Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Finančna matematika

Tina Vovk, Gašper Potočnik

Najdaljša 2-robna pot brez sekanja

Projektna pri predmetu Finančni praktikum

5. december 2021

Kazalo

1 Uvod

V projektni nalogi pri predmetu finančni praktikum sva v evklidski ravnini z množico P $3n$ točk iskala 2-robne poti, in sicer takšne, ki pokrivajo množico točk P, so paroma disjunktne in se ne križajo.
Trojice točk sva povezala tako, da se poti niso sekale in s pomočjo celoštevilskega linearnega programa poiskala rešitve optimizacijskih problemov: problema maksimizacije vsote dolžin in maksimizacije najkrajšega roba. Najprej sva izračunala vse razdalje, ki sva jih kasneje uporabila pri reševanju optimizacijskih problemov, nato pa sva preverila ali se poti sekajo. Primerjala sva razmerje med dolžinami obeh rešitev in časovno zahtevnost za naključno ustvarjene točke. Zanimalo naju je, ali se optimalna vrednost veča ali manjša, ko se n veča. Problem sva reševala v okolju SageMath za grafe pa sva uporabila program R.

2 Celoštevilski linearni program

2.1 Splošna oblika CLP

Problem sva rešila s celoštevilskim linearnim programiranjem (CLP). Za CLP vemo, da je NP-težek problem, kar pomeni, da za reševanje ne poznamo učinkovitih polinomskih algoritmov in se verjame, da ti ne obstajajo.
Celoštevilski linearni program v standardni obliki izgleda tako:
Imamo podatke:

$$A \in \mathbb{R}^{m \times n}, \quad c \in \mathbb{R}^n, \quad b \in \mathbb{R}^m \tag{1}$$

in iščemo $x \in \mathbb{Z}^n$, kjer je dosežen $max\ c^T x$ pri pogojih $Ax \leq b$, $x \geq 0$
X pri tem problemu predstavlja binarno spremenljivko z vrednostjo 1, kadar neko odločitev sprejmemo in 0, kadar neke odločitve ne sprejmemo.

2.2 CLP

2.2.1 Spremenljivke

Najina problema sta bila oba maksimizacijska. Definirala sva spodnjo spremenljivko:

$$x_{ij} = \begin{cases} 1; \text{ ko se uporabi daljica } ij, \text{ pri čemer je } i \text{ vrh} \\ 0; \text{ sicer} \end{cases} \tag{2}$$

Vsaka točka se mora pokazati na enem mestu kot list ali na dveh kot vrh.
Definirala sva realno spremenljivko $w \in \mathbb{R}$, ki sva jo uporabila pri maksimizaciji najkrajšega roba in za katero velja $w \geq 0$. Potrebovala sva tudi matriko $M \in \mathbb{R}^{n \times n}$, ki vsebuje razdalje med vsemi točkami in ima na diagonalah vrednosti 0. M_{ij} bo torej predstavljala dolžino daljice ij. Pri maksimizaciji najkrajšega roba sva definirala tudi spremenljivko $M_{max} = max(M_{ij})$, ki je najdaljša možna povezava med izbranimi točkami.

2.2.2 Cilj

Cilja najinega programa sta bila maksimizacija vsote uporabljenih povezav in maksimizacija dolžine najkrajše uporabljene povezave. Cilj prvega problema zapišemo na naslednji način:

$$max \sum_{i=1}^{n} \sum_{j=1}^{n} (x_{ij} \cdot M_{ij}) \tag{3}$$

Cilj iskanja maksimalne dolžine najkrajše uporabljene poti pa zapišemo kot:

$$max(w) \tag{4}$$

kjer je w spremenljivka, ki smo ji postavili pogoj, da je vedno manjša ali enaka najkrajši uporabljeni povezavi, kar bo natančneje opisano spodaj pri pogojih. Ko maksimiziramo w, s tem maksimiziramo tudi dolžino najkrajše uporabljene povezave.

