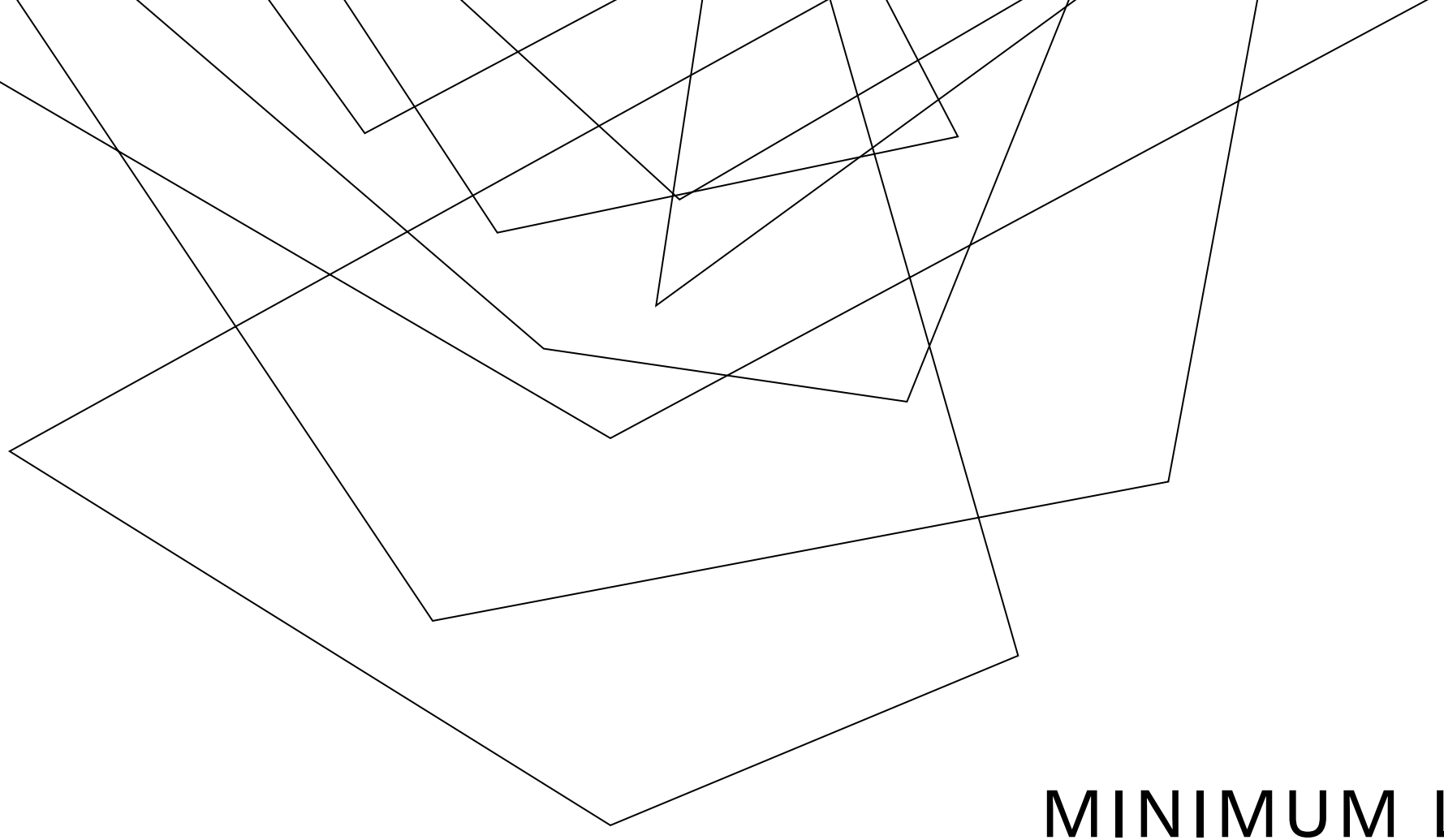


MINIMUM LINEAR ARRANGEMENT

Milan Brcin 247/2019

Milica Obradovic 83/2019

OPIS PROBLEMA

U ovom radu bavimo se optimizacijom i rešavanjem problema Minimum Linear Arrangement (skraceno MinLA) koristeći razne metode.

