Univerzitet u Sarajevu
Prirodno-matematički fakultet
ODSJEK ZA MATEMATIKU

A Zmaja od Bosne 33-35, 71 000 Sarajevo, BiH
T +387 33 279 874 F +387 33 649 342
W www.pmf.unsa.ba/matematika
E matematika@pmf.unsa.ba

2opt i 3opt optimizacije - Tabela poretka

Said Salihefendić

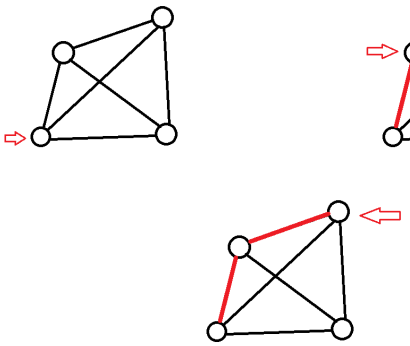
Odsjek za matematiku
Prirodno-matematički fakultet
Univerzitet u Sarajevu
08.01.2018

1 Uvod

Travelling Salesman Problem (TSP) je jedan od najpoznatijih NP kompletnih problema u kompjuterskim naukama i do dan danas je tema istraživanja u mnogim institucijama. O čemu se radi? Radi se o posjećivanju svih datih gradova, tj. lokacija sa što manjom udaljenošću tako da se posjete svi gradovi tačno jednom. Jednostavna je postavka, ali pronalazak njegovog optimalnog rješenja je daleko od jednostavnog. Pošto moramo posjetiti gradove tačno jednom, možemo posjetu gradova posmatrati kao određenu permutaciju. To nama automatski daje za n gradova $n!$ mogućih permutacija. To znači da već za $n = 31$ dobijamo približno 8.222839×10^{33} mogućih permutacija posjećivanja gradova. Poređenja radi, univerzum je star otprilike 4.32×10^{17} sekundi i neka imamo superkompjuter koji može pronaći 10^{15} permutacija po sekundi i pustimo ga da radi od rođenja univerzuma do danas. Iako superkompjuter radi relativno brzo, ni u tom ogromnom vremenskom intervalu ne bi stigao sve moguće permutacije testirati i dati optimalno rješenje. Trebalo bi mu jos otprilike 19 puta toliko vremena koliko je univerzum star kako bi mogao testirati sve moguće permutacije za $n = 31$. Ovakav način je suludan i vrlo teško da se izvrši u realnom vremenu, pa moramo posmatrati neke druge pristupe koji mogu da nađu vrlo dobro rješenje koja ima mala odstupanja od optimalnog. Mogući su pronalasci koliko iznosi optimalna udaljenost, ali ne i koja permutacija daje tu optimalnu udaljenost, pa na osnovu toga možemo lahko testirati naše algoritme. Tema ovog projekta nije u dokazivanju optimalnih rješenja, već da nađemo što bolje permutacije koje daju što bliže optimalnom rješenju, ako ne i da dadnu optimalno rješenje.