2.2.3 Pogoji

Prvi pogoj je, da imamo točke oblike $3n$. Temu pogoju sledi pogoj, da lahko iz vsake točke gresta natanko dve povezavi, ali pa vanjo pride natanko 1 povezava:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(x_{ij} + 2 \cdot x_{ji}) = 2 \tag{5}$$

Nato sledi pogoj, da se poti med sabo ne sekajo. To sva naredila s pomočjo determinante 3×3 matrike takšne, kot je prikazana spodaj v funkciji, ki sva jo uporabila za preverjanje tega pogoja. Predznak te determinante nam pove ali je daljica med točkama usmerjena levo ali desno.
Naj bodo torej a, b in c naše točke.

$$zasuk(a,b,c) = \begin{vmatrix} 1 & a_x & a_y \\ 1 & b_x & b_y \\ 1 & c_x & c_y \end{vmatrix} \tag{6}$$

Zgoraj izračunano determinanto sva uporabila v naslednji enačbi, ki pove, da se daljici ab, cd sekata, če velja:

$$zasuk(a,b,c) \cdot zasuk(a,b,c) < 0 \wedge zasuk(a,b,d) \cdot zasuk(b,c,d) < 0 \tag{7}$$

Tako dobimo informacijo, če se dve povezavi sekata ali ne, in če se, je bila lahko uporabljena samo ena izmed njiju, kar sva zagotovila s kodo:

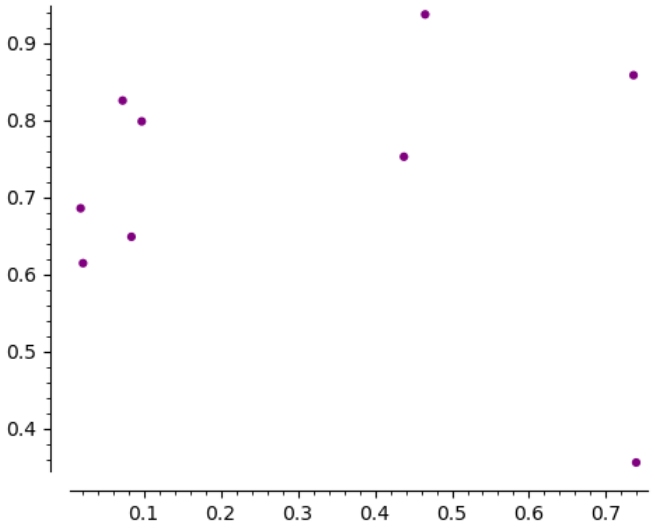
```
for i in range(n):
    for j in range(n):
        for k in range(n):
            for l in range(n):
                clp.add_constraint(x[i,j]+x[j,i]+x[k,l]+x[l,k]+seka(i,j,k,l)<=2)
```

Pri drugem problemu sva poleg prej omenjenih pogojev potrebovala tudi pogoj, da bo spremenljivka w vedno manjša ali enaka najkrajši povezavi, ki jo bomo uporabili:

$$w \leq M_{ij} + (1 - x_{ij}) \cdot M_{max} \tag{8}$$

3 Opis kode

Najprej sva za merjenje časa, ki ga bo porabil program pri določenih n uvozila knjižnico *time*. Izbrala sva željeno število točk, ki morajo biti oblike $3n$, kjer je $n \in \mathbb{N}$, sicer program vrne napako "število točk more biti pozitivno", če za število točk vzamemo negativno število in "izbrati je treba 3k število točk za k iz naravnih števil", če izberemo število točk, ki ni deljivo s 3. Točke sva naključno generirala v kvadratu $(0,1) \times (0,1)$ s pomočjo funkcije *random*(), ter jih shranila v slovar $p =$ {i: (random(), random())for i in range(n)}.



