Univerzitet u Beogradu

Matematički Fakultet

Minimum Linear Arrangement

Projekat iz Računarske Inteligencije

Milan Brčin

Milica Obradović

Sadržaj

1 Uvod

U domenu računarske inteligencije je od izuzetne važnosti optimizacija NP teških problema koji igraju ogromnu ulogu i u pravom životu, pored nauke. Problem koji je od velikog značaja zadnjih godina i koji je privukao poprilično pažnje je problem minimalnih linearnih rasporeda (MinLA). Zadnjih godina se sve više razvijaju razni sistemi veštačkih inteligencija koji koriste mnogo podataka i imaju potrebu da ti podaci budu što linearniji i organizovaniji tako da se za ove potrebe koriste algoritmi kao što je MinLA.

2 Opis problema

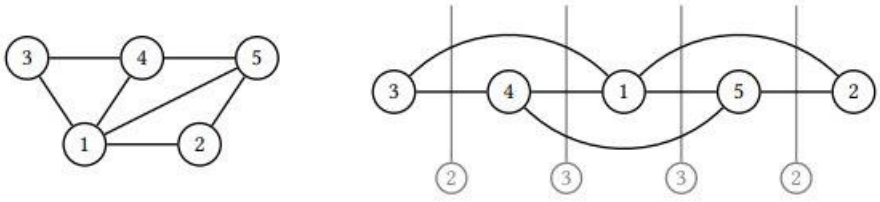
Minimum Linear Arrangements (MinLA) je koncept koji se koristi u okviru oblasti Računarske Inteligencije koji predstavlja problem rasporedjivanja elemenata u odredjenom redosledu sa što manjim troškovima.

U svetu se ovakvi problemi vidjaju svakodnevno. Na primer kada je potrebno organizovati neke dogadjaje ili kod pravljenja različitih vremenskih rasporeda. Cilj je pronaći raspored čvorova grafa koji će minimizovati ciljnu (object) funkciju. Kao kriterijume možemo na primer uzeti minimalno vreme izvršavanja, udaljenost izmedju elemenata i slično. Primenom MinLA možemo da postignemo efikasnije rasporedjivanje resursa, smanjenje vremena izvršavanja i tako dalje. Veoma je korisna i u oblastima kao što su transport robe i proizvodnja dobara.

Problem se sastoji iz pronalaženja najbolje permutacije čvorova koja minimizuje ukupnu dužinu zbira svih grana. Dužinu jedne grane računamo kao apsolutnu razliku pozicija njenih čvorova u permutaciji.

$$c(\pi, G) = \sum_{\{u,v\} \in E} |\pi(u) - \pi(v)|$$

Cenu rasporeda možemo računati kao na slici:



Na levoj slici je graf G kako bi ga normalno crtali, dok je na desnoj slici isti taj graf samo su čvorovi povezani linearno. Presek uspravnih linija i grana daje odredjeni broj

preseka. Suma tih brojeva nam daje meru kojom odredjujemo cenu trenutne permutacije. Cilj nam je da minimizujemo cenu, tj broj preseka.

Za rešavanje ovog problema se koriste različiti algoritmi kao što su simulirano kaljenje, VNS, genetski algoritmi ili druge heurističke metode.

3 Rešavanje problema

U ovom radu ćemo koristiti simulirano kaljenje, VNS, genetske algoritme, tabu pretragu, kao i algoritam grube sile i pohlepni algoritam kako bi uporedili rezultate.

3.1 S-Metaheuristike

S-metaheuristike (Single solution metaheuristics) pronalaze pojedinačno optimalno rešenje.

3.1.1 Simulirano kaljenje

3.1.1.1 Uopšteno o metodi

Simulirano kaljenje koristi analogiju kaljenja metala tako što postepeno menja inicijalno rešenje koje varira u skladu sa metodom hladjenja i grejanja metala. Početno rešenje ovim pristupom može da "izadje" iz lokalnog minimuma ili maksimuma i pretražuje širi prostor kako bi potencijalno našao globalni minimum ili maksimum.

