$. Imamo pot z le dvema vozliščema, torej je ta enaka kar njuni povezavi $P_{1,2} = e_{1,2}$.

Zgornje izpeljave lahko sedaj združimo v rekurzivno formulo.

$$P_{i,j} = \begin{cases} e_{1,2} & i = 1, j = 2 \\ P_{i,j-1} + e_{j-1,j} & i < j - 1 \\ \min_{1 \leq k \leq j-2} P_{k,j-1} + e_{k,j} & i = j - 1 \end{cases}$$

4. PROGRAM

V programskem jeziku R, sva pripravila program, ki vsebuje:

- funkcijo za generiranje točk v \mathbb{R}^2
- funkcijo za risanje točk
- funkcijo za izračun evklidske razdalje med točkama
- program za iskanje najkrajše bitonične poti
- funkcijo za izpis poti

4.1. Generiranje podatkov. Za generiranje podatkov oziroma točk v ravnini poskrbi pomožna funkcija `generiraj_tocke`, ki sprejme število zelenih točk in koordinate pravokotnika (`zgornja_x`, `spodnja_x`, `zgornja_y`, `spodnja_y`) s katerim omejimo območje naših podatkov.

Predpostavila sva, da so x -koordinate točk različna cela števila znotraj izbranega pravokotnika (če je slednji premajhen za prej izbrano želeno število točk, funkcija vrne največje možno število točk znotraj pravokotnika), y -koordinate pa so naključna realna števila.

4.2. Program za iskanje najkrajše bitonične poti. Glavni del programa lahko, v rahlo poenostavljeni obliki, predstavimo z naslednjo psevdokodo.

```
Razvrsti seznam_tock
Ustvari tabeli B[1...n, 1...n] in C[1...n, 1...n]
B[1,2] = razdalja(seznam_tock[1,], seznam_tock[2,])
C[1,2] = 1
for (j in 3:n)
  for (i in 1:(j - 2))
    B[i,j] = B[i,j - 1] + razdalja(seznam_tock[j - 1,], seznam_tock[j,])
    C[i,j] = j - 1
  B[j - 1,j] = B[1,j - 1] + razdalja(seznam_tock[1,], seznam_tock[j,])
  C[j - 1,j] = 1
  for (k in 2:(j - 2))
    B[j-1,j] = min(B[j-1,j], B[k,j-1] + razdalja(seznam_tock[k,], seznam_tock[j,]))
    if (B[j-1,j] == B[k,j-1] + razdalja(seznam_tock[k,], seznam_tock[j,]))
      C[j-1,j] = k
  B[n,n] = B[n - 1,n] + razdalja(seznam_tock[n - 1,], seznam_tock[n,])
  C[n,n] = n - 1
return B and C
```

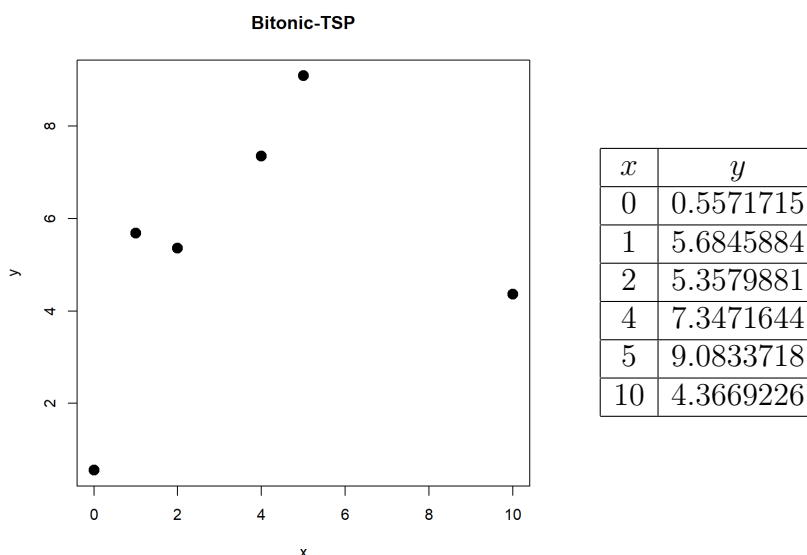
Program sprejme generiran seznam točk (x, y) , ki jih najprej razvrsti po naraščajoči x -koordinati. Matrika B vsebuje dolžine poti, kjer je element v i -ti vrstici in j -tem stolpcu ($B[i, j]$) dolžina najkrajše bitonične poti $P_{i,j}$ (od vozlišča v_i , strogo levo do v_1 in nato strogo desno do v_j). Matrika C pa na $[i, j]$ -tem mestu vsebuje predzadnje vozlišče poti $P_{i,j}$, torej predhodnika vozlišča v_j na tej poti.