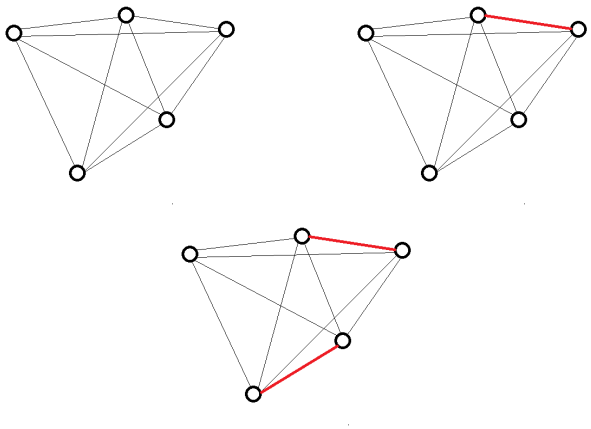
2 Greedy pristup

Jedan od početnih pristupa rješavanju ovog problema je *greedy* pristup. U ovom dijelu ćemo posmatrati dva algoritma koji su slični algoritmima za traženje minimalno pokrivajućeg stabla u grafu. Ideja je da se modifikuju Primov algoritam i Kruskalov algoritam i posmatrati kakve ture sve možemo dobiti.
Za modifikaciju Primovog algoritma ćemo nazvati najbliži susjed. Ideja je da se krene od jednog grada po našem odabiru i posjetimo neposjećeni najbliži susjed, a zatim od tog susjeda kojeg smo posjetili ponovo tražimo najbliži neposjećeni susjed i tako sve dok nismo posjetili svaki grad tačno jednom. Ovdje tura zavisi od početnog grada.



Slika 1: Dio ture nastankom algoritmom najbližeg susjeda

Drugi način kako možemo turu dobiti *greedy* postupokom je da modifikujemo Kruskalov algoritam i nazvat ćemo je najmanja grana. Ovdje je glavna ideja da uzmemo najmanju udaljenost dva grada u državi i uzimamo sve dok svaki grad nije posjećen. Moramo ovdje obratiti pažnju da odabirom najmanje grane ne stvorimo ciklus unutar ture i da svaki grad može imati maksimalno dvije grane, tj. ne smije biti raskrsnica.



Slika 2: Dio ture nastankom algoritmom najmanje grane

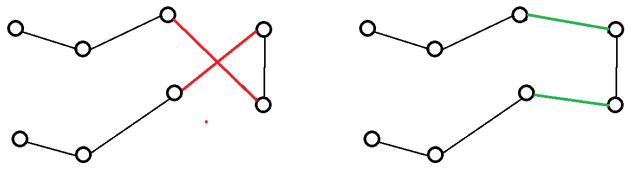
Vidjet ćemo kroz analizu algoritama koje budemo posmatrali da, u prosjeku, ovakav pristup ne daje baš dobra rješenja. Razlog tome je što se ne uzimaju u obzir druge putanje koje možda u zbiru daju bolje rješenje nego na način da uzimamo najmanju granu ili najbližeg susjeda. Jednostavno ovakvim pristupom ne uzimamo u razmatranja alternativne putanje.

3 Heuristički pristup

Dakle, vidjeli smo da sa nekim algoritmima možemo dobiti turu koju možemo da posmatramo. Naravno, lahko je izgenerisati turu na razne načine, da uzmemo turu takvu kakvu smo i unijeli u naš računar ili jednostavno neka računar sam izgeneriše turu. Ali, ne moramo tu stati i gledati da računar izgeneriše dovoljno dobru turu za naše potrebe. Postoje par algoritama koji mogu optimizovati našu datu turu, a to je upravo i tema ovog projekta da posmatramo heurističke algoritme koji poboljšavaju već date ture, a to su 2-opt i 3-opt heuristički algoritmi.

3.1 2-opt algoritam

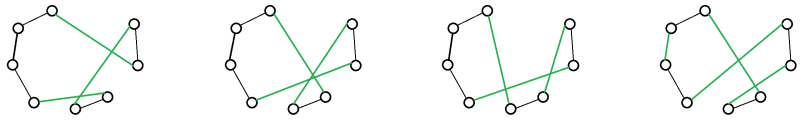
2-opt algoritam je *local search* algoritam koji uzima datu turu i vrši zamjenu neke dvije grane ukoliko se ispostavi da upravo tom zamjenom će dobiti bolju turu. U suštini, zbog prirode *local search*-a će lokalno da pretražuje svaku moguću kombinaciju zamjenu dvije grana i porediti dobivenu dužinu sa najboljom dužinom koji algoritam je do tada našao. Ukoliko je tok algoritma našao neko bolje rješenje, tada nakon što algoritam završi ponovo će se izvršiti nad izmijenjenom turom i sve to radi dok algoritam nije više u mogućnosti da nađe bolje rješenje zamjenom dvije grane.
Algoritam je relativno jednostavan i efikasan u polinomijalnom vremenu, što je mnogo bolje nego faktorijelsko vrijeme koje smo imali na raspolaganje u uvodu. Vidjet ćemo kasnije koliko je prosječno odstupanje od optimalnog rješenja za 2-opt algoritam primijenjen nad nekim turama koje budemo posmatrali.
2-opt, u suštini, mijenja najčešće dvije grane tamo gdje se one sijeku međusobno i vrši zamjenu tako da se više ne sijeku. Ideja je da se vrši 2-opt nad više tura i uzme se najbolja tura koja je dobivena pomoću 2-opt, jer 2-opt relativno brzo nađe približno dobro rješenje.
U ovom projektu, 2-opt se vrši nad nekom već datom permutacijom koja predstavlja poredak posjećivanja gradova. Ideja je da se taj vektor podijeli na tri dijela određenim i i j koji predstavljaju indeks odsječka vektora. Neka je taj vektor sada u dijelovima ABC i poredimo udaljenosti među granama koje želimo da zamijenimo i poredimo da li dobijemo bolje rješenje. Ukoliko smo dobili bolje rješenje, tada se vrši zamjena tako što A ostavimo kako jeste, obrnemo poredak dijela B, a C u istom poretku, tj. $A\bar{B}C$. Na ovaj način smo uspjel izvršiti tzv. *2-opt* zamjenu.