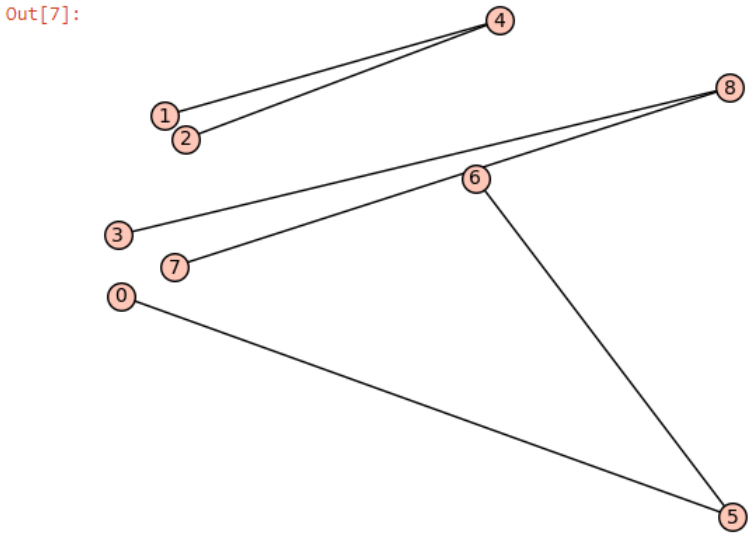
Slika 1: Naključno generirane točke v (0,1)x(01)

Nato sva definirala matriko M, v katero sva shranila razdalje med vsemi točkami. Element M_{ij} tako predstavlja razdaljo med točko i in j. $M_{ii} = 0$, kar pomeni, da so po diagonali matrike M vse vrednosti enake 0. Razdalje sva v matriko shranjevala na naslednji način:

```
for i in range(n):
    for j in range(n):
        M[i,j] = (((p[i][0])-(p[j][0]))^2 + ((p[i][1])-(p[j][1]))^2)^((0.5))
```

Preveriti sva morala, ali se povezave med točkami med seboj sekajo. To sva naredila z funkcijo $zasuk(a,b,c)$ ki je vrnila determinanto matrike A. To sva potem uporabila v funkciji $seka(a,b,c,d)$, ki nam pove ali daljica ij seka daljico lk, kjer so i,j,k,l števila med 0 in $n-1$.
Nato sva se lotila problema maksimizacije vsote robov. Definirala sva celošte-

vilski linearni program, ki maksimira vsoto dolžin vseh povezav, ki jih bomo uporabili. Potem sva definirala spremenljivko x in dodala zgoraj omenjene pogoje in cilj našega programa, torej v tem primeru maksimum vsote uporabljenih povezav. Program nama je vrnil vsoto povezav. S pomočjo funkcije $time.time()$ sva dobila podatke o času potrebnem za izvajanje tega algoritma merjen v sekundah. Nato sva preverila katere povezave smo uporabili in jih shranila v slovar, ter jih uporabila, da sva narisala graf, ki prikazuje izbrane točke in povezave.



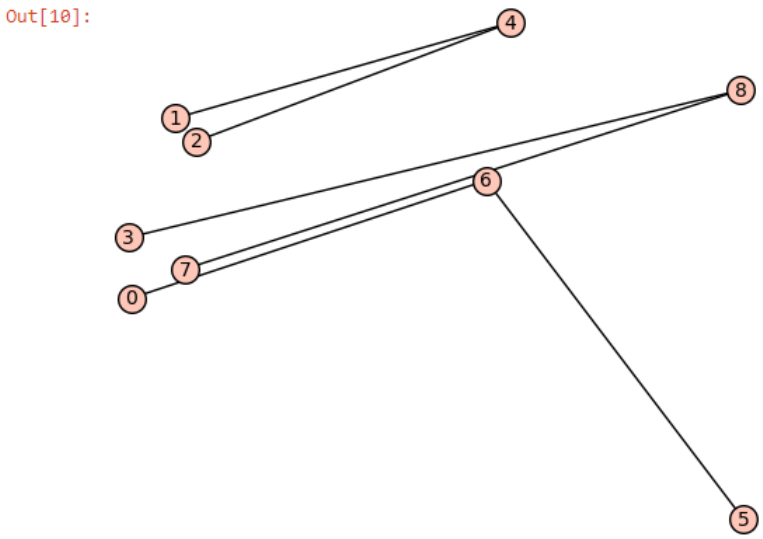
Slika 2: Povezave med točkami, generirane glede na max vsoto povezav