Program vrne celotni matriki B in C . Rezultat našega iskanja problema najkrajše bitonične poti, ki vsebuje vsa vozlišča $(P_{n,n})$, bo tako $[n, n]$ -ti element matrike B . Matriko C pa bomo uporabili pri konstrukciji slednjega cikla.

5. EKSPERIMENTIRANJE S PROGRAMOM

Sedaj opravimo nekaj konkretnih eksperimentov z napisanom programom.

5.1. 1. eksperiment. Generirajmo 6 točk v zaprtem pravokotniku $[0, 10] \times [0, 10]$. Funkcija nam vrne naslednji seznam točk, ki jih tudi narišimo.



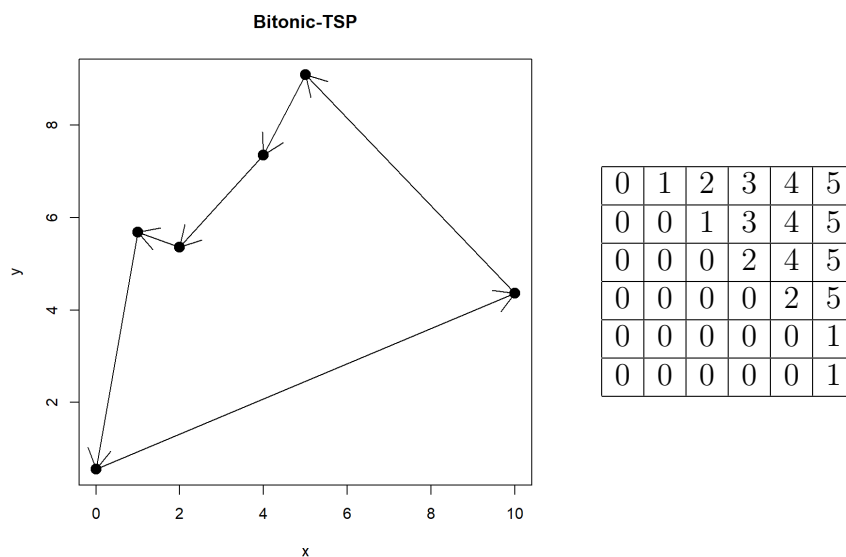
Generirane točke v ravnini

S slednjim generiranim seznamom točk poženemo program, ki nam vrne naslednji matriki B in C .

0	5.224022	6.276005	9.096789	11.10039	17.97388
0	0	10.424776	13.245560	15.24916	22.12265
0	0	0	13.854668	15.85827	22.73176
0	0	0	0	18.49453	25.36803
0	0	0	0	0	21.80152
0	0	0	0	0	28.67501

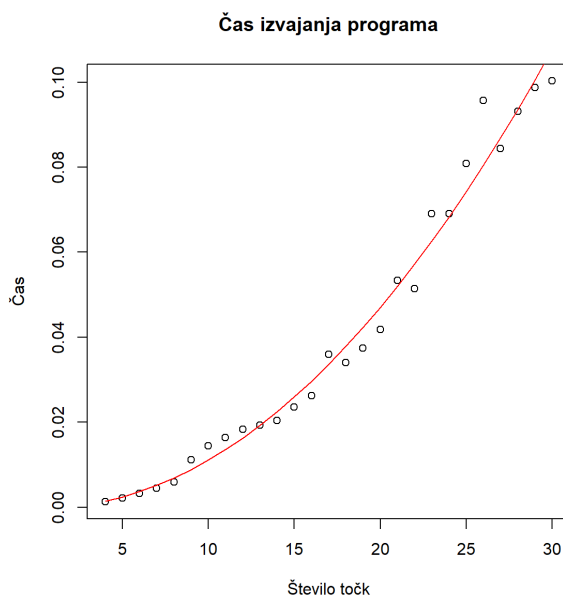
Matrika B

V matriki B na mestu $[6, 6]$ najdemo dolžino najkrajše bitonične poti za naš problem potujočega trgovca: $B[6, 6] = 28.67501$.



Cikel najkrajše bitonične rešitve $P_{6,6}$ in matrika C

5.2. Čas izvajanja programa. Glavni program za iskanje najkrajše bitonične rešitve potujočega trgovca ima, kot smo enkrat že povedali, kvadratično časovno zahtevnost $O(n^2)$. Naslednji graf 4 prikazuje porabljen čas programa v odvisnosti od števila generiranih točk. Čas generiranja točk ni vštet. Rdeča krivulja pa je kvadratna funkcija, ki se točkam najbolj prilaga.



SLIKA 4. Čas izvajanja programa (v sekundah) v odvisnosti od števila točk