Slika 3: 2-opt zamjena nad granma koje se sijeku

3.2 3-opt algoritam

Kako smo posmatrali za 2-opt optimizaciju, vrlo je slična ideja i 3-opt, s tim da ovdje ulaze tri grane u zamjeni. Ovdje moramo obratiti pažnju, jer postoje više vrsta zamjena tri grana za razliku od 2-opta koji je imao zamjenu dvije grane na jedan način, dok ovdje, nakon malo više ispisivanja, imamo 4 načina zamjena tri grana.



Slika 4: Moguće 3-opt zamjene

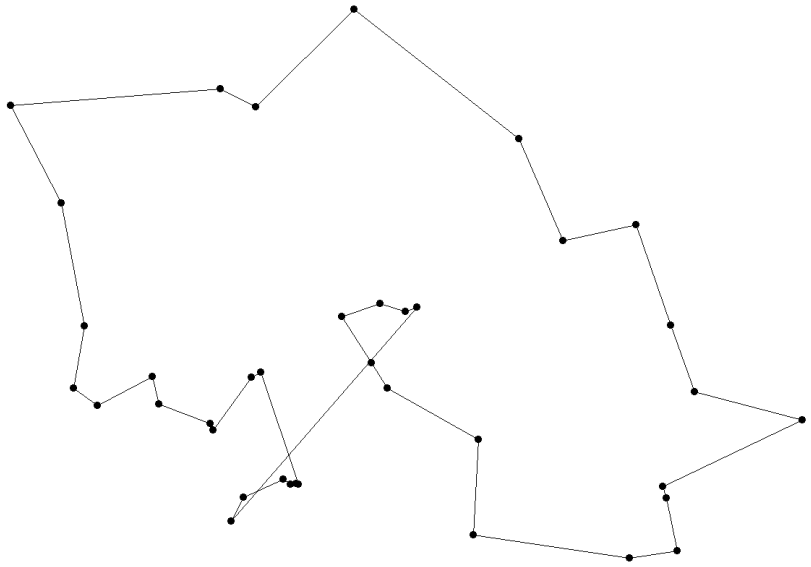
Struktura zamjena tri grana je ono što ih čini različitim. Ove zamjene se mogu postići sa više 2-opt zamjena na različitim mjestima tako da dobijemo zamjenu tri grana.
Ukoliko je tura 3-opt optimalan, onda je i 2-opt optimalan za tu datu turu. To znači da, ako je dati 3-opt našao turu koju ne može više da optimizuje, tada ni 2-opt ne može više da je optimizuje.
Što se tiče implementacije, ideja je slična 2-optu. Razlika je u tome što mi za dati vektor koja predstavlja turu podijelimo na četiri dijela određenim sa i, j i k i neka su dijelovi vektora redom označeni $ABCD$. Tada legalne zamjene tri grana koje se mogu jednostavno postići pozivom funkcije zamjena dvije grane na više različitih mjesta dobijamo sljedeće permutacije: $A\bar{B}\bar{C}D$, $ACBD$, $A\bar{C}BD$ i $AC\bar{B}D$. Primijetimo da su A i D fiksni i da samo mijenjamo ono što se nalazi između njih, mada zbog indeksa i, j i k vršimo izmjenu nad čitavim vektorom, vremenom. Kao i 2-opt, 3-opt vršimo sve dok ima poboljšanja i ukoliko nakon svih iteracija nije našao bolju turu, tada se prekida algoritam.
Za očekivati je da će 3-opt dati bolje rješenje od 2-opt heurističkog algoritma iz razloga to što mi ovdje imamo više mogućnosti izmjena grana. Također