Pri maksimizaciji najkrajšega roba moremo povezave izbrati tako, da maksimiramo dolžino najkrajše izbrane povezave. Pri tem problemu sva morala dodatno definirati realno nenegativno spremenljivko w, ki bo vedno manjša ali enaka najkrajši povezavi, ki jo bomo uporabili (za katero je $y_{ij} = 1$). Kot cilj maksimiziramo velikost spremenljivke w, kar maksimizira dolžino najkrajše uporabljene povezave. Definirati sva morala tudi spremenljivko M_{max}, ki je najdaljša možna povezava med izbranimi točkami. Pri tem problemu sva torej morala dodati naslednji pogoj:

```
y[i,j] == 1).
for i in range(n):
    for j in range(n):
        clp2.add_constraint(w[1] <= M[i, j] + (1 - y[i, j]) * Mmax)
```

Algoritem nama je vrnil dolžino najkrajše uporabljene povezave, prav tako pa sva tudi pri tem CLP s pomočjo funkcije $time()$ merila, koliko časa pro-

gram potrebuje, da vrne rezultat. Tudi tukaj sva pogledala katere povezave smo uporabili, jih shranila v slovar in narisala graf z izbranimi povezavami.



Slika 3: Povezave med točkami, generirane glede na max najkrajšega roba

Da sva lahko primerjala vsote in čas postopkov, sva pri primeru maksimiranja najkrajše povezave izračunala še vsoto vseh uporabljenih povezav. Ustvarila sva seznam vseh povezav in seštela razdalje vseh uporabljenih povezav:

```
vsota2 = sum(M[list2[i][0],list2[i][1]] for i in range(n*(2/3)))
```

Izračunala sva razmerje med *vsoto*2 in maksimalno vsoto dolžin povezav. Poiskala sva tudi najkrajšo uporabljeno povezavo pri primeru maksimiziranja vsote povezav, tako da sva definirala seznam uporabljenih povezav in poiskala minimalno dolžino s pomočjo tega seznama:

```
dolzina = min(M[list1[i][0],list1[i][1]] for i in range(n*(2/3)))
```

Povprečne vrednosti sva izračunala tako, da sva seštela vrednosti v ustreznem stolpcu in jih delila s številom preizkusa, tako sva definirala vektor povprečnih vrednosti za maksimalno razdaljo najkrajše povezave za oba primera maksimizacije in njuno razmerje, vektor s povprečnimi vrednostmi vsot povezav za oba primera maksimizacije in njuno razmerje, ter vektor s povprečnimi vrednostmi porabljenega časa za oba primera maksimizacije in njuno razmerje na naslednji način:

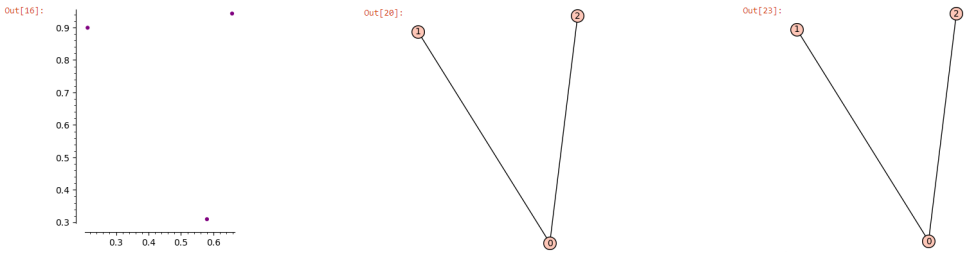
```
for j in range(5):
    for i in range(10):
        pov_raz_vsot[0,j] = pov_raz_vsot[0,j] + Vsota_razmerje[i,j]/10
```