U svakoj iteraciji se porede staro rešenje i novo rešenje i uvek se prihvata bolje rešenje kao i deo rešenja koja su gora od prethodnog u nadi da će se izaći iz lokalnog optimuma. Verovatnoća prihvatanja lošijeg rešenja može biti promenljiva ili nepromenljiva. U analogiji sa kaljenjem metala bi to predstavljalo temperaturu. U ovom radu smo odlučili da će se taj parametar smanjivati sa porastom iteracija, odnosno:

$p = \frac{1}{i}$, , gde je i broj iteracija u petlji u kodu.

Za osnovu se koristi klasična lokalna pretraga. Metode modifikacije trenutnog rešenja koje smo implementirali su:

- **Swap:** zamena mesta dva nasumično izabrana čvora u rasporedu
- **Inverse:** invertovanje odabranog podniza elemenata u trenutnom rešenju, izbor indeksa je nasumičan
- **Scramble**: mešanje redosleda čvorova odredjenog podniza trenutnog rešenja korišćenjem *random* Python modula.

Vredi napomenuti da smo testirali i metod lokalne pretrage u kojem, nakon što se utvrdi da novonastalo rešenje nije bolje od trenutnog, pozivamo u pomoc k puta (k zavisi od ukupnog broja iteracija lokalne pretrage) klasicni algoritam generisanja sledeće permutacije, u nadi da se bolje rešenje nalazi hronološki blizu. Ovaj metod daje neznatno bolje rezultate od obične lokalne pretrage. Ovim se vraćamo na *simulirano kaljenje*.

3.1.1.2 Ideja kroz korake

- Inicijalizacija promenljivih koje će čuvati rešenje (*solution*), najbolje rešenje do sada (*best_soluton*) , iteraciju tog najboljeg rešenja (*best_i*) kao i cenu najboljeg rešenja (*best_value*).

- Iteriramo kroz petlju
    - Pravimo novo rešenje funkcijom koju prosledjujemo kao parametar algoritmu koja pravi manju promenu nad trenutnim rešenjem. Ovo je posebno pogodno jer možemo da testiramo kasnije različite verzije ovog algoritma.
    - Ukoliko je nova vrednost cene generisanog rešenja bolja od stare cene onda se prihvata ta nova cena, a ukoliko je lošija onda se generiše nasumičan broj od 0 do 1 i ukoliko je manji od parametra p se prihvata i ovo lošije rešenje
- Vraćamo najbolje rešenje, cenu tog rešenja kao i redni broj iteracije u kojoj smo našli to najbolje rešenje kako bi mogli da uporedimo sa drugim algoritmima

Dobijeno rešenje može ali i ne mora biti globalno najbolje rešenje problema.

3.1.2 Variable neighborhood search

3.1.2.1 Uopšteno o metodi

Variable neighborhood search (VHS) je metoda koja koristi različite okoline (neighborhoods) za pretragu rešenja. Počinje se sa trenutnim rešenjem i primenjuje se lokalna pretraga nad tom okolinom da bi se našao lokalni minimum ili maksimum. Ako se nadje neki minimum ili maksimum onda algoritam prelazi na neku drugu okolinu i nastavlja se pretraga. Ovako se izbegava zaglavljivanje u lokalnom minimumu i maksimumu i moguće je naći najbolje rešenje.

Za svaku iteraciju razmatramo različite okoline trenutnog rešenja. Ovo radimo *shaking* funkcijom koju smo implementirali na dva načina kako bi mogli da uporedimo.