se smatra da je 2-opt potez legalan 3-opt potez, pa onda se jednostavno vidi razlog zbog čega bi 3-opt trebao da dadne bolje rješenje.

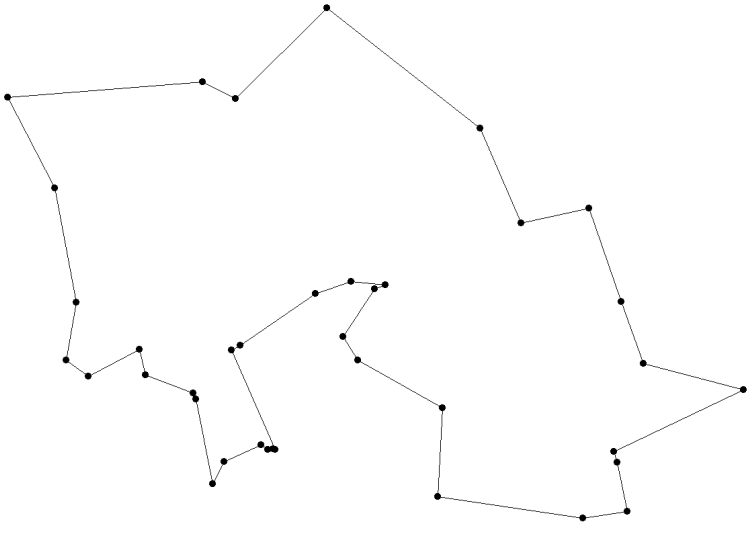
Ono što možemo zaključiti iz posmatranja heurističkog pristupa je da ona ne generiše permutaciju, već jednostavno poboljšava datu turu koja joj se proslijedi. Međutim, u tom slučaju mogu nastati određeni problemi kao što su *lokalni optimumi*. Lokalni optimum se pojavljuje upravo onda kada 2-opt i 3-opt ne mogu da nađu bolje rješenje. Samo optimalno rješenje je lokalni optimum, ali nije nužno jedini lokalni optimum, zbog čega se ovi algoritmi moraju vršiti nad različitim turama i odabrati onu turu koja se pokazala da ima najbolje rezultate. Moguće su situacije kada nije moguće izvršiti bolju zamjenu grana, a da odstupanje od optimalnog rješenja bude veća od 5%, što ovi algoritmi u prosjeku daju (vidjet ćemo kasnije na primjerima).
Jedan od načina da se izbjegnu *loši* lokalni optimumi jeste da se ponovo generiše random permutacija, pri čemu se onda ponovo vrši heuristički algoritmi nad njim i pritom se provjeri da li smo doblili bolje rješenje. Drugi način je da odaberemo što goru moguću permutaciju, pa da nad njim vršimo heurističke algoritme u nadi da ćemo moći naći vrlo dobro rješenje.
U biti, ovi algoritmi su vrlo efikasni (vrše se u polinomijalnom vremenu) i dobijamo približno dobre rezultate, pa možemo i da vidimo zbog čega imaju veliku primjenu u raznim oblastima nauke i inžinjeringa.