4 Analiza rezultatov

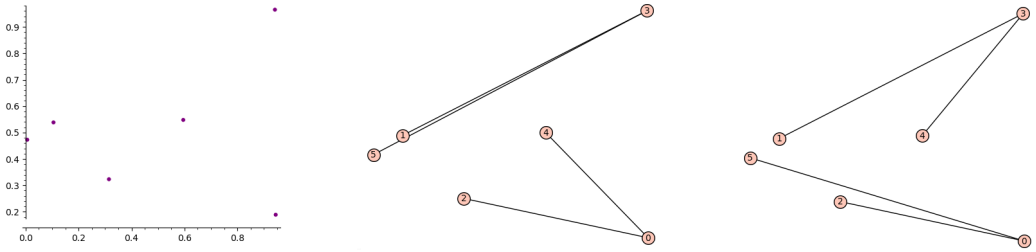
Program sva preizkusila na različnem številu točk ter rezultate poskusov shranjevala v matrikah. V vsaki matriki je 5 stolpcev, ki po vrsti zadoščajo poskusu za $n = 3, 6, 9, 12, 15$ in 10 vrstic, saj sva za vsak n program pognala 10-krat in shranila rezultate, nato pa izračunala njihovo povprečje. Tako sva definirala matriko vsot dolžin pri maksimiranju vsot, kjer stolpci torej predstavljajo rezultate 10 ponovitev poskusa pri različnih n. Na isti način sva definirala tudi matriko vsot dolžin pri maksimiranju najkrajše povezave. Razmerja med tema dvema matrikama sva shranila v novi matriki za lažjo primerjavo in sicer sva vsote iz prvega problema delila z vsotami iz drugega.

```
[1.00000000000000 1.01249972815762 1.09433573272758 1.03769196446751 1.03246457689101]
[1.00000000000000 1.00000000000000 1.01617198305794 1.08171311804303 1.05146238968603]
[1.00000000000000 1.17685275774988 1.00000000000000 1.10189361040510 1.10691055020441]
[1.00000000000000 1.00000000000000 1.00000000000000 1.02120434157170 1.06094160769596]
[1.00000000000000 1.00000000000000 1.00000000000000 1.00727991255721 1.05080218886993]
[1.00000000000000 1.00000000000000 1.02103723472299 1.23885949133115 1.13248977183535]
[1.00000000000000 1.00000000000000 1.07819531300746 1.00628496578448 1.00000000000000]
[1.00000000000000 1.00000000000000 1.03610975245193 1.04369388407678 1.11328431198552]
[1.00000000000000 1.00000000000000 1.01441107545364 1.00000000000000 1.05356425248026]
[1.00000000000000 1.04469426985450 1.04921914304855 1.02338422158566 1.07607211244847]
```

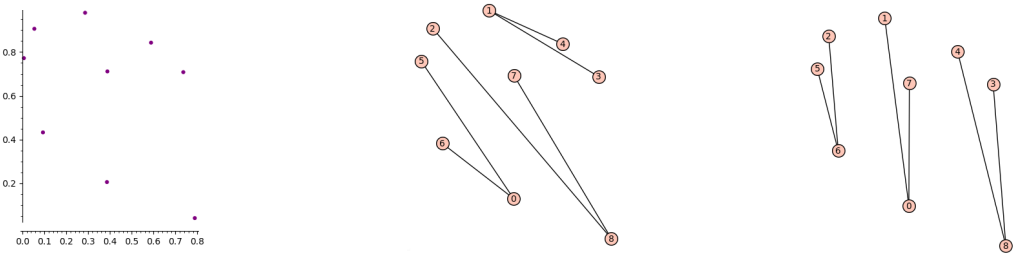
Iz matrike je razvidno, da se z večjimi n, čedalje veča tudi razmerje med vsotami, pri $n = 3$ so namreč povezave pri obeh problemih ostajale enake v vsaki izmed najinih 10 ponovitev, kot je prikazano tudi spodaj na primeru.