- ***Swap:*** k puta biramo 2 indeksa koja menjamo medjusobno

- ***Inverse***: invertuje odabrani podniz elemenata u trenutnom rešenju

3.1.2.2 Ideja kroz korake

- Inicijalizacija promenljive koja predstavlja rešenje (solution) i promenljive kojoj pamtimo najbolju iteraciju
- Iteriramo kroz petlju onoliko puta koliko je navedeno unapred promenljivom (num_iters)
    - Novo rešenje se pravi uz pomoć funkcije kora vrši shaking, a zatim se traži lokalni optimum tog rešenja
    - Ukoliko je ovo rešenje bolje od prethodnog onda se pamti to novo resenje kao najbolje
    - Takodje ukoliko nam je cena novog rešenja ista kao i prethodnog onda postoji mogućnost da se ipak zapamti to novo rešenje tako što generišemo broj od 0 do 1 i uporedjujemo ga sa unapred zadatim parametrom *(move_prob)* i ukoliko se ispuni uslov onda menjamo najbolje rešenje
- Vraćamo rešenje, najbolje rešenje i kada je nadjeno to najbolje rešenje, tj u kojoj iteraciji petlje.

3.2 Genetsko programiranje

3.2.1 Uopšteno o metodi

Genetsko programiranje je algoritam koji je inspirisan genetikom i procesima prirodne selekcije. Koristi se u mnogim oblastima, uključujući inženjering, optimizaciju, mašinsko učenje, umetnu inteligenciju, finansije i drugo.

Za razliku od simuliranog kaljenja, ovde radimo sa populacijom koja se sastoji iz više pojedinaca. Cilj nam je da populacija evoluira ka što boljem rešenju.

Populacija se obično generiše nasumično ili prema nekim pravilima koja odgovaraju konkretnom problemu. Kod MinLA problema generišemo populaciju kao više pojedinaca od kojih svaki čini permutaciju liste brojeva od 1 do broja čvorova koji navedemo unapred. Pojedinci se medjusobno razliku tako što se nasumično pri generisanju pojedinaca rasporedjuju brojevi u listu.

Fitnes funkcija predstavlja funkciju kojom računamo cenu nekog pojedinačnog rešenja. U svakoj novoj generaciji ponovo računamo fitnes za svakog pojedinca. Kod problema MinLa smo definisali fitnes funkciju kao funkciju koja računa ukupno rastojanje svih čvorova izmedju kojih postoji grana u grafu. Ukoliko postoji grana onda se na ukupan zbir dodaje rastojanje izmedju indeksa ta dva čvora u listi.

Algoritmu se prosledjuje argument koji predstavlja broj generacija koji će biti generisan. Svaka nova iteracija pravi novu generaciju koja je generisana od dela stare populacije koji se pokazao da ima najbolji fitnes.

Bitan korak u genetskom algoritmu je selekcija kojom se biraju pojedinci za reprodukciju. Postoje razne tehnike selekcije, a u ovom radu ćemo razmatrati četiri tehnike:

(1) **Ruletska selekcija:** Svaka jedinka ima šansu da bude odabrana u zavisnosti od njenog fitnesa. Slično kao kod igre rulet samo što su verovatnoće nejednake. Nakon odredjivanja verovatnoća se nasumično bira tačka na ruletu i jedinke čiji segmenti obuhvataju tu tačku se biraju.
(2) **Turnirska selekcija:** Nasumično biramo neki broj jedinki iz populacije koji je odredjen unapred kao argument funkcije. Od tih jedinki biramo onu koja ima najbolji fitnes.
(3) **Rank selekcija:** Selekcija ne zavisi direktno od fitnesa već od ranka fitnesa. Populacija se rangira prema fitnesu i veći rang dobijaju jedinke sa većim fitnesom.
(4) Linearna rank selekcija: Slično kao i obična rank selekcija sa izuzetkom što se primenjuje linearna transformacija na rangove kako bi i lošije jedinke dobile mogućnost da budu izabrane.

