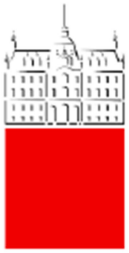
Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*



Perfect non-cross matching between $2n$ points in Euclidean space

Optimization problem of maximizing and minimizing the total length of the segments defined by the matching using ILP solver

Avtorja:
- Maj Gaberšček
- Aljoša Rebolj

Ljubljana, 2021

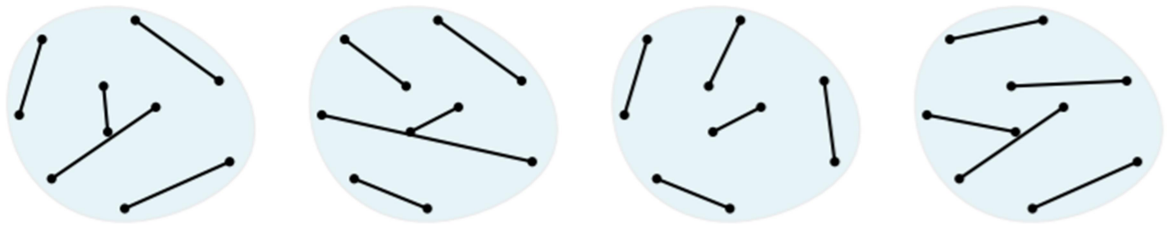
Kazalo vsebine

1. Uvod

V seminarski nalogi sva se lotila reševanja zahtevnega problema povezovanja sodega števila točk v evklidski ravnini v pare, kjer se daljice med pari točk ne sekajo. Problem sva reševala z uporabo celoštevilskega linearnega programiranja, najprej v okolju MATLAB in nato še v okolju SageMath.

Problem je enostaven za opisati, a kot bomo videli v seminarski nalogi zelo zahteven za izračun. Cilj reševanja problema je, da dobimo pare točk, za katere vemo, da je vsota dolžin daljic med pari največja oziroma najmanjša, odvisno od problema. Na spodnji sliki so prikazani štirje primeri povezovanja za deset različnih točk, a po navadi ima le en največjo in en najmanjšo vsoto dolžin vseh daljic.

Slika 1: Prikaz različnih načinov povezovanja desetih točk



2. Celoštevilski linearni program

Problem bova reševala s pomočjo celoštevilskega linearnega programiranja (CLP). Logika za načinom definicije reševanja problemov s CLP je zelo naravna, a v praksi računsko izjemno zahtevna, saj je CLP NP-težek problem, se pravi ne poznamo učinkovitih (polinomskih) algoritmov za reševanje. Za probleme potrebujemo veliko računsko moč, kar bova pokazala tudi v najini seminarski nalogi.

Celoštevilski linearni program je definiran na naslednji način. Imamo podatke:

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

in iščemo

$$x \in \mathbb{Z}^{n},$$

kjer je dosežen

$$\begin{aligned} &\max c^{T} x \\ &\text{pri pogojih } Ax \leq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

Pri reševanju najinega problema bova pri pogojih privzela $Ax = b$ in $1 \geq x \geq 0$, saj je taka, kot bova predstavila narava najinih pogojev.

3. Poimenovanje spremenljivk

V seminarski nalogi bova s P označila množico, ki vsebuje $2n$ točk $\alpha_i = (\alpha_i(1), \alpha_i(2)) \in \mathbb{R}^2, i = 1, 2, 3, \dots, 2n$ in $n \in \mathbb{N}$. Povezava uv predstavlja usmerjeno povezavo iz točke $\alpha_u \in P$ v točko $\alpha_v \in P$. Definiramo tudi

$$r_{uv} = |\alpha_u - \alpha_v|,$$

razdaljo med točkama, ki jo potrebujemo za računanje vrednosti maksimalne in minimalne vsote uporabljenih daljic uv. Očitno je, da $r_{uv} = r_{vu}$. Ker bova problem reševala s pomočjo celoštevilskega linearnega programiranja potrebujeva tudi binarno spremenljivko

$$x_{uv} = \begin{cases} 1, uporabimo\ povezavo\ iz\ \alpha_u\ v\ \alpha_v \\ 0, sicer \end{cases}$$

Kot $f(\alpha_u, \alpha_v)$ označimo daljico med α_u in α_v in jo potrebujemo pri pogoju o sekanju daljic med pari točk.