Za $n = 6$ se je nekajkrat že zgodilo, da so bile pri problemih uporabljene različne povezave, kar se pozna tudi pri razmerjih kot je videti v matriki. Z večanjem n se veča tudi možnost uporabe različnih povezav pri obeh problemih. Spodaj je prikazan primer različno uporabljenih povezav za $n = 6$ in za $n = 9$, kjer prvi graf predstavlja sliko naključno generiranih točk, drugi povezave glede na maksimizacijo vsote dolžin, tretji pa povezave glede na maksimizacijo najkrajše povezave.



Slika 4: Povezave v grafih glede na oba problema za n=6



Slika 5: Povezave v grafih glede na oba problema za n=9

S pomočjo matrike v katero sva shranila povprečne vrednosti razmerja vsot, vidimo, da se razmerja med vsotami večajo, kadar večamo število uporabljenih točk.

```
[1.00000000000000 1.02340467557620 1.03094802344701 1.05620055098226 1.06779917620969]
```

Slika 6: Matrika povprečnih vrednosti razmerij vsot

Enako kot prej sva definirala tudi matriko dolžin najkrajše uporabljene povezave pri maksimiranju vsot in dolžin najkrajše uporabljene povezave pri maksimiranju najkrajše povezave, kjer sta obe matriki spet elementa $\mathbb{R}^{5\times 10}$ in stolpci predstavljajo rezultate za različne n v 10 ponovitvah poskusa. Razmerja sva izračunala in jih shranila v novi matriki, kjer sva delila dolžino najkrajših povezav pri prvem problemu z dolžino najkrajših poti pri drugem problemu. Rezultat je bil takšna matrika:

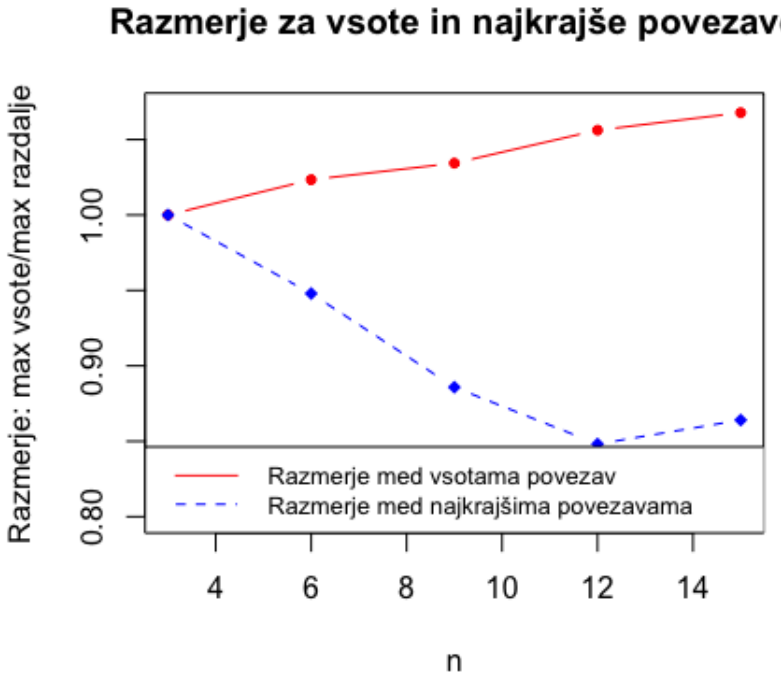
```
[ 1.00000000000000 0.730199159386745 0.933949127940258 0.666153187922696 0.724841668320586]
[ 1.00000000000000  1.00000000000000 0.946863591915966 0.781223719245016  1.00000000000000]
[ 1.00000000000000  1.00000000000000  1.00000000000000 0.871717562479080 0.954143603489697]
[ 1.00000000000000  1.00000000000000  1.00000000000000 0.881082105216653 0.902624170552039]
[ 1.00000000000000  1.00000000000000  1.00000000000000  1.00000000000000 0.765117723127253]
[ 1.00000000000000  1.00000000000000 0.814783316117972 0.702554811174273 0.955752458463634]
[ 1.00000000000000  1.00000000000000 0.748142785876080  1.00000000000000  1.00000000000000]
[ 1.00000000000000  1.00000000000000  1.00000000000000 0.923485330339241 0.774023842337408]
[ 1.00000000000000  1.00000000000000 0.943382391950100  1.00000000000000 0.789869592485818]
[ 1.00000000000000 0.749251673049195 0.859335045914712 0.655582913667197 0.774090944562114]
```