Pored selekcije bitan korak je ukrštanje jedinki. Kod ukrštanja se razmenjuje deo koda izmedju dva roditelja kako bi se stvorio novi kod, tj dete. Ovako se stvaraju nove jedinke sa takodje dobrim fitnesom. Kod problema MinLA radimo sa permutacijama tako da nije trivijalno kombinovanje dve permutacije kako bi se dobila nova koja ispunjava uslov da su svaka dva čvora različiti. Koristićemo dve funkcije:

(1) **Ordered Crossover (OX):** Redom kopiramo segment jednog roditelja, a zatim segment drugog roditelja bez mešanja
(2) **Partially Mapped Crossover (PMX):** Malo drugačiji jer dozvoljava zamene izmedju roditelja. Elementi se dodatno razmenjuju izmedju potomka i drugog roditelja sve dok se ne uklone svi duplikati.

Nakon ukrštanja se dešava mutacija jedinki u populaciji. Postoji parametar koji se unapred zadaje koji predstavlja verovatnoću da će deo jedinke da mutira. U ovom konkretnom problemu mutacije nisu sasvim trivijalne jer se mora očuvati integritet jedinke, odnosno svaka jedinka i dalje mora imati jedinstvene čvorove. Kako bi se ovo očuvalo mutacije se dešavaju tako što se dva dela koda obično menjaju, a ne samo jedan. Možemo razmatrati tri tehnike mutacije:

(1) **Swap:** Menjamo dva nasumično izabrana čvora u redosledu elemenata.
(2) **Inverse:** Kod ove tehnike menjamo skup elemenata unutar dva nasumično izabrana indeksa tako što ih samo obrnemo u redosledu.
(3) **Scramble:** Isto kao kod inverse tehnike samo što se elementi ne obrću po redu nego se svi elementi nasumično ispremeštaju.

3.1.1.2 Ideja kroz korake

- Generiše se populacija nasumično pomoću klase Individual kojoj prosledjujemo broj čvorova i graf
- Iterira se kroz petlju koja kao uslov zaustavljanja ima broj generacija koji je unapred zadat
    - Populacija se sortira i prvi broj najboljih se odmah dodeljuje novoj populaciji na prva mesta. Taj broj se zadaje unapred kao *elitism_size*
    - Iterira se kroz petlju za ostala mesta u populaciji
        - Biraju se dva roditelja procesom selekcije
        - Dobija se nova jedinka od ta dva roditelja
        - Ta jedinka može da mutira
        - Računa se fitnes novih jedinki

Na kraju algoritma se vraća najbolji pojedinac iz populacije.

3.3 Tabu Search

3.3.1 Uopšteno o metodi

Tabu Search je metaheuristički optimizacioni algoritam koji se koristi za rešavanje različitih optimizacionih problema. Ovaj algoritam je sličan algoritmu lokalne pretrage sa tim što ima način za izbegavanje zadržavanja u lokalnim minimumima i maksimuma tako što predje na neko lošije rešenje kada primeti da rešenja konvergiraju.

Ime Tabu je dobio po tome što pamti nedavno posećena rešenja u listi, pa u analogiji sa tim rešenjima koja ne smeju da se “diraju” je dobio ime. Time što je zabranjeno posećivanje istih rešenja smo dobili na tome da pretražujemo nove regione umesto da se vraćamo na već pretražene regione čime dobijamo i na vremenu. U suštini kombinuje se intenzifikacija i diversifikacija kako bi balansirali između lokalne pretrage i pretrage šireg dela prostora. Algoritam se obično zaustavlja ako se ispuni neki od kriterijuma zaustavljanja kao što su odredjeni broj iteracija ili ako funkcija cilja dostigne neku unapred zadatu vrednost. U ovom radu je kriterijum bio unapred zadat broj iteracija jer nam je cilj bio da vidimo koliko je iteracija bilo potrebno kojem algoritmu.

