SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Rješavanje problema trgovačkog putnika koristeći genetski algoritam

Meko računarstvo

Laboratorijska vježba 2

Ivan Gudelj

Diplomski studij računarstva, DRB

Osijek, 2022.

1. UVOD

Cilj druge laboratorijske vježbe bio je proučiti osnove i način rada genetskog algoritma kroz problem određivanja najkraćeg puta obilaska zadanih hrvatskih gradova, ali uz uvjet da se svaki grad posjeti točno jednom.

Program za rješavanje ovog problema napisan je u Python programskom jeziku, gdje je moguće unositi veličinu populacije, postotak mutacije i broj elitnih članova. Broj generacija ograničavamo unošenjem određene vrijednosti. Zbog stohastičke naravi genetskog algoritma svaki eksperiment bilo je potrebno ponoviti najmanje 5 puta i zabilježiti srednji (medijan) rezultat. Najbolje pronađeno rješenje od svih obavljenih mjerenja bit će prikazano na karti hrvatske.

2. OPIS PROBLEMA I RJEŠENJA

2.1. Opis problema i genetski algoritam

Prvo ćemo opisati osnove genetskog algoritma. Genetski algoritam je metaheuristička metoda optimiranja koja se temelji na ideji biološke evolucije vrsta, tj. preživljavanje najsposobnijih vrsta. Evolucija je prirodni proces traženja najbolje i najprilagodljivije jedinke u okolini i uvjetima u prirodi. Jedinka koja je najbolje prilagođena uvjetima i okolini u kojoj živi ima najveću vjerojatnost preživljavanja i parenja, a time i prenošenja svojega genetskog materijala na svoje potomke. Genetički podaci (parametri) koji obilježavaju jedinku zapisani su u kromosomima.

U populaciji jedne vrste, nova se jedinka stvara selekcijom (reprodukcijom) i rekombinacijom (križanjem) genetičkih materijala (gena) obaju roditelja. Time se dobiva različitost među jedinkama iste vrste, ali i sličnosti s roditeljima jedinke. Na gen jedinke (djeteta) može djelovati i mutacija. Riječ je o slučajnom mijenjanju genetskog materijala koji nastaje pod djelovanjem vanjskih uzroka. Za dobivanje dobrog rješenja genetskog algoritma dovoljno je kodirati problem i kvalitetno definirati funkciju cilja (definirati što je dobro rješenje).

Odabrani roditelji dobrih svojstava imaju šansu dati potomka koji će imati bolja svojstva od svakog pojedinog roditelja. Roditelji dobrih svojstava imaju veću šansu dati potomke i prenijeti svoja svojstva (gene) u iduću generaciju. Svaka sljedeća generacija imat će sve više dobrih svojstava.

Populacija je skup jedinki odnosno rješenja u i-tom koraku rada algoritma. Kromosom je jedna jedinka rješenja odnosno jedno moguće rješenje zadanog problema. Dok gen predstavlja jediničnu informaciju odnosno nositelj je jedne informacije iz rješenja. Geni se mogu kodirati na razne načine koje odgovaraju pojedinim tipovima problema.

Osnovna strukrura genetskog algoritma podjeljena je na 6 koraka:

Generiraj početnu populaciju mogućih rješenja (kromosomi)
Odredi sposobnost svakog kromosoma u populaciji
Generiraj nove kromosome koristeći genetičke operatore
Odbaci nepoželjne jedinke (kromosome) populacije
Uključi nove kromosome u populaciju da se stvori nova populacija
Korake 2. – 6. nastavi sve dok nije zadovoljen unaprijed određeni uvjet

Genetski algoritam prvo mora odabrati određene „dobre“ roditelje za stvaranje nove populacije. Odabir roditelja se vrši pomoću metoda selekcije. Stoga je svrha selekcije čuvanje i prenošenje dobrih svojstava na slijedeću generaciju jedinki.