Tudi tukaj so razmerja pri $n = 3$ pri vseh desetih ponovitvah enaka in so z večanjem n vedno redkeje enaka 1. Opazimo tudi, da se razmerja z večjimi n manjšajo. To se opazi tudi s pomočjo matrike, v katero sva izračunala povprečna razmerja razdalj, vendar pa nama, da bi bila o tem popolnoma prepričana, manjkajo podatki za večje n.

```
[ 1.00000000000000 0.947945083243594 0.924645625971509 0.848179963004416 0.864046400333855]
```

Slika 7: Matrika povprečnih vrednosti razmerij dolžin

Da bi se bolje prepričala, kako se spreminjajo vrednosti razmerij, ko večamo število uporabljenih točk, sva rezultate predstavila grafično s pomočjo programa R in tudi tukaj je razvidno, da se razmerja maksimalne vsote razdalje večajo, medtem ko za razmerje med najkrajšima povezavama ne moremo reči zagotovo, da padajo, saj so pri najinem poskusu pri $n = 15$ začela ponovno rasti.



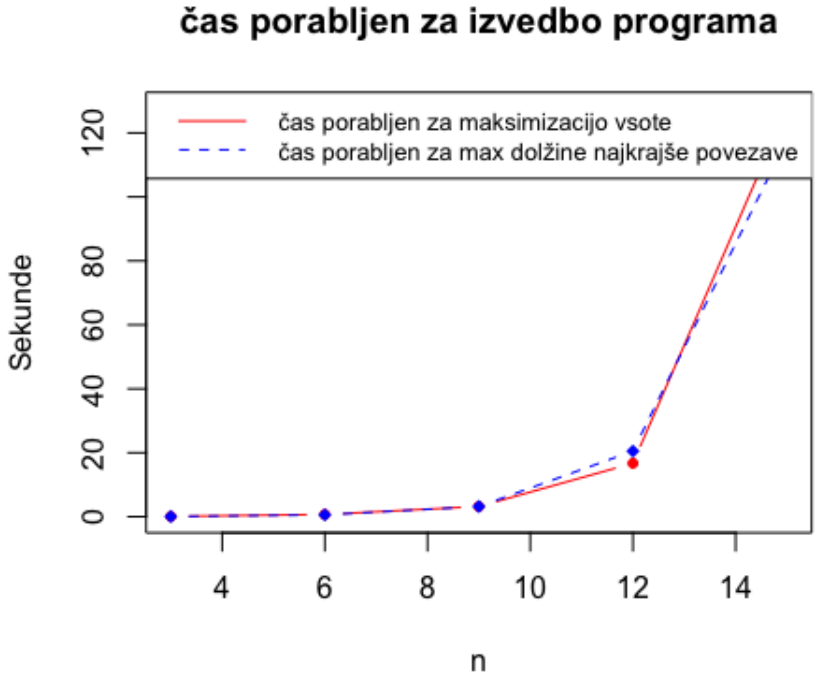
Navsezadnje sva preverila še čas porabljen pri maksimiziranju vsot in najkrajše povezave s prej omenjeno funkcijo *time*(). Tudi tukaj sva rezultate pri obeh primerih shranila v matrike in izračunala njihova razmerja ter jih shranila v spodnjo matriko.

```
[ 1.65876931314058  1.42218614586512 0.848161691750979 0.879849998525309  2.11519022705577]
[ 6.29818719940806  1.74042603386632  1.01088978386977 0.863850340824775 0.648189242140452]
[0.895170881186142  3.54190072344484  6.92817532259785 0.777058644394164 0.528178967383628]
[ 6.08313929798009 0.541352411066445  1.28242646288542 0.685885036007035  1.45409329476936]
[ 1.74494654124314 0.947085348923988  1.08163063787788 0.598012055159910 0.833402598968654]
[ 1.94032055692719  1.01197035902203  1.14889509161329  1.27574983418853  1.13603471082809]
[ 1.74833055091820  1.03596550644478  1.11948450640862 0.773222899934013 0.841191637293291]
[0.134971082141595  1.02828912278584 0.787645330845568 0.799640311637019  1.45053868119250]
[ 2.96521708108879  1.01345607189609  1.10009739506473 0.760006691681803  1.71266700827157]
[ 4.55636597531523  1.06030401957941 0.721098083959216 0.821060844207384  1.13126525606639]
```