Što se tiče metode za generisanje modifikacija trenutnog rešenja (odnosno "komšiluka"), implementirali smo:

- **Swap Neighbors:** menjamo mesta svim susednim elementima u resenju
- **Swap:** pravimo sve moguće kombinacije zamena mesta u rešenju
- **Inverse:** inverzovanje nasumičnog dela niza bazirano na trenutnom indeksu
- **Inverve Complete**: pravljenje svih kombinacija inverzovanja(podniz svake dužine za svaku početnu poziciju)
- **Scramble**: nasumično mešanje pozicija elemenata podniza rešenja
- **Scramble Continuous**: metod isti kao prethodni, medjutim ovde umesto n različitih modifikacija nad početnim rešenjem, i-to rešenje dobijamo modifikacijom (i-1)-og, čime na kraju dobijamo jako raznovrsan skup u odnosu na intuitivnije metode

3.3.2 Ideja kroz korake

- Inicijalizacija početnih parametara koji predstavljaju trenutno rešenje *(current_solution)*, trenutnu vrednost *(current_value)* kao i najboljeg rešenja *(best_solution)* i najbolje vrednosti *(best_value)* koji samo predstavljaju kopije trenutnih vrednosti. Takodje inicijalizujemo i listu *(tabu_list)*koja će predstavljati sve čvorove koji su već obidjeni. Pored svega ovoga pratimo i broj iteracija, ali to nije bitno za algoritam već se prati radi upoređivanja rezultata.

- Generiše se skup rešenja (*neighbors*) koji se nalazi u "komšiliku" funkcijom koja se unapred zadaje kao parametar algoritma *(neigbor_gen_func)*
- Iz liste okolnih rešenja se traži ono koje nije u tabu listi, a ako su sva rešenja u tabu listi onda uzimamo najbolje
- Ažurira se tabu lista tako što se dodaje novo izabrano rešenje
- Ukoliko je rešenje bolje od prethodnog najboljeg rešenja onda se ažurira najbolje rešenje

Na kraju se vraćaju najbolje rešenje, njegova vrednost i iteracija petlje u kojoj je rešenje pronadjeno.

4 Analiza rezultata

Sledeće tabele predstavljaju najbolja rešenja koja daju algoritmi kao i vremena za koja se izvršavaju. S obzirom na broj algoritama je tabela podeljena na tri tabele radi veće preglednosti. Žutom bojom su obojeni rezultati koji su dobijeni primenom algoritama na datasetove koji su pronadjeni na internetu kako bi se što bolje i objektivnije uporedili rezultati. Kolona *Dim* predstavlja dimenziju grafa koji se rešava.

	Dim	brute_force	brute_force_time	greedy	greedy_time	local_search	local_search_time
0	5.0	13.0	0.0	17.0	0.0	13.0	0.01
1	8.0	21.0	0.58	39.0	0.0	21.0	0.01
2	10.0	76.0	79.03	112.0	0.0	76.0	0.02
3	11.0	59.0	552.97	113.0	0.0	59.0	0.02
4	15.0	nan	nan	254.0	0.0	125.0	0.04
5	20.0	nan	nan	805.0	0.0	508.0	0.04
6	33.0	nan	nan	3110.0	0.0	2139.0	0.18
7	38.0	nan	nan	4992.0	0.0	3638.0	0.11
8	50.0	nan	nan	11293.0	0.0	8681.0	0.45
9	70.0	nan	nan	31606.0	0.0	26034.0	0.55
10	12.0	241.0	10351.71	279.0	0.0	241.0	0.02
11	16.0	nan	nan	660.0	0.0	560.0	0.03
12	17.0	nan	nan	791.0	0.0	667.0	0.03
13	23.0	nan	nan	1894.0	0.0	1582.0	0.05
14	30.0	nan	nan	4052.0	0.0	3435.0	0.1

Tabela 1: Rezultat i vreme izvršavanja najboljih rezultata za brute-force, pohlepni algoritam i lokalnu pretragu