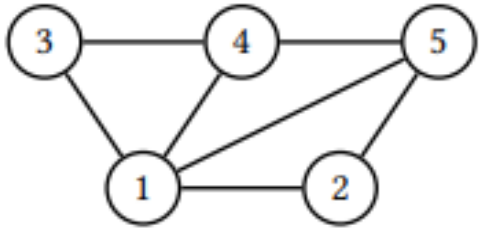
MinLA spada u poznat NP-težak problem.

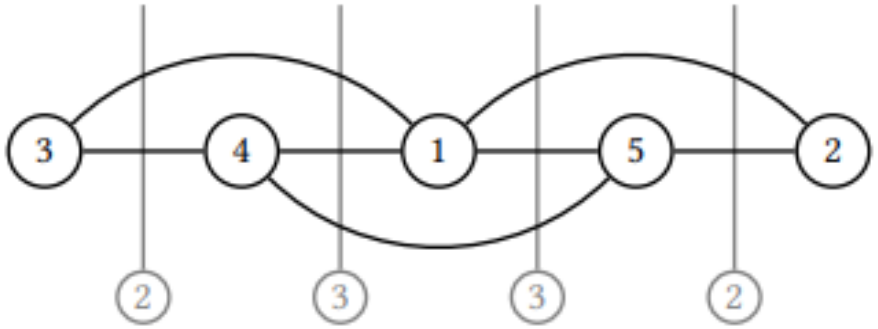
MinLA problem se može definisati na sledeći način:

Ukoliko imamo povezan neusmeren graf G(V, E) , treba za njega naći permutaciju (arrangement) njegovih čvorova za koju važi da ona minimizuje zbir dužina svih grana.

Dužinu grane (A,B), pri čemu su A i B čvorovi grafa, računamo kao absolutnu razliku pozicija čvora A i čvora B u prethodno navedenoj permutaciji. Formalnije:

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



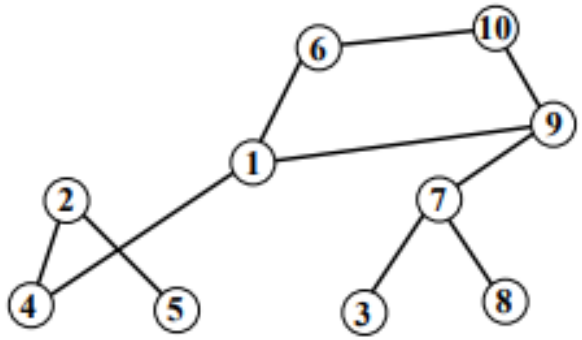


Alternativni nacin gledanja na problem (primer i ilustracija iz "Minimum Linear Arrangement Revisited" od Michael Zündorf-a).

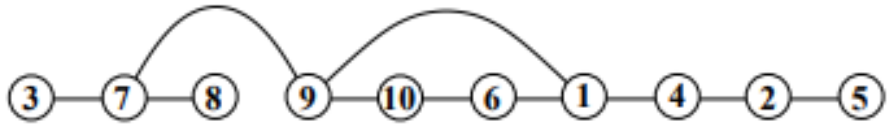
Cenu rasporeda mozemo racunati kao na desnoj slici, prvo redjajuci cvoroce kao u permutaciji, zatim spajajuci cvorove bazirano na granama koje postoje, onda izmedju svaka dva cvora povucemo uspravnu liniju i brojimo koliko preseka je napravila sa granama, zatim saberemo sve preseke i to je cena resenja. U primeru gore: **(2+3+3+2=10)**
Ovo se moze racunati drugacije, za svaku postojecu granu racunamo apsolutnu razliku indeksa njenih cvorova u permutaciji:
(1,2) - 2
(1,5) - 1
(1,4) - 1
(1,3) - 2
(2,5) - 1
(3,4) - 1
(4,5) - 2

2+1+1+2+1+1+2=10

Jos jedan primer grafa i jednog njegovog resenja. Uslovi koje sam graf treba da ispunjava su neusmerenost i povezanost (svaki cvor mora biti kraj barem jedne grane, odnosno povezan sa barem jednim drugim cvorom).