Genetske algoritme, s obzirom na vrstu selekcije, dijelimo na generacijske i eliminacijske. Način kodiranja ili prikaz rješenja može bitno utjecati na učinkovitost genetičkog algoritma, pa je stoga izbor prikaza izuzetno značajan. Neki od tipova genetskog kodiranja su: binarni, vrijednosni, permutacijski i stablasti.

Kod binarnog kodiranja gen može poprimiti samo dvije vrijednosti: 0 ili 1 (primjer punjenje spremnika, određuje se da li je određeni predmet u spremniku). Kod vrijednosnog kodiranja gen može poprimiti cjelobrojne/realne vrijednosti iz zadanog intervala (primjeri traženja maksimuma f-je više varijabli). Kod permutacijskog kodiranja gen može poprimiti cjelobrojne vrijednosti tako da kromosom uvijek sadrži sve brojeve 1 ... N u različitom redoslijedu (primjer: problem trgovačkog putnika, pronaći najkraći put obilaska N gradova tako da se svaki grad posjeti točno jednom). Kod stablastog kodiranja gen je čvor stabla (primjer: pronalaženje analitičke funkcije iz skupa vrijednosti).

Veličina populacije N određuje se na samom početku algoritma. Najčešće se početna populacija kromosoma (ili potencijalnih rješenja) generira tako da se generira N slučajnih brojeva ili rješenja u intervalu [d,g] te se prikažu u odgovarajućem obliku ovisno o načinu prikaza. Obično se uzima da je N = konst, odnosno veličina populacije se ne mijenja tijekom evolucije.

Najvažniji dio primjene genetskog algoritma je zapravo funkcija dobrote, tj. fitness funkcija. Ova funkcija govori koliko je određeno rješenje dobro i što je dobrota jedinke, tj. rješenja veća (odnosno manja, ovisno o problemu i načinu implementacije), jedinka ima veću vjerojatnost preživljavanja i križanja. Ova funkcija je ključ za proces selekcije. Funkcija sposobnosti može biti bilo koja nelinearna, prekidna, nederivabilna pozitivna funkcija jer je bitno samo odrediti sposobnost za svaki kromosom.

Problem u svojoj definiciji zahtjeva obilazak gradova s popisa bez ponavljanja. Za njegovo rješavanje potrebno je koristiti permutacijsko kodiranje budući ne sme biti istih gradova, a samim time i odgovarajuće algoritme stvaranja početne populacije, te rekombinacije i mutacije. Za navedeni problem je potrebno u kromosome uvrstiti 21 cjelobrojni gen budući su svih 21 gradova označeni indeksom i potrebno ih jew obići određenim redoslijedom. U funkciji dobrote (*fitness function*) potrebno je zbrajati udaljenosti između dva susjedna gena odnosno grada kako bi dobili ukupnu udaljenost ("dobrotu") koja pripada pojedinom kromosomu.

Rješavanje optimizacijskog koda kao što je problem trgovačkog putnika gdje nam na početku nije poznato rješenje (optimum koji želimo postići), genetski algoritam radi kontinuirano do maksimalno dozvoljene generacije. Nemamo predefiniran cilj koji prekida evoluciju genetskog algoritma nakon što ga se postigne. Nakon što algoritam dosegne svoju konačnu generaciju treba ispisati redoslijed obilaska gradova ili grafički ga prikazati, kao i ukupnu udaljenost. Rezultirajući kromosom se formira kao niz cjelobrojnih gena koji indeksiraju gradove iz datoteke *gradovi.xml* kao npr.