4. Pogoji

Prvi zelo očiten pogoj je, da potrebujemo sodo število točk. Brez tega pogoja je vse nadaljnje delo nesmiselno in zato tudi število točk definiramo kot $2n$, kjer $n \in \mathbb{N}$. Upoštevati moramo tudi, da je vsaka točka lahko vozlišče le ene neusmerjene povezave oziroma dveh usmerjenih nasprotnih povezav, kot sva se midva lotila reševanja problema. Kot dodaten in najbolj zanimiv pogoj povemo, da se daljice oziroma povezave, ki jih uporabimo med seboj ne smejo sekati.

Problem sva zastavila tako, da sva namesto polnega grafa z neusmerjenimi povezavami vzela poln graf usmerjenih povezav. Dodala sva pogoj

$$x_{uv} = x_{vu}$$

in tako poskrbela, da če izberemo povezavo oziroma daljico iz α_u v α_v, moramo izbrati tudi obratno povezavo iz α_u v α_v. Povezovanje je popolno v kolikor imamo n neusmerjenih oziroma $2n$ usmerjenih povezav uv, kjer je vsaka točka $\alpha_i \in P$ vozel natanko ene usmerjene oziroma dveh usmerjenih povezav in velja $x_{uv} = x_{vu}$. Na ta način poskrbimo, da ne pride do veriženja usmerjenih povezav. Naslednja stvar, ki nas skrbi v polnem usmerjenem grafu je, da gre lahko iz vsake točke le ena povezava in v vsako točko le ena povezava.

Pogoji, da gre lahko iz vsake točke le ena povezava predstavimo na naslednji način:

$$\sum_{v=1}^{2n} x_{uv} = 1, za\ \forall\ u = 1, 2, \dots, 2n.$$

Pogoji, da gre lahko v vsako točko le ena povezava pa podobno:

$$\sum_{u=1}^{2n} x_{uv} = 1, za\ \forall\ v = 1, 2, \dots, 2n.$$

Zadnji pogoj opisuje, da se daljice, ki so izbrane med seboj ne smejo sekati. Psevdo koda pogoja:

$$\begin{aligned} &if\ x_{uv} = x_{vu} = 1 \\ &\quad for\ i = 1,2, \dots, 2n-1\ and\ i \neq u \neq v \\ &\quad\quad for\ j = i+1,2, \dots, 2n\ and\ j \neq u \neq v \\ &\quad\quad\quad if\ f(\alpha_u, \alpha_v) \bigcap f(\alpha_i, \alpha_j) \neq \emptyset \\ &\quad\quad\quad\quad x_{ij} = x_{ji} = 0 \end{aligned}$$

V programu je pogoj napisan kompleksno in bo opisan kasneje.

5. MATLAB

Zaradi boljšega znanja sva se reševanja lotila v MATLAB-u, ki je znan po uporabi za numerično računanje. MATLAB ima tudi vgrajeno funkcijo z reševanje celoštevilskih linearnih programov z imenom *intlinprog*. Funkcija je definirana tako, da za argumente sprejme vektor f, matriki A in Aeq, vektorja b in beq ter vektorja lb in ub, kjer je problem definiran na naslednji način:

iščemo

$$x \in \mathbb{Z}^n,$$

kjer je dosežen

$$\begin{aligned} &\max fx \\ &\text{pri pogojih } Ax \leq b \\ &\qquad\qquad Aeq = beq \\ &\qquad\qquad ub \geq x \geq lb \end{aligned}$$

Sisteme enačb sva dobro zastavila, a se je izkazalo, da MATLAB za reševanje problemov CLP ni optimalen, saj sva dobila naslednje sporočilo:

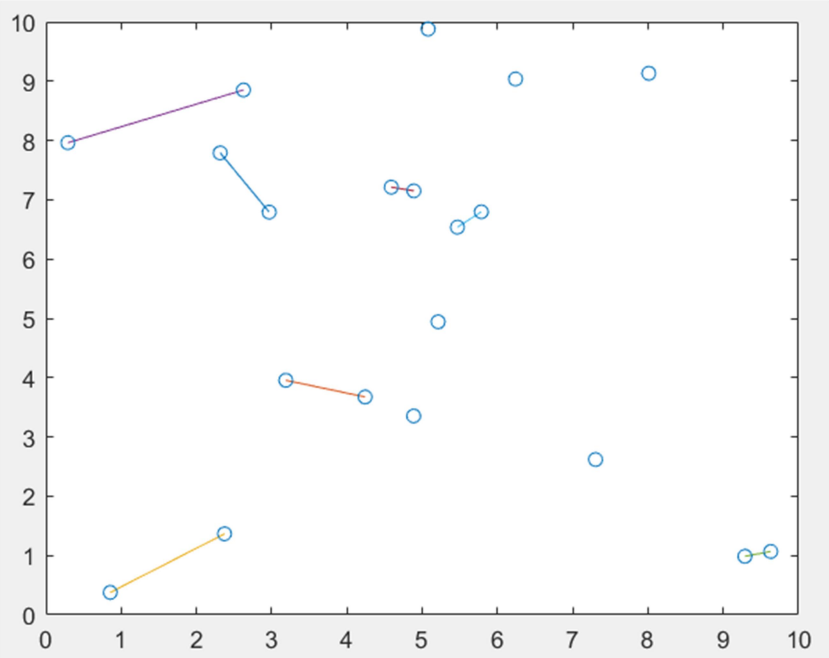
Slika 2: sporočilo MATLABA ob nenatančni rešitvi problema

```
Intlinprog stopped at the root node because the objective value is within a gap tolerance of the optimal value, options.AbsoluteGapTolerance = 0
(the default value). The intcon variables are integer within tolerance, options.IntegerTolerance = 1e-06 (the selected value).
```

Kljub ročno nastavljenim nastavitvam za največjo možno natančnost se z večanjem števila točk veča tudi verjetnost, da MATLAB ne izračuna koeficientov x, ki bi bili celo število med 0 in 1, se pravi ali 0 ali 1, ampak vrne vrednost koeficienta 0.5. To za naš problem ni uporabno saj za vrednost koeficienta 0.5 ne vemo ali to pomeni povezavo med točkama ali ne.

Kot se vidi na spodnji lahko povežemo le pare točk za katere dobimo vrnjen koeficient 1 in tako ostane nekaj točk nepovezanih.

Slika 3: Grafični prikaz rešitve problema za 20 točk

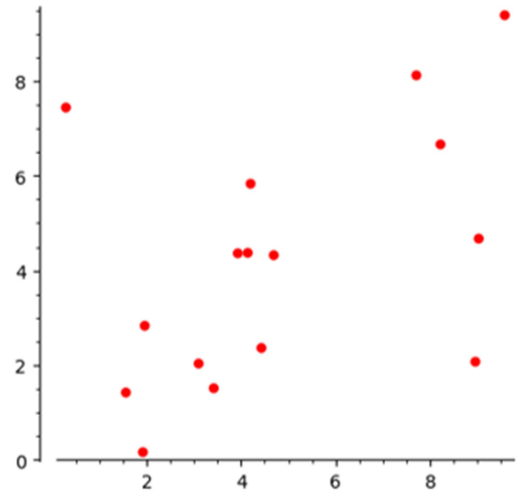


Zaradi teh težav sva se lotila reševanja v drugem okolju. SageMath je program, ki temelji na programskem jeziku Python in je osnovan za računanje in primeren za najin problem.

6. SageMath in opis kode

Slika 4: Naključno izbranih 16 točk v prostoru $[0{,}9.99]$ *x* $[0{,}9.99]$

```
Out[63]: Izbrane točke:
[4.14, 4.37]
[9.03, 4.67]
[1.96, 2.83]
[0.29, 7.44]
[4.43, 2.36]
[8.96, 2.07]
[9.58, 9.39]
[4.69, 4.32]
[3.93, 4.36]
[3.1, 2.03]
[7.71, 8.12]
[3.42, 1.51]
[1.92, 0.16]
[1.56, 1.42]
[4.2, 5.83]
[8.22, 6.66]
```



Na začetku programa sva nastavila število točk s spremenljivko »*stevilo_tock*«. Če pustimo seznam »*zacetne_tocke*« prazen in ne vnesemo svojega seznama točk, bo program naključno izbral »*stevilo_tock*« točk v prostoru $[0{,}9.99]$ x $[0{,}9.99]$. Program si bo pomagal s paketom »*random*«. V nastavitvah si nato izberemo še način, bodisi želimo maksimizirati skupno dolžino povezav ali pa jo minimizirati. Če želimo izbrati točke po meri, jih izpišemo kot dvoelementne sezname v seznam »*zacetne_tocke*«.