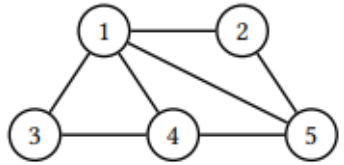


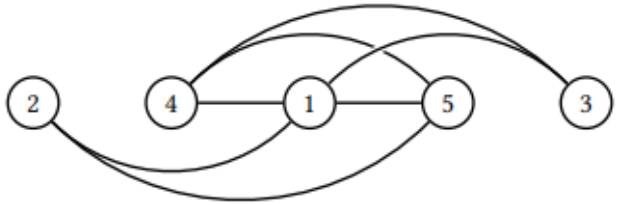
(a) Example graph G.

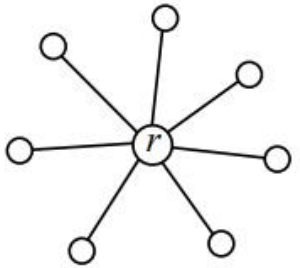


(b) MinLA presentation of the example graph G.

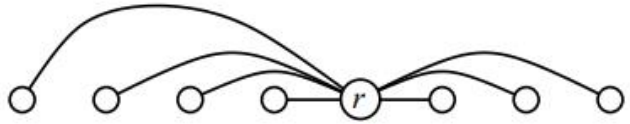
Primer preuzet iz "Contributions to the Minimum Linear Arrangement Problem" od Prof. Dr. Gerhard Reinelt i Prof. Dr. Dr. h. c. Hans Georg Bock



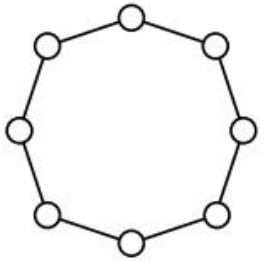




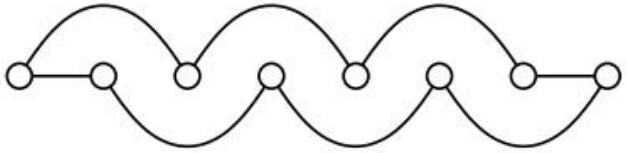
(a) Star corresponding to (1.2).



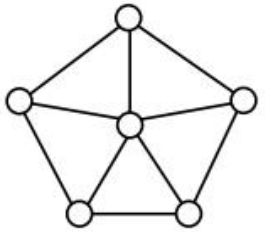
(b) Star in its MinLA presentation.



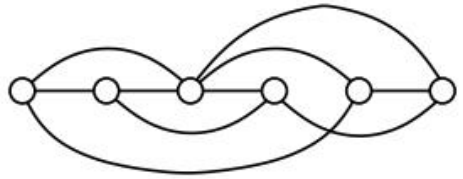
(c) Cycle corresponding to (1.4).



(d) Cycle in its MinLA presentation.



(e) Wheel corresponding to (1.5).



(f) Wheel in its MinLA presentation.

Primeri preuzeti iz "Contributions to the Minimum Linear Arrangement Problem" od Prof. Dr. Gerhard Reinelt i Prof. Dr. Dr. h. c. Hans Georg Bock
Kao i "Minimum Linear Arrangement Revisited" od Michael Zündorf-a

METODE RESAVANJE KOJE SMO KORISTILI

S-METAHEURISTIKE

Različite vrste pretraga koje koriste različite metode modifikacija i prelazaka sa jedne iteracije na drugu, različite vrste metode "simulirano kaljenje", VNS (variable neighbourhood search) sa svim kombinacijama prethodno navedenih lokalnih pretraga i raznim "shaking" metodama

TABU PRETRAGA

Pretraga prostora resenja koristeci pomoc random faktora i tabu liste koja nas sprecava da ponavljamo vec nedavno vidjena resenja kako bi se osigurao siri spektrum pretrage

GENETSKI ALGORITMI

Primena genetskih algoritama sa svim kombinacijama selekcija, mutacijama koje obuhvataju sve metode modifikacije korišćene u lokalnim pretragama i raznim vrstama "crossover" koraka

BRUTE-FORCE I POHLEPNI ALG.

Provera kvaliteta svako moguće permutacije za dobijanje garantovano najboljeg rešenja kao i za aproksimaciju kvaliteta ostalih eksperimantalnih metoda.