Kromosom = [0, 10, 11, 4, 2, 8, 17, 13, 14, 19, 3, 7, 12, 6, 16, 20, 9, 1, 5, 18, 15]

2.2. Rješavanje problema trgovačkog putnika

Potrebno je riješiti problem trgovačkog putnika tako da se odredi najkraći put obilaska hrvatskih gradova, ali uz uvjet da svaki grad treba posjetiti točno jednom (permutacijsko kodiranje). Kao prvi zadatak nismo uzimali u obzir prelazak granice Slovenije te Bosne i Hercegovine. Drugi zadatak gledamo kao nadogradnju prvog zadatka, gdje stavljamo uvjet da trgovački putnik obilazak gradova mora obaviti bez prelaska hrvatske granice (uzeći u opbzir da se kreće ravnom linijom između gradova). Bilo je potrebno analizirati ponašanje algoritma za 21 grad. Također, bilo je potrebno ispitati utjecaj promjene parametara algoritma kao što su veličina populacije,faktor mutacije i broj elitnih članova na brzinu konvergiranja prema rješenju i točnost rješenja. Svaki eksperiment bilo je potrebno ponoviti najmanje 5 puta i zabilježiti srednji (medijan) rezultat.

**Grafovi s funkcijama dobrote (*fitness* funkcije) prikazuju funkciju medijana 5 rješenja, za određene parametre algoritma.

3. ANALIZA REZULTATA

Parametre genetskog algoritma bilo je potrebno mijenjati na sljedeće vrijednosti:

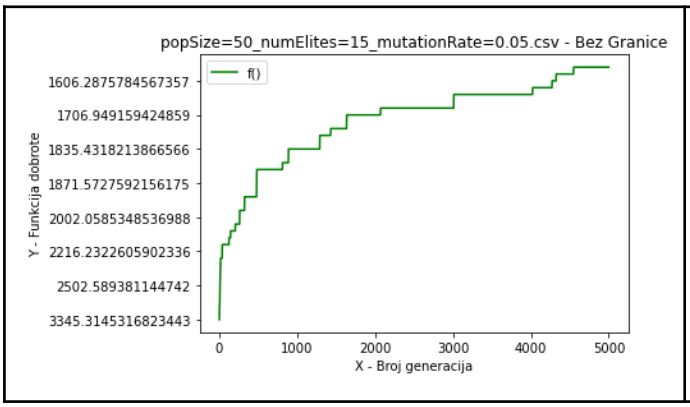
- populacija: 50, 100, 200, 400

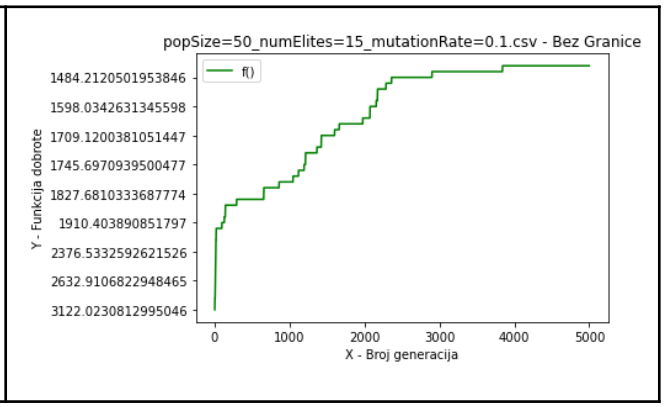
- mutacija: 5%, 10%, 15%, 20%

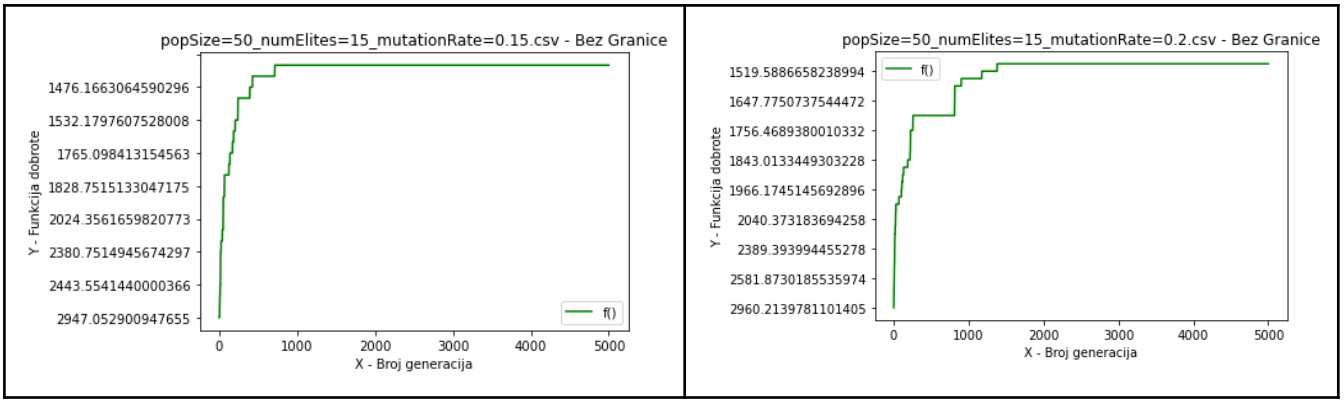
- broj elitnih članova: 5, 10, 15, 20

3.1. Ovisnost o postotku mutacije (bez granica)

Populacija	50			
Broj generacija	5000			
Broj elitnih članova	15			
Mutacija	5%	10%	15%	20%
Broj generacija po završetku algoritma	1648,1559, 1712,1578, 1624	1466,1720, 1639,1511, 1646	1613,1430, 1587,1493, 1591	1542,1630, 1667,1634, 1509
Prosječan najkraći put	1624.53	1596.95	1543.41	1596.88
Najbolje rješenje	[13, 19, 14, 5, 1, 9, 20, 16, 17, 3, 8, 7, 12, 6, 2, 4, 11, 10, 0, 18, 15]	[15, 18, 5, 1, 9, 20, 8, 6, 12, 7, 3, 17, 13, 19, 14, 16, 2, 4, 11, 10, 0]	[0, 10, 11, 4, 2, 8, 5, 15, 18, 1, 9, 14, 20, 16, 19, 13, 17, 3, 7, 6, 12]	[13, 17, 3, 7, 12, 6, 8, 20, 5, 15, 18, 1, 9, 14, 19, 16, 2, 4, 11, 10, 0]