	Dim	brute_force	brute_force_time	greedy	greedy_time	local_search	local_search_time
0	5.0	13.0	0.0	17.0	0.0	13.0	0.01
1	8.0	21.0	0.58	39.0	0.0	21.0	0.01
2	10.0	76.0	79.03	112.0	0.0	77.33	0.02
3	11.0	59.0	552.97	113.0	0.0	60.33	0.02
4	15.0	nan	nan	254.0	0.0	137.0	0.03
5	20.0	nan	nan	805.0	0.0	525.67	0.04
6	33.0	nan	nan	3110.0	0.0	2200.33	0.15
7	38.0	nan	nan	4992.0	0.0	3693.33	0.12
8	50.0	nan	nan	11293.0	0.0	8852.67	0.43
9	70.0	nan	nan	31606.0	0.0	26503.0	0.55
10	12.0	241.0	10351.71	279.0	0.0	241.0	1.38
11	16.0	nan	nan	660.0	0.0	560.0	0.03
12	17.0	nan	nan	791.0	0.0	669.33	0.03
13	23.0	nan	nan	1894.0	0.0	1590.0	0.05
14	30.0	nan	nan	4052.0	0.0	3446.67	0.1

Tabela 2: Rezultat i vreme izvršavanja prosečnih rezultata za brute-force, pohlepni algoritam i lokalnu pretragu

Vidimo iz tabele iznad da se algoritam grube sile pokazuje dobro na malim instancama problema, ali se već za instance veće od 10 izvršava znatno duže i iako daje dobar rezultat u vidu rešenja, zbog vremena izvršavanje je praktično neupotrebljiv.

Kod pohlepnog algoritma je obrnuta situacija u odnosu na algoritam grube sile. Ako bismo gledali samo vreme izvršavanja ovo bi bio najbolji algoritam, ali ako pogledamo i rešenje koje algoritam dobija, ono je skoro uvek neoptimalno. Ovaj algoritam nam služi prvenstveno za poredjenje sa drugim metodama.

Ako pogledamo obe lokalne pretrage (Tabele 1, 2, 3, 4), videćemo da rade jako slično. One prave kompromis u vidu brzine i dobrih rešenja problema. Pri malim

dimenzijama gotovo uvek nalaze optimalno rešenje što je i za očekivati, ali se pri većim dimenzijama retko kada ubode globalni optimum.

	Dim	local_search_perm	local_search_perm_time	sim_annealing	sim_annealing_time	vns	vns_time
0	5.0	13.0	0.05	13.0	0.01	13.0	0.1
1	8.0	21.0	0.06	21.0	0.01	21.0	0.26
2	10.0	76.0	0.11	76.0	0.02	76.0	0.42
3	11.0	59.0	0.12	61.0	0.02	59.0	0.4
4	15.0	125.0	0.18	128.0	0.03	125.0	0.84
5	20.0	502.0	0.3	497.0	0.05	498.0	3.5
6	33.0	2125.0	0.7	2185.0	0.19	2108.0	16.71
7	38.0	3635.0	0.67	3673.0	0.11	3646.0	13.58
8	50.0	8633.0	2.69	8688.0	0.43	8561.0	185.92
9	70.0	25591.0	2.86	26230.0	0.53	25064.0	316.9
10	12.0	241.0	0.1	241.0	0.01	241.0	0.35
11	16.0	560.0	0.16	563.0	0.03	560.0	0.88
12	17.0	667.0	0.2	667.0	0.03	667.0	1.31
13	23.0	1594.0	0.42	1584.0	0.05	1582.0	7.13
14	30.0	3422.0	0.55	3419.0	0.1	3416.0	9.8

Tabela 3: Rezultat i vreme najboljih izvršavanja za modifikovanu lokalnu pretragu, simulirano kaljenje i vns