Pohlepni algoritmi za demonstraciju njihove neadekvatnosti kod ovakog problema i poredjenje rezultata

GRUBA SILA

Metod grube sile generalno podrazumeva generisanje svih resenja ili prolazak kroz sva resenja i pronalaženje najboljeg medju njima. Ne koristi ikakvu kompleksnu matematiku ili algoritmiku.

U nasem slucaju, ovaj metod je podrazumevao generisanje svake moguće permutacije čvorova grafa G, zatim računanje ukupne dužine svih grana grafa za svaku datu permutaciju, pri čemu se pamti najbolji rezultat i rešenje koje daje taj rezultat.

Ukoliko imamo n čvorova u grafu, vremenska kompleksnost metode bice O(n! * n^2) , zato sto n! mogućih permutacija postoji, a racunanje vrednosti svake od njih traje O(n^2).

O(n!) se moze porediti sa O(2^n) vremenskom kompleksnoscu, stavise, smatra se i gorom , što ovaj metod cini potpuno neupotrebljivim za veće dimenzije problema MinLA.

POHLEPNI ALGORITAM

Pohlepni algoritmi se baziraju na ideji pronalazenja najboljeg lokalnog resenja u svakom koraku resavanja problema.

Za MinLA problem, ovo bi podrazumevalo sastavljanje permutacije korak po korak, gde u svakom koraku sledeci broj u permutaciji biramo tako da on dodaje najmanju tezinu na dosadasnje resenje.

Primer:

Ako imamo delimicno resenje [1, 0, 3] u grafu od 5 cvorova sa granama:

(0,2), (0,3), (0,4), (1,2), (1,4), (2,3)

onda ce sledeci broj biti 4, zato sto :

val(0,4) = 2

val(1,4) = 3

ciji je zbir 5

kada bi bio 2 onda:

val(1,2) = 3

val(0,2) = 2

val(2,3) = 1

ciji je zbir 6

([1 u , 3, 9,0, 4,[1, 3, 9, 0, 4,

LOKALNA PRETRAGA

Algoritam lokalne pretrage, kao sto ime nakazuje, pretrazuje resenja u lokalu kako bi nasla najbolje.

Koristeci razne tehnike, u zavisnosti od prirode problema, modifikujemo pocetno nasumicno resenje kako bi dobili novo. Zatim vrsimo evaluaciju tog novog resenja kako bi videli da li je bolje od prethodnog, u kom slucaju nastavljamo da pretrazujemo njegove modifikacije, u suprotnom u sledecoj iteraciji pravimo novu modifikaciju od istog resenja.

Najbolji rezultat pamtimo i proveravamo u svakoj iteraciji.

U MinLA problemu lokalna pretraga se implementira tako sto modifikujemo pocetnu permutaciju nekom tehnikom (swap, inverzovanje, mesanje itd), zatim izracunamo njenu vrednost I poredimo je sa prethodnom.

Algoritam ponavlja prethodno do kraja broja iteracija, isteka vremena ili pronalazeja dovoljno dobrog (nama zadovoljavajuceg) resenja.

LOKALNA PRETRAGA

Inicijalizacija --->

Modifikacija trenutnog resenja --->

Provera da li je novo bolje od trenutnog--->

Azuriranje do sad najboljer resenja--->

```python
def local_search(graph, random_solution, value, num_iters, change_func):
    solution = deepcopy(random_solution)
    best_solution = deepcopy(solution)
    best_value = value
    best_i = None

    for i in range(num_iters):
        #print(solution, value, i)
        new_solution = change_func(graph, solution)
        new_value = dg.calculate_total_edge_length(graph, new_solution)

        if new_value < value:
            value = new_value
            solution = deepcopy(new_solution)

            if new_value < best_value:
                best_i = i
                best_value = new_value
                best_solution = deepcopy(new_solution)

    return best_solution, best_value, best_i
```

MODIFIKOVANA LOKALNA PRETRAGA

Jenda modifikacija klasicne lokalne pretrage koja je nasa kreacija I koristi benefite cinjenice da je resenje ovog problema dato u vidu permutacije cvorova grafa.

Ideja je sledeca:

Lokalna pretraga radi normalno dok ne dodje do iteracije u kojoj je novonastalo resenje gore od prethodnog.