3.3 Visualizacija

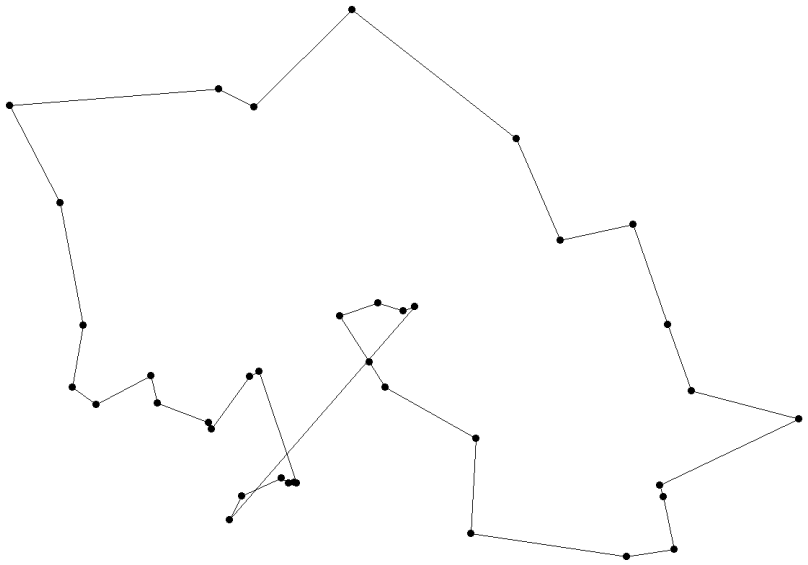
Vidjet ćemo sada na konkretnom primjeru šta rade ovi algoritmi, konkretno. Posmatrat ćemo iz ove stranice `http://www.math.uwaterloo.ca/tsp/world/countries.html` državu *Djibouti* koja ima 38 gradova. Relativno mal broj gradova, ali sjetimo se da ako bismo išli na onaj način iz uvoda, morali bismo provjeriti svih 38! permutacija kako bismo našli optimalno rješenje, a vidjeli smo šta znači za 31! koliko nam treba vremena, tako da ovo nije uopšte pametan način da riješimo problem.
Dokazano je da za Djibouti optimalna tura je dužine 6656 jedinica, pa ćemo preuzet podatke iz ove stranice i provjeriti kako se ponašaju naša implementacija ovih algoritama (pored ove dokumentacije imate i kod sa komentarima. Sva pitanja, komentare i sugestije možete slati na mail autora *said-salihefendic@gmail.com*). Visualizacija ovih algoritama ostvarena je pomoću C++ buildera (RAD 10.2.2 je zadnja verzija od ovog datuma).



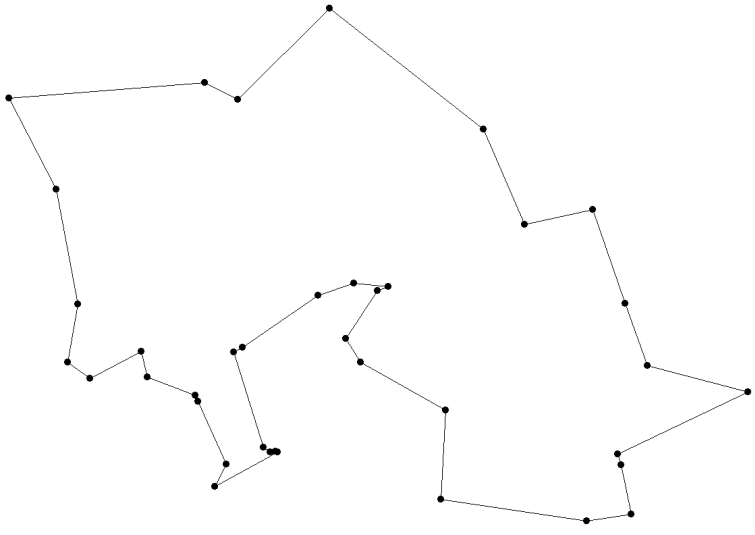
Slika 5: Tura za Djibouti - Algoritam najmanja grana