V nadaljevanju program izpiše izbrane točke (bodisi naključno generirane, bodisi izbrane po meri). Na sliki zgoraj je program naključno generiral 16 točk in jih izrisal v ravnini.

Za vsak par točk poiščemo razdaljo med njima. To izračunamo s funkcijo:

```
def razdalja(tocka1, tocka2): return sqrt((tocka1[0]-tocka2[0])^2 + (tocka1[1]-tocka2[1])^2)
```

S pomočjo le-te nato definiramo matriko razdalj $R = r_{u,v=1,\dots,2n}$, pri čemer r_{uv} predstavlja razdaljo med točkama α_u in α_v. Potem za vsako povezavo izračunamo, s katerimi drugimi možnimi povezavami se seka in s katerimi se ne. Ali se dve povezavi sekata ali ne, nam pove funkcija `alisesekata(t1, t2, t3, t4):`, ki nam vrne vrednost 1, če se daljica skozi točki t_1 in t_2 seka z daljico skozi t_3 in t_4, sicer pa nam vrne vrednost 0. Sekanje daljic izračuna tako, da obe daljici pretvori v premici, ki gresta skozi daljici, potem izračuna presečišče teh dveh premic. Nato pogleda ali presečišče leži na daljicah ali ne. Iz tega naredimo matriko presečišč P, ki je velikosti n^2. Matrika na mestu $(2n \cdot i + j, 2n \cdot k + l)$ vsebuje informacijo o tem, ali se daljici $f(\alpha_i, \alpha_j)$ in $f(\alpha_k, \alpha_l)$ sekata.

Nato ustvarimo linearni program in vanj vpišemo zahteve:

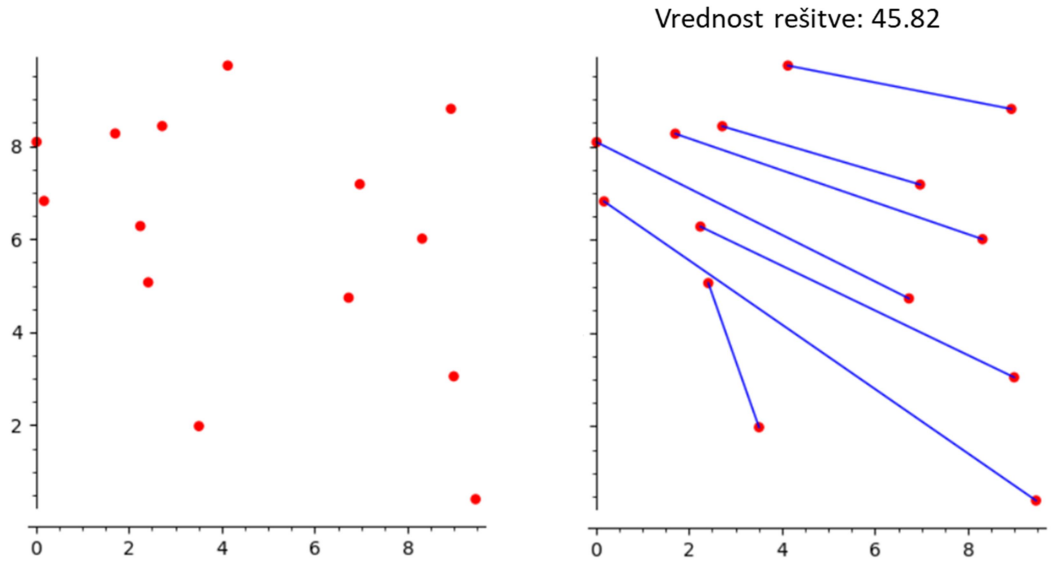
- iz vsake točke vodi samo ena povezava
- v vsako točko pride samo ena povezava
- če je uporabljena povezava uv mora biti uporabljena tudi vu
- povezave se ne smejo sekati (tukaj si pomagamo matriko presečišč)

Pri cilju programa si pomagamo z matriko razdalj (razdalje namreč želimo maksimirati).