U tom slucaju, manji broj puta (u zavisnosti od ukupnog broja iteracija), generisemo hronoloski sledecu permutaciju, u nadi da cemo naci bolje resenje koje se ne razlikuje mnogo od trenutnog.

Ako ga nadjemo, nastavljamo sa njim –a ko ne, nastavljamo sa prethodnim.

Ovaj dodatak ima neprimetan uticaj na performansu algoritma, kao sto se vidi u rezultatima

SIMULIRANO KALJENJE

Slicno istoimenoj tehnici u obradi metala, ideja algoritma je da ponekad 'hladimo' resenje, odnosno postoji sansa da u nekoj iteraciji nastavimo da se bavimo resenjem gorim od prethodnog – u nadi da cemo ovom diverzifikacijom pretrage resenja kasnije naici na najbolje resenje.

Ovo se koristi kako se ne bi zaglavili u lokalnom maksimumu bez sanse da tragamo za drasticno drugacijim resenjima.

Ipak, ne zelimo da u poznom delu izvrsavanja algoritma mnogo menjamo prostor pretrage, zato se sansa za uzimanjem losijeg resenja smanjuje sa brojem iteracija (obicno koristeci neku funkciju od broja iteracija: 1/i, 1/i^2 itd.)

SIMULIRANO KALJENJE

Inicijalizacija --->

Modifikacija trenutnog resenja --->

Provera da li je novo bolje od trenutnog--->

Azuriranje do sad najboljer resenja--->

Potencijalno uzimamo gore resenje da bi diverzifikovali pretragu --->

```python
def simulated_annealing(graph, random_solution, value, num_iters, change_func):
    solution = deepcopy(random_solution)
    best_solution = deepcopy(solution)
    best_value = value
    best_i = None

    for i in range(1, num_iters + 1):
        #print(solution, value)
        new_solution = change_func(graph, solution)
        new_value = dg.calculate_total_edge_length(graph, new_solution)

        if new_value < value:
            value = new_value
            solution = deepcopy(new_solution)

            if new_value < best_value:
                best_i = i
                best_value = new_value
                best_solution = deepcopy(new_solution)

        #elif random.random() < 1 / (i**0.5):
        elif random.random() < 1 / i:
            #print('divs')
            value = new_value
            solution = deepcopy(new_solution)

    return best_solution, best_value, best_i
```

VNS

VNS (Variable Neighnourhood Search) je s-metaheuristika koja koristi shaking (pretresanje) metodu kako bi 'pomerilo pretragu u drugi komsiluk' i kako pretraga ne bi sve vreme bila izvrsavana u lokalu.

Ova shaking metoda se poziva nad pocetnim resenjem, nakon toga se nad tim rezultatom vrsi na prethodnim slajdovima opisana pretraga. Ako je resenje bolje od prethodnog nastavljamo da se bavimo njime, u suprotnom nastavljamo da modifikujemo prethodno resenje. Najbolje resenje se pamti.

Kod MinLA problema se shaking funkcija moze implementirati na vise nacina, neki od njih su:

SWAP – k puta izaberemo 2 random indeksa, zatim zamenimo pozicije brojevima na tim indeksima

INVERSE – nasumicno izaberemo indeks ind1, na njega dodamo k i dobijemo drugi indeks ind2, sve element od i1 do i2 obrnemo

Shaking funkcija mora imati smisla kada se radi sa permutacijama, ne mozemo dodavati nove elemente, niti ponavljati iste elemente, svi elementi moraju imati vrednost od 0 do m gde je n broja cvorova.

VNS

```python
def vns(graph, random_solution, value, num_iters, change_func, shaking_func, local_search_func, k_min, k_max, move_prob):
    solution = deepcopy(random_solution)
    best_i = None
    for i in range(num_iters):
        for k in range(k_min, k_max):
            #print('vns: ', solution, value, i)
            new_solution = shaking_func(graph, solution, k) #diversification

            new_value = dg.calculate_total_edge_length(graph, new_solution)

            #print('post shaking: ', new_solution, new_value, i)
            new_solution, new_value, _= local_search_func(graph, new_solution, total_edge_length, 10, change_func)

            if new_value < value or (new_value == value and random.random() < move_prob):
                if(new_value < value):
                    best_i = i

                value = new_value

                solution = deepcopy(new_solution)

    return solution, value, best_i
```