Za čas naju je prav tako zanimala povprečna vrednost pri različnih n, zato sva definirala matriko, v kateri sva izračunala še povprečne vrednosti.

```
[ 2.80254184793490  1.33429357428949  1.60285043068733 0.823433665655994  1.18507516239697]
```

Pri merjenju časa iz matrike razmerij in iz matrike s povprečnimi vrednostmi razmerij porabljenega časa pri različnih n ne moremo razbrati, kaj točno se z vrednostmi dogaja, ko mi večamo število točk, zato se tukaj bolj splača pogledati, koliko časa potrebujemo za vsak problem posamično, torej brez uporabe razmerij. Vidimo, da se z večanjem števila točk drastično poveča čas, ki smo ga potrebovali da smo dobili željene rezultate našega problema. Tudi to sva se odločila prikazati grafično s pomočjo programa R in grafični prikaz potrjuje prej povedano.



Za $n = 18, 21, 24, 27, 30$ je program porabil zelo veliko časa, zato sva poskuse izvajala naknadno in njihove rezultate shranila v datoteki *Jason*. Rezultati so bili naslednji:
Za $n = 18$ smo za računanje prvega problema potrebovali 476.6385910 sekund, za $n = 21$ smo potrebovali 3634.5325091 in za $n = 24$ smo potrebovali 23340.6318052 sekund. Za ostale n je program delovall predolgo, zato do zdaj nimama še nobenih podatkov. Iz tega vidimo, da čas z večanjem n res raste eksponentno, kot je nakazoval prejšnji graf.

5 Zaključek

V najini projektni nalogi sva se ukvarjala s problemom maksimizacije vsote dolžin in maksimizacije najkrajšega roba. Ugotovila sva, da je program časovno zelo zahteven za veliko število točk in, da se čas izvajanja programa eksponentno povečuje. Iz primerov sva ugotovila, da se z večanjem n veča tudi možnost uporabe različnih povezav pri obeh problemih. Pri proučevanju razmerij pa sva ugotovila, da razmerja med vsotami dolžin pri maksimiranju poti in vsotami dolžin pri maksimiranju najkrajše povezave naraščajo, ko narašča n, medtem ko za razmerja med najkrajšimi povezavami pri obeh maksimizacijskih problemih glede na našo analizo ne moremo zagotovo reči, kako se bodo gibala.