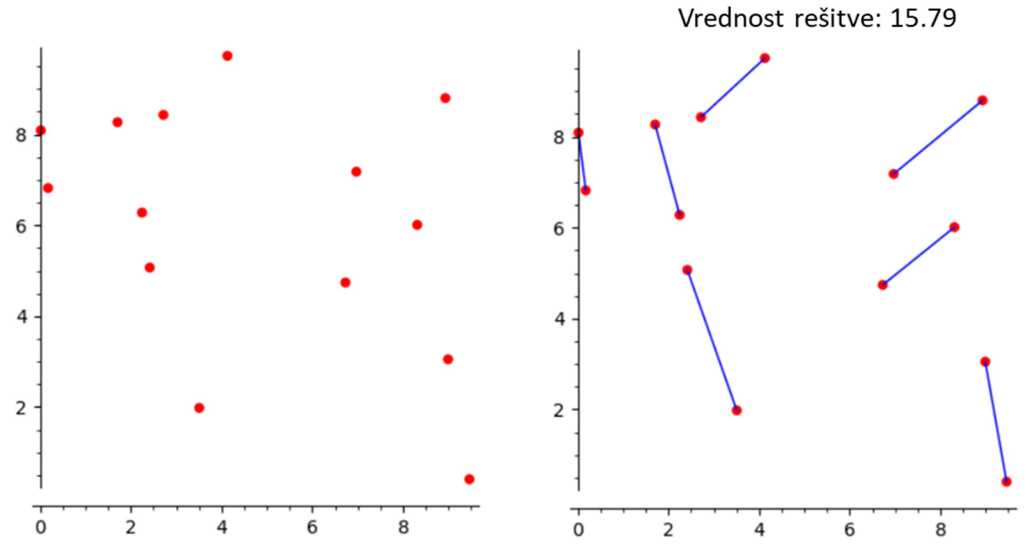
Po tem ko računalnik reši CLP, nam napiše vrednost vsote najdaljših oz. najkrajših med sabo ne sekajočih daljic.

Spodaj je primer rešitve v primeru minimiranja skupne dolžine povezav naključno izbranih 14 točk ter maksimiranja skupne dolžine povezav istih točk.