GENETSKI ALGORITMI

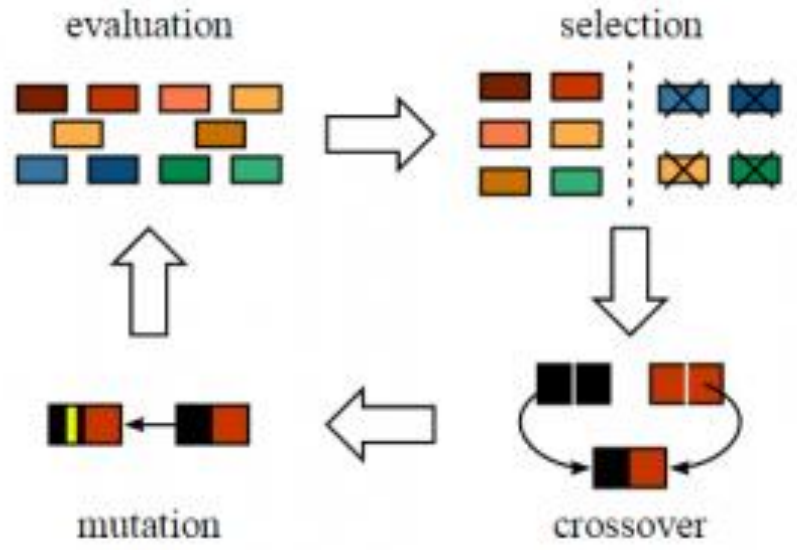
Genetski algoritmi spadaju u vrstu P-metaheuritika, sto znaci da radi sa populacijom kandidata za resenje u svakoj iteraciji, za razliku od prethodno spomenutih tehnika koje se u svakoj iteraciji bave samo jednim.

Inspirisani procesom naturalne selekcije, imamo inicijalnu populaciju "jedinki" (potencijalnih resenja problema) koju smo generisali koristeci random mehanike. Koraci algoritama u svakoj iteraciji su:

1. **Selekcija:** Biramo 2 roditelja koristeci neki mehanizam (turnirska selekcija, rulet selekcija itd.)
2. **Crossover:** U zavisnosti od tipa naseg genetskog algoritma, od prethodno izabranih roditelja pravimo jednu ili vise novih jedinki njihovim ukrstanjem, sto takodje moze biti uradjeno na mnogo nacina. Kada su permutacije u pitanju, nova permutacija mora postovati neke restrikcije, tako da mozemo koristiti PMX crossover, ordered crossover...
3. **Mutacija:** Kako nam nova resenja ne bi bila bazirana samo na inicijalnoj populaciji, cime rizikujemo limitiranje prostora pretrage i samim time pronalazak potencijalno najboljih resenja, novodobijene jednike cemo mutirati. Ovo je slicno change funkcijama kod lokalnih pretraga. Uzimajuci u obzir permutacije, mozemo koristiti swap, inverzovanje dela resenja, mesanje dela resenja...

Ako koristimo elitizam, u svakoj iteraciji cemo sortirati populaciju bazirano na fitnesu jedinki, I cuvacemo nekoliko najboljih.

GENETSKI ALGORITMI DIJAGRAM



GENETSKI ALGORITMI

```
population = [Individual(num_nodes, graph) for _ in range(population_size)]
new_population = population.copy()

for i in range(num_generations):
    population.sort(key=lambda x: x.fitness, reverse=True)
    new_population[:elitism_size] = population[:elitism_size]
    for j in range(elitism_size, population_size, 2):
        parent1_i = selection(population, tournament_size, forbidden=-2)
        parent2_i = selection(population, tournament_size, parent1_i)
        #disabled - causes bloated file size
        #print('gen:', i, 'iter:', j, 'parents:', parent1_i, parent2_i)
        if ordered_cross:
            new_population[j].code, i1, i2 = crossover(population[parent1_i], population[parent2_i])
            new_population[j+1].code, _, _ = crossover(population[parent2_i], population[parent1_i], i1, i2)
        else:
            new_population[j].code = crossover(population[parent1_i], population[parent2_i])
            new_population[j+1].code = crossover(population[parent1_i], population[parent2_i])

        mutation(new_population[j], mutation_prob)
        mutation(new_population[j+1], mutation_prob)

        new_population[j].fitness = new_population[j].calc_fitness(num_nodes, graph)
        new_population[j+1].fitness = new_population[j+1].calc_fitness(num_nodes, graph)

    population = new_population.copy()
return max(population, key=lambda x: x.fitness)
```