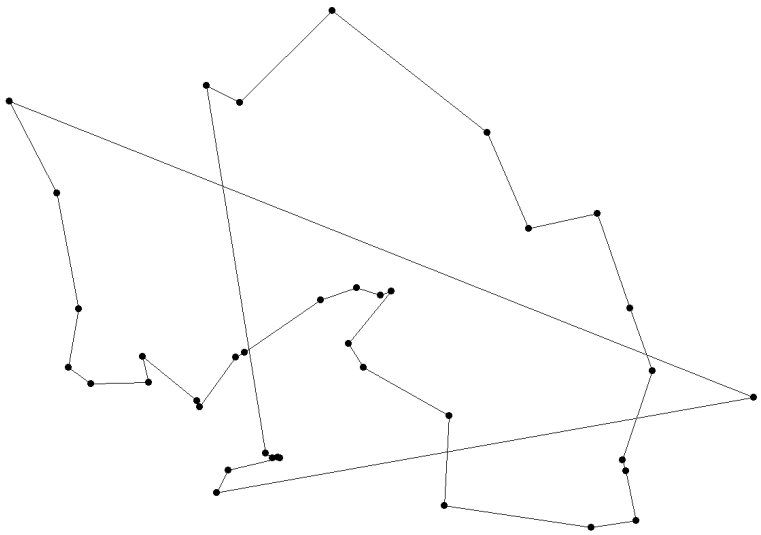
Slika 6: Tura za Djibouti - 2-opt nad turom najmanja grana



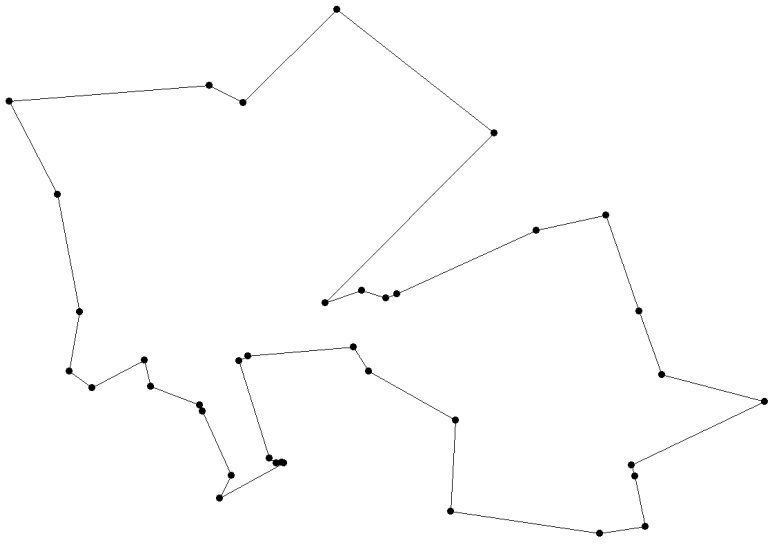
Slika 7: Tura za Djibouti - 3-opt nad turom najmanja grana



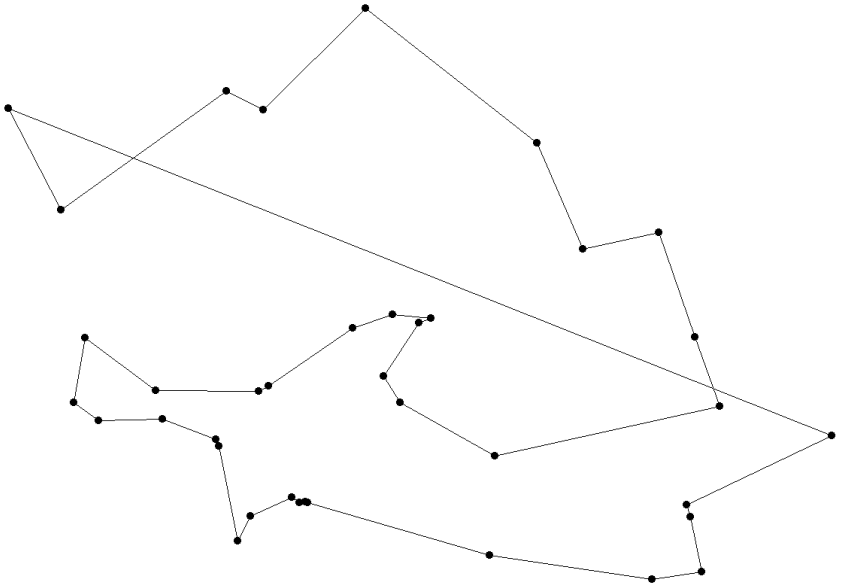
Slika 8: Tura za Djibouti - 3-opt nad turom najmanja grana nakon 2-opt