Slika 5: Rešitev maksimiranja vsote dolžin daljic za 16 točk

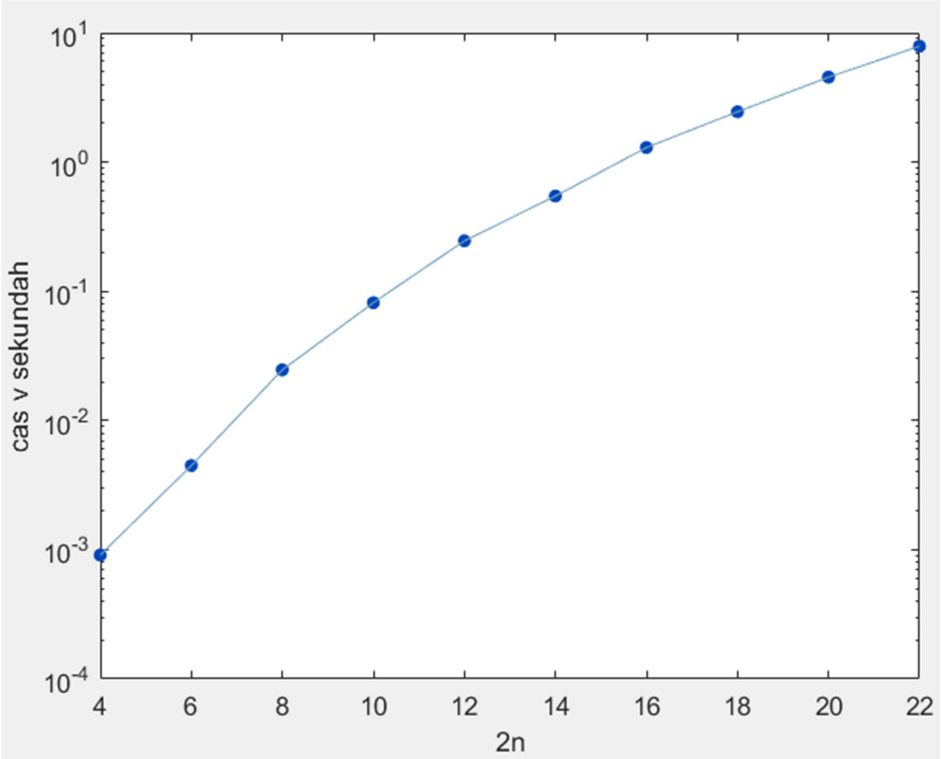


Slika 6: Rešitev minimiranja vsote dolžin daljic za 16 točk

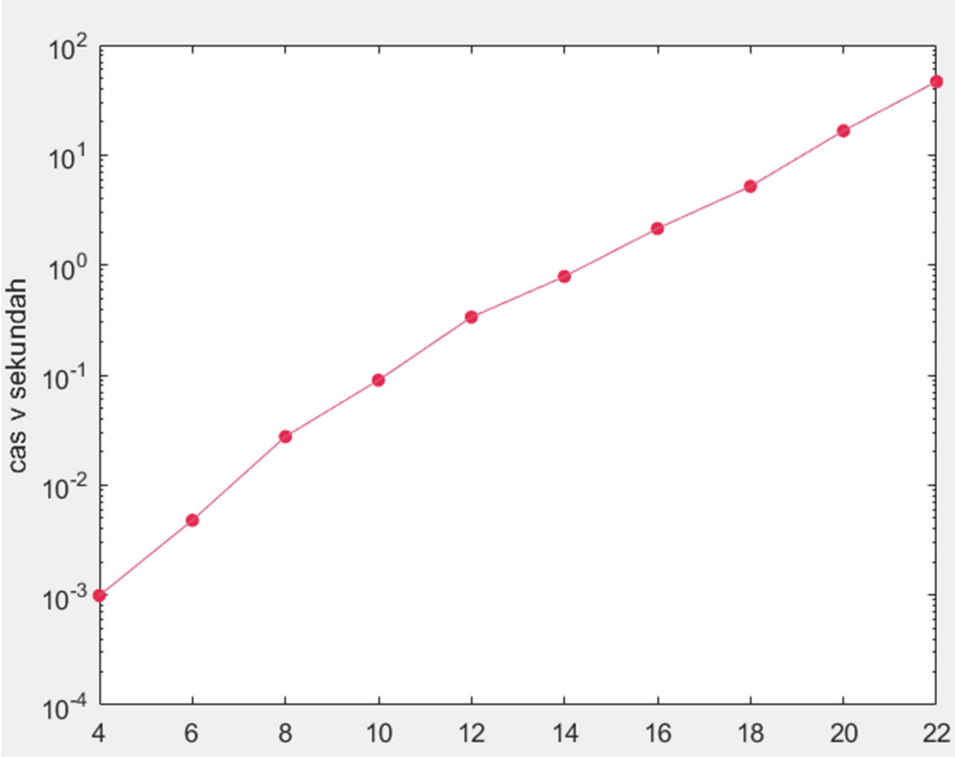


Izjemno zanimive so ugotovitve o časovni zahtevnosti algoritma in primerjavi med iskanjem maksimalne vsote dolžin daljic in minimalne vsote dolžin daljic. Potreben čas za izračun sva izmerila za sodo število točk od 4 do 22, saj kot bova prikazala na sliki postane problem izjemno časovno zahteven in prihaja do težav s platformo CoCalc zaradi zastonjske verzije uporabniškega profila.