TABU PRETRAGA

U pitanju je metaheuristicki optimizacioni algoritam dizajniran da efikasno nadje resenja blizu optimalnog za kompleksne kombinatorne probleme. Koristi strukturu koja se zove "tabu lista" u kojoj cuva prethodno posecena resenja kako bi sprecila pretragu da im se vrati neko vreme. Ovo forsira algoritam da pretrazuje druge regione prostora resenja.

Nakon inicijalizacije random resenja, generisemo "komsiluk" (skup resenja bazirana na trenutnom, koristeci razne tehnike).

Poredimo trenutno resenje sa onima iz komsiluka – ako je najbolji iz komsiluka bolji ili ispunjava neki drugi kriterijum koji mu dajemo i ako nije vec u tabu listi, onda to postaje nase novo resenje i biva ubaceno u tabu listu - kako ne bi bilo ponovo uzimano u sledecim iteracijama (duzina ostanka resenja u tabu listi je proizvoljan parametar).

Ako su sva resenja vec u tabu listi, onda cemo uzeti najbolje od njih i nastaviti sa radom.

Nakon isteka zadatog vremena ili ako tabu lista dostigne odredjenu duzinu, izbacujemo iz nje elemen FIFO principom (kako bi svako resenje u listi provelo isti broj iteracija).

U svakoj iteraciji proveravamo da li smo dobili novo najbolje resenje i cuvamo ga.

TABU PRETRAGA

Inicijalizacija --->

Generisanje "komsiluka" (skupa resenja) --->

Uzimamo novo resenje koje nije tabu--->

Ako su sva tabu, uzimamo najbolje--->

Azuriramo tabu listu--->

Azuriramo najbolje resenje --->

```python
def tabu_search(graph, solution, max_iterations, neighbor_gen_func, tabu_tenure):
    num_nodes = len(graph)
    current_solution = deepcopy(solution)
    current_value = dg.calculate_total_edge_length(graph, current_solution)
    best_solution = current_solution.copy()
    best_value = current_value
    tabu_list = []

    best_i = -1

    for ind in range(max_iterations):
        neighbors = neighbor_gen_func(current_solution)
        neighbors.sort(key=lambda solution: dg.calculate_total_edge_length(graph, solution))
        #print(neighbors[0], calculate_total_edge_length(graph, neighbors[0]))

        next_solution = None
        for neighbor in neighbors:
            if neighbor not in tabu_list:
                next_solution = neighbor
                break

        #if all neighbors are tabu, choose the best one
        if next_solution is None:
            next_solution = neighbors[0]

        #update tabu list
        tabu_list.append(next_solution)
        if len(tabu_list) > tabu_tenure:
            tabu_list.pop(0)

        #update current solution
        current_solution = deepcopy(next_solution)
        current_value = dg.calculate_total_edge_length(graph, current_solution)

        #update best solution
        if current_value < best_value:
            best_i = ind
            best_solution = deepcopy(current_solution)
            best_value = current_value

    return best_solution, best_value, best_i
```

KOMBINACIJA TABU PRETRAGE I NEKIH S-METAHEURISTIKA

Odlucili smo da kombinujemo ove 2 tehnike.

Ideja je da se koristi prethodno opisan algoritam tabu pretrage, ali da generisanju "komsija" resenja prethodi neki vid s-metaheuristike (lokalna ili sim. kaljenje).

Dakle, u svakoj iteraciji cemo uzeti najboljeg ne-tabu 'komsiju', a onda pokusati da ga unapredimo na pocetku sledece iteracije.

Na ovaj nacin imamo aspekat raznovrsnosti zahvaljujuci tabu tehnikama generisanja skupa resenja i tabu listi, ali takodje i intenzivno traganje za najboljim resenjem u manjem prostoru pretrage zbog drugih s-metaheuristika.

Limitiramo broj iteracija lokalnih pretraga bazirano na broju ukupnih iteracija algoritma.