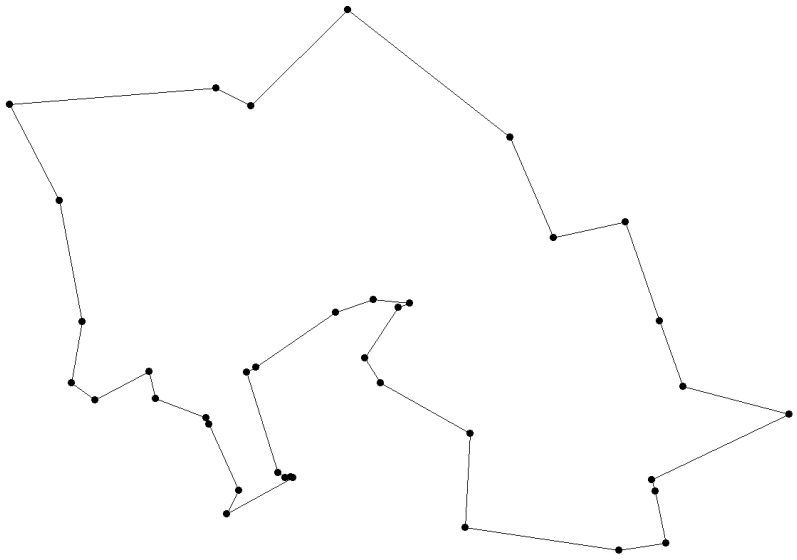
Slika 9: Tura za Djibouti - Algoritam najbliži susjed (početak od 0)



Slika 10: Tura za Djibouti - 2-opt nad turom najbliži susjed



Slika 11: Tura za Djibouti - 3-opt nad turom najbliži susjed



Slika 12: Tura za Djibouti - 3-opt nad turom najbliži susjed nakon 2-opt

4 Analiza

U ovom dijelu ćemo posmatrati kako heuristički algoritmi 2-opt i 3-opt poboljšavaju neke ture i koliko procentualno odstupaju od optimalnog rješenja. Cilj je da primijenimo neke *greedy* algoritme prije nego što izvršimo 2-opt optimizaciju, a nakon toga želimo posmatrati 3-opt prije 2-opt optimizacije i nakon 2-opt da primijenimo 3-opt i želimo da vidimo kakve sve rezultate možemo da dobijemo.

Posmatrat ćemo turu krenuvši od početka i redom do kraja grada, onako kako su i unešeni i program, te ćemo ovakvu turu nazvati tura poredak. Također, iskoristit ćemo i ture dobijene greedy algoritmima najbližeg susjeda i najmanje grane, a zatim ćemo posmatrati neku slučajnu turu i nazvat ćemo je random turu.

4.1 Tura poredak

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt	Optimalno
Djibouti (38)	17098	7176	8540	6656	6656
Sahara (29)	52284	28084	33223	27603	27603
Qatar (194)	39561	10172	10572	9758	9352

Tabela 1: Dužine ture

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	156.881	7.8125	28.3	0
Sahara (29)	89.414	1.7425	20.36	0
Qatar (194)	323.02	8.77	13.04	4.34

Tabela 2: Odstupanje od optimalnog rješenja (u %)

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	0.001	0.005	0.316	0.31
Sahara (29)	0.001	0.005	0.162	0.162
Qatar (194)	0.001	1.511	320.79	429.44

Tabela 3: Trajanje algoritama (u sekundama)