	Dim	local_search_perm	local_search_perm_time	sim_annealing	sim_annealing_time	vns	vns_time
0	5.0	13.0	0.06	13.0	0.01	13.0	0.33
1	8.0	21.0	0.09	21.0	0.01	21.0	0.6
2	10.0	76.0	0.11	77.33	0.02	76.0	1.05
3	11.0	61.67	0.13	61.0	0.02	59.17	1.03
4	15.0	138.33	0.19	135.33	0.03	126.25	2.37
5	20.0	513.0	0.34	507.0	0.06	512.42	5.97
6	33.0	2161.0	0.8	2213.33	0.17	2164.92	24.17
7	38.0	3687.67	0.61	3771.0	0.11	3724.42	21.44
8	50.0	8699.33	2.7	8808.0	0.44	8669.17	101.39
9	70.0	26189.33	3.05	26583.33	0.54	25518.58	180.08
10	12.0	241.0	0.1	241.0	0.01	241.0	1.01
11	16.0	562.0	0.17	565.67	0.03	560.67	2.23
12	17.0	675.0	0.19	671.67	0.03	667.25	2.81
13	23.0	1603.67	0.38	1596.0	0.06	1590.08	6.49
14	30.0	3472.0	0.57	3435.67	0.11	3427.08	15.45

Tabela 4: Rezultat i vreme prosečnih izvršavanja za modifikovanu lokalnu pretragu, simulirano kaljenje i vns

Simulirano kaljenje u proseku daje slične rezultate kao i lokalne pretrage sa nešto boljim vremenima izvršavanja.

Od svih algoritama vidimo da VNS daje najbolje rešenje problema što je i bilo očekivanje jer pretražuje različite delove domena rešenja. Loša stvar u vezi VNS-a je vreme izvršavanja koje je definitivno najgore od svih algoritama. Testirano je dvanaest različitih kombinacija tako da je vreme izvršavanja veliko i zbog toga.

	Dim	genetic_alg	genetic_alg_time	tabu	tabu_time	taloc_combo	taloc_combo_time
0	5.0	13.0	0.14	13.0	0.02	13.0	0.34
1	8.0	21.0	0.13	21.0	0.06	21.0	0.49
2	10.0	76.0	0.2	76.0	0.28	76.0	0.93
3	11.0	59.0	0.22	61.0	0.27	59.0	0.86
4	15.0	125.0	0.34	125.0	2.07	125.0	1.31
5	20.0	497.0	0.56	502.0	6.92	497.0	3.33
6	33.0	2119.0	1.14	2082.0	62.09	2055.0	82.29
7	38.0	3555.0	0.82	3577.0	59.22	3506.0	11.25
8	50.0	8218.0	1.44	8244.0	204.91	8113.0	21.99
9	70.0	24483.0	3.24	25058.0	1567.18	24492.0	1301.84
10	12.0	241.0	0.15	241.0	0.15	241.0	0.83
11	16.0	560.0	0.23	561.0	2.14	560.0	1.55
12	17.0	667.0	0.25	667.0	1.38	667.0	1.82
13	23.0	1581.0	0.42	1584.0	10.01	1581.0	3.79
14	30.0	3416.0	0.71	3425.0	31.68	3416.0	7.67

Tabela 5: Rezultat i vreme najboljih izvršavanja za gentski algoritam, tabu pretragu i modifikovani tabu

	Dim	genetic_alg	genetic_alg_time	tabu	tabu_time	taloc_combo	taloc_combo_time
0	5.0	13.0	0.24	13.29	0.07	13.0	0.51
1	8.0	21.0	0.25	21.29	0.28	21.0	0.83
2	10.0	76.67	0.31	78.14	0.77	76.0	1.45
3	11.0	60.11	0.3	63.57	0.86	59.2	1.71
4	15.0	128.28	0.37	138.43	2.36	126.07	3.22
5	20.0	510.28	0.55	530.86	9.41	505.43	10.14
6	33.0	2136.89	1.2	2224.57	73.68	2105.2	65.24
7	38.0	3633.94	0.85	3791.71	75.19	3565.38	45.81
8	50.0	8525.78	1.53	8733.0	259.28	8278.7	210.99
9	70.0	25226.0	3.38	26421.43	1266.5	24863.14	943.19
10	12.0	241.0	0.2	245.86	0.94	241.0	1.47
11	16.0	561.22	0.28	566.0	2.93	560.77	3.54
12	17.0	668.39	0.3	679.29	3.77	667.0	4.3
13	23.0	1589.0	0.48	1620.43	13.19	1588.03	11.97
14	30.0	3440.67	0.77	3547.0	40.88	3422.23	30.81