Slika 7: Potreben čas za rešitev problema minimiranja vsote dolžin daljic

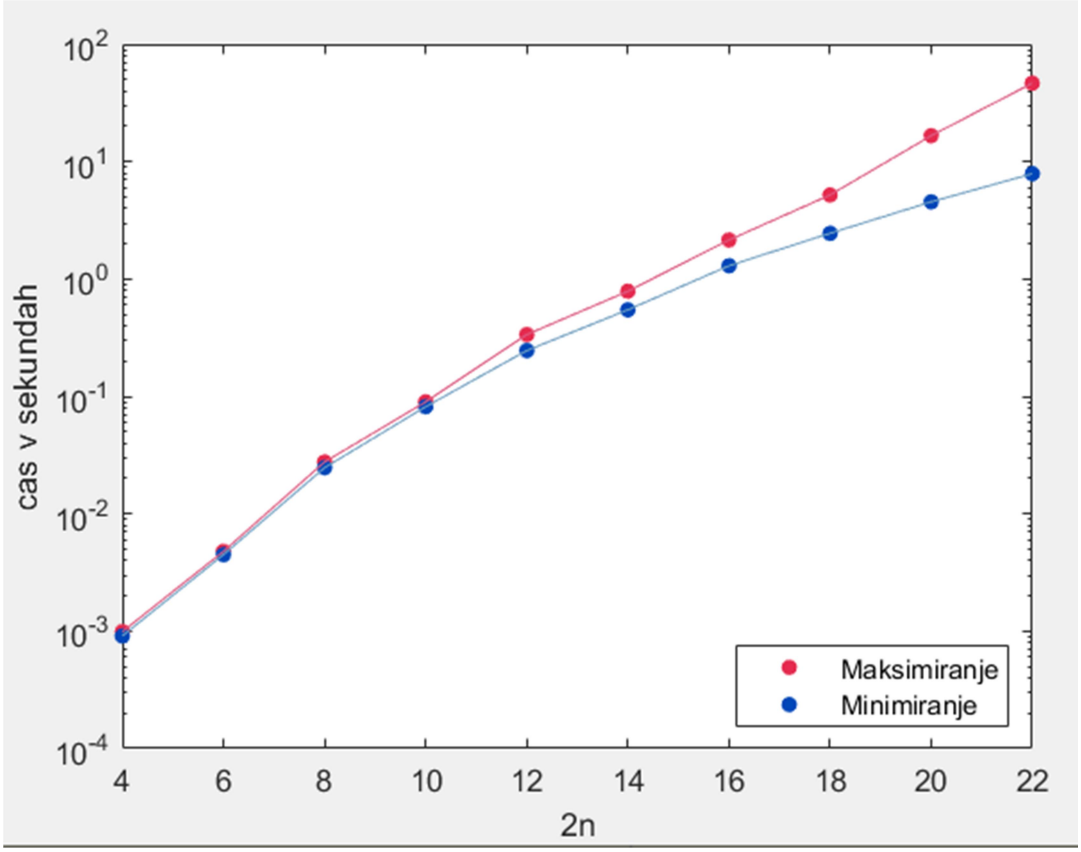


Slika 8: Potreben čas za rešitev problema maksimiranja vsote dolžin daljic



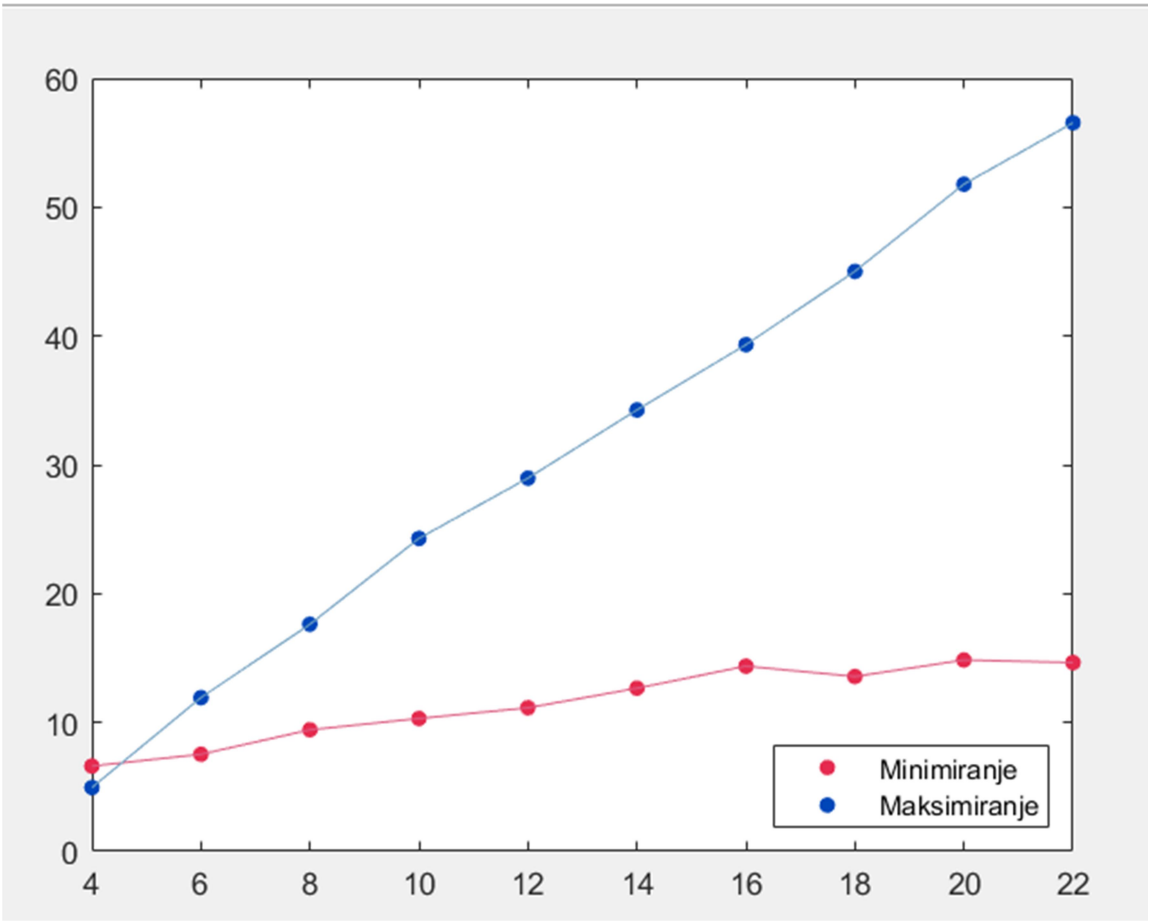
Na prvi pogled izgleda časovna zahtevnost približno enaka, a ko primerjamo podatke skupaj vidimo, da je časovna zahtevnost maksimiranja vsote dolžin daljic bistveno večja.

Slika 9: Primerjava časovne zahtevnosti maksimiranja in minimiranja problema



Za zaključek še prilagava graf, ki prikazuje vrednosti problema maksimiranja in minimiranja pri različnem številu točk.

Slika 10: Vsota dolžin daljic pri maksimiranju in minimiziranju



7. Zaključek

Problem se je na koncu izkazal za kar zahtevnega, precej je bilo težav predvsem zaradi pogoja o (ne)sekanju. Precej časa sva porabila tudi za to, da sva postavila podlago, torej definirala razdalje, povezave, … saj na začetku nisva natanko vedela, kaj vse rabiva.

Ko sva enkrat imela celoštevilski linearni program zapisan »na papirju«, z zapisom kode ni bilo preveč težav. Misliva, da je bil najtežji del najinega problema predvsem ugotoviti, kako celoštevilski linearni program zastaviti.

Glede na zgornje grafe se zdi časovna zahtevnost eksponentna. Eksperiment sva izvedla samo na točkah od $2n = 4$ do $2n = 22$, saj program za $2n = 24$ točk rešitve ni zmogel izračunati v manj kot 10 minutah.

Za konec naj poudariva še, da je opis kode v tem dokumentu zapisan bolj vsebinsko. V sami kodi so namreč tudi vmesni komentarji, ki še posebej razlagajo, kaj katera funkcija naredi in kje se kaj dogaja.

S tem problemom sva definitivno obnovila najino znanje Operacijskih metod iz lanskega šolskega leta, poleg tega pa sva se naučila tudi precej novega, predvsem o celoštevilskem programiranju s pomočjo Matlaba in Sagematha, pa tudi nekaj splošnega o tem, kako definiramo CLP-je za reševanje problemov z grafi oz. točkami.