4.2 Tura najbliži susjed

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt	Optimalno
Djibouti (38)	9745	7152	8558	6656	6656
Sahara (29)	36388	28804	33256	28044	27603
Qatar (194)	11640	10580	9848	9848	9352

Tabela 4: Dužina ture

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	46.4	7.45	28.58	0
Sahara (29)	31.82	4.35	20.48	0
Qatar (194)	24.46	13.13	5.3	5.3

Tabela 5: Odstupanje od optimalnog rješenja (u %)

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	0.001	0.0019	0.314	0.324
Sahara (29)	0.001	0.001	0.162	0.108
Qatar (194)	0.002	1.089	430.285	444.374

Tabela 6: Trajanje algoritama (u sekundama)

4.3 Tura najmanja grana

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt	Optimalno
Djibouti (38)	7019	6660	7019	6656	6656
Sahara (29)	39691	28804	33367	28044	27603
Qatar (194)	11499	10331	10709	9726	9352

Tabela 7: Dužina tura

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	5.453	0.06	5.453	0
Sahara (29)	43.79	4.35	20.88	1.59
Qatar (194)	22.96	10.47	14.51	3.99

Tabela 8: Odstupanja od optimalnog rješenja (u %)

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	0.001	0.004	0.147	0.316
Sahara (29)	0.002	0.003	0.108	0.109
Qatar (194)	0.018	0.665	338.109	323.911

Tabela 9: Trajanje algoritama (u sekundama)

4.4 Random tura

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt	Optimalno
Djibouti (38)	28977	7075	6886	6656	6656
Sahara (29)	119291	28503	29439	27603	27603
Qatar (194)	94205	10356	10127	9566	9352

Tabela 10: Dužine tura

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	335.351	6.29	3.45	0
Sahara (29)	332.17	3.26	6.65	0
Qatar (194)	907.32	10.73	8.28	2.29

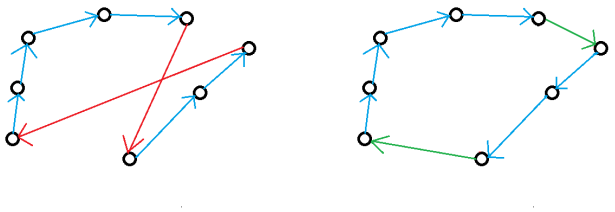
Tabela 11: Odstupanja od optimalnog rješenja (u sekundama)

Država	Bez optimizacije	2opt	3opt	3opt nakon 2opt
Djibouti (38)	0.001	0.001	0.481	0.329
Sahara (29)	0.001	0.004	0.108	0.161
Qatar (194)	0.001	1.083	319.443	317.987

Tabela 12: Trajanje algoritma (u sekundama)

5 Slučaj usmjerenih grafova

Sve vrijeme smo obraćali pažnju na neusmjerene gravoe, to jeste, nismo obraćali pažnju na smjer kretanja ture. Ukoliko bismo dodali i smjer u kojem bismo željeli ići, što se tiče euklidske udaljenosti, tok i logika programa je ista, ali ukoliko bismo obraćali pažnju na, recimo, gorivo kao težinu grane u grafu, tada bismo morali malo promijeniti logiku. Jedan od načina je da izvršimo zamjenu grana, promijenimo smjer kretanja u segmentu gdje se izvršila zamjena grana (pošto svakako posmatramo vektor permutacija), te izmjerimo konačnu dužinu, tj. ukupnu težinu te ture i provjerimo da li je bolje rješenje. Sve zavisi koja je težina grana, ali ako su u oba smjera iste težine, onda je logika ista kao kod neusmjerenih, a ukoliko se težine razlikuju, tada moramo izvršiti ne samo zamjenu grana, nego i smjer u kojem se kretala tura.
Sljedeća slika prikazuje kako bi se ponašalo mijenjanje smjera nakon zamjene dvije grane. Primijetite promjenu smjera ture nakon zamjena crvenih grana. Slična logika vrijedi i za 3-opt.



Slika 13: 2-opt zamjena nad granma koje se sijeku (usmjereni)