Tabela 6: Rezultat i vreme prosečnih izvršavanja za gentski algoritam, tabu pretragu i modifikovani tabu

Metod koji se u proseku najbolje pokazao je bio genetski algoritam. Vidimo da su rezultati u proseku veoma zadovoljavajući, a vreme izvršavanja je čak i pri velikim dimenzijama skoro zanemarljivo. Umesto kompleksnih i velikih promena nad rešenjem u svakoj iteraciji, što postaje vremenski zahtevno sa većim grafovima, genetski algoritam koristi brže funkcije koje rade jednostavne promene nad već dobrim rešenjima, kako bi ih napravile još boljim. Pri testiranju smo uvideli da turnirska selekcija i rank (*rank_selection_linear_ranking*) daju najbolje rezultate i takodje smo uvideli da zapravo selekcija igra najveću ulogu u kvalitetu rešenja.

Obična Tabu pretraga daje optimalne rezultate za manje i srednje grafove ali pri većim dimenzijama grafa ipak ne uspeva da nadje najoptimalnije rešenje, što je i očekivano s obzirom na to da radi slično kao i lokalna pretraga. Za veće grafove generisanje “komšija” rešenja postaje problematično, bez obzira koju od mnogih testiranih tehnika koritsimo, rešenja postaju prevelika da bi se dovoljan broj rešenja za diversifikovan komšiluk generisao efikasno. Ovo u većim dimenzijama utiče na vreme izvršavanja. Najbolji odnos kvaliteta rešenja i vremena izvršavanja daju metode koje prave mali kompromis na oba fronta - *inverse* i *scramble.*

Taloc (Tabu i S-metaheuristike) skoro uvek uspe da nadje bolje rešenje nego sve ostale testirane metode. Do odredjene dimenzije se može reći da je ovo najbolja metoda i treba se koristiti za ovaj problem. Medjutim za veće dimenzije genetski algortitam ostaje najbolji izbor.

5 Zaključak

Iz prethodne analize vidimo da su se genetski algoritmi pokazali kao najbolji metod za rešavanje MinLA problema u pogledu kompromisa izmedju kvaliteta rešenja i vremena izračunavanja. Za neke manje dimenzije grafova se mogu koristiti i drugi algoritmi poput VNS-a kako bi se postigla veća tačnost ukoliko je to potrebno. Ukoliko je bitno dobiti neko rešenje koje je suboptimalno za veće dimenzije, onda se mogu koristiti i drugi algoritmi pored genetskih kao što su na primer simulirano kaljenje.

6 Literatura

1. https://github.com/MATF-RI/Materijali-sa-vezbi
2. Prof. Dr. Gerhard Reinelt, Prof. Dr. Dr. h. c. Hans Georg Bock, Contributions to the Minimum Linear Arrangement Problem (https://www.semanticscholar.org/paper/Contributions-to-the-Minimum-Linear-Arrangement-Seitz/a06161f233005595d7bb0fb11a800ca9c5f75e93)
3. Fred Glover, Tabu Search – A Tutorial (https://www.researchgate.net/publication/242527226_Tabu_Search_A_Tutorial)
4. Michael Zündorf, Minimum Linear Arrangement revisited (https://scale.iti.kit.edu/_media/resources/theses/ma_zuendorf.pdf)
5. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)
6. https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)#Rank_Selection
7. https://data.mendeley.com/datasets/n5f3zcdg92/2