REZULTATI
NAJBOLJIH KOMBINACIJA SVAKE TEHNIKE

	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

REZULTATI
NAJBOLJIH KOMBINACIJA SVAKE TEHNIKE

	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

REZULTATI
NAJBOLJIH KOMBINACIJA SVAKE TEHNIKE

	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

REZULTATI
PROSECNI REZULTATI SVIH KOMBINACIJA SVIH METODA

	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

REZULTATI
PROSECNI REZULTATI SVIH KOMBINACIJA SVIH METODA

	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

REZULTATI
PROSECNI REZULTATI SVIH KOMBINACIJA SVIH METODA

	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

ZAKLJUCAK

Gruba sila postaje neupotrebljiva cim dimenzije grafa(broj cvorova) postanu dvocifrene. Pohlepni algoritam, kao sto je ocekivano, nikada ne daje ni priblizno optimalna resenja, sluzi prvenstveno za poredjenje sa boljim metodama i kao pokazatelj da se ovakvi problemi moraju resavati adekvatnim merama. Kombinatorika ove vrste zahteva drugaciji pristup.

Obe vrste lokalne pretrage daju solidne rezultate u svim dimenzijama: u manjim dimenzijama nalaze optimalno resenje, a u vecim se nikada ne razlikuju puno od boljih resenja. Ono sto im je najveci adut jeste konzistentno visoka brzina, sto je ocekivano bez prisustva kompleksnijih modifikacija resenja.

Simulirano kaljenje u proseku daje iste rezultate kao lokalne pretrage, za (prosecno) isto vreme.

VNS daje konzistento najbolja resenja od svih s-metaheuristika, sto je za ocekivati sa velikim fokusom na pretragu razlicitih delova domena resenja. Problem je sto je takodje konzistentno najsporija metoda, vreme izvrsavanja brzo raste sa rastom dimenzije grafa.
Ipak treba drzati na umu da smo testirali 12 razlicitih kombinacija za VNS, koristeci razlicite metode za shaking, lokalnu pretragu i funkcije promene.
Prosecno, prostije kombinacije su imali po kvalitetu jako slicna resenja kao kompleksnije kombinacije, ali sa znacajno manjim vremenom izvrsavanja – dakle isplativije su.

ZAKLJUCAK

Mozda metod koji se najbolje pokazao su genetski algoritmi, sa ekvivalentnim istim ili cesto boljim od VNS-a, ali zato znacajno brzi u velikim dimenzijama. Umesto kompleksnih i velikih promena nad resenjem u svakoj iteraciji, sto postaje vremenski zahtevno sa vecim grafovima, GA koristi uvek brzi crossover kako bi dobio nova resenja, radeci prvenstveno sa indeksima. Mutacije su takodje prostije od kombinacija shaking funckije i lokalnih pretraga i retko se desavaju. Od tri dela algoritma, rezultati nam govore da zapravo selekcija igra najvecu ulogu u kvalitetu resenja – u proseku najbolje su turnirska i 'rank_selection_linear_ranking'.

Obicna Tabu pretraga daje optimalne rezultate za manje i srednje grafove, i to cini brze nego VNS i genetski algoritmi. Medjutim, za vece grafove generisanje "komsija" resenja postaje problematicno, bez obziru koju od mnogih testiranih tehnika koristimo, resenja postaju prevelika da bi se dovoljan broj resenja za diverzifikovan komsiluk generisao efikasno. Ovo u vecim dimenzijama utice na vreme izvrsavanja. Najbolji odnos kvaliteta resenja i vremena izvravanja daju metode koje prave mali kompromis na oba fronta - 'inverse' i 'scramble'.

Taloc (tabu+s-metaheuristike) skoro uvek uspe da nadje resenje bolje nego sve ostale testirane metode. Do odredjenog broja cvorova se moze reci da je ovo najbolja metoda i moze se koristiti za ovaj problem. Medjutim, za vece dimenzij GA ostaje najbolji izbor, jer bez obzira na kombinaciju metoda za generisanje "komsiluka" i pretragu (od kojih smo napravili 29), nikada nece pobediti GA u odnosu kvalitet resenja/vreme.

LITERATURA, REFERENCE I MATERIJALI KOJI SU KORISCENI PRILIKOM IZRADE RADA

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

HVALA NA PAZNJI!