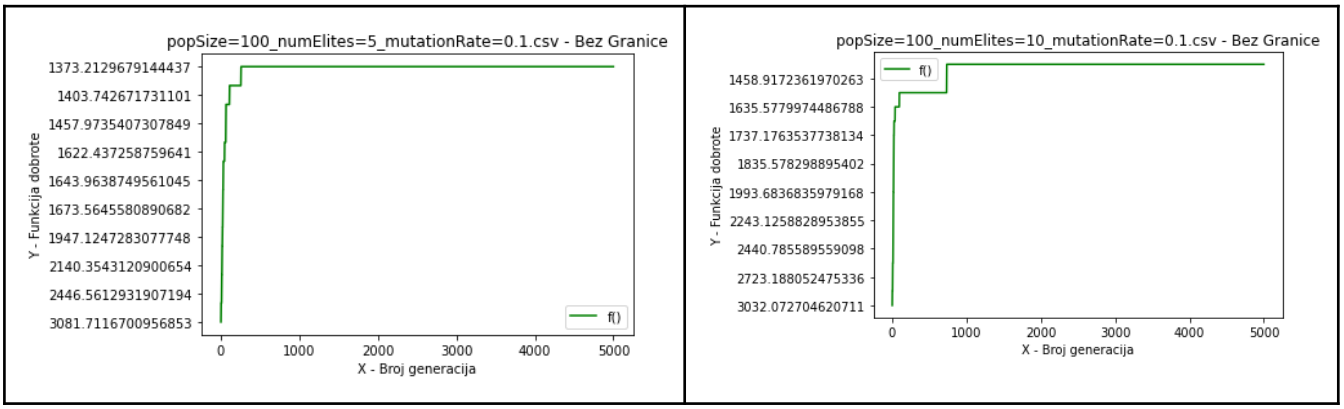


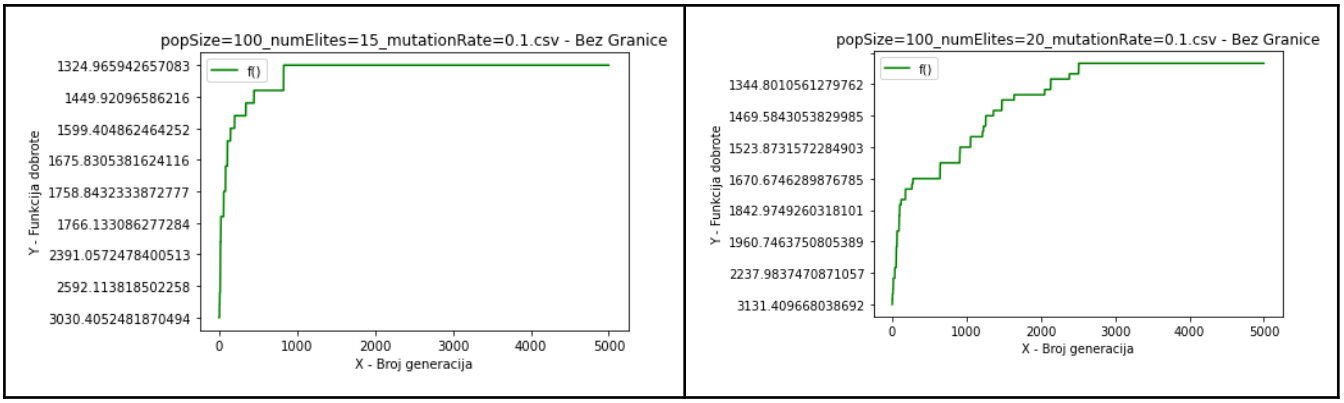




3.2. Ovisnost o broju elitnih članova (bez granica)

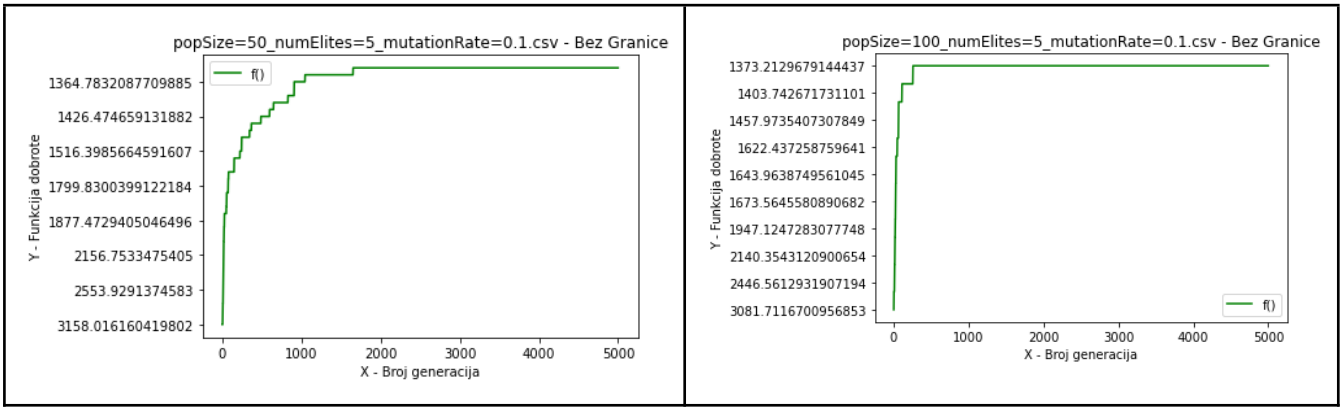
Populacija	100			
Broj generacija	5000			
Mutacija	10%			
Broj elitnih članova	5	10	15	20
Broj generacija po završetku algoritma	1373,1502, 1511,1489, 1679	1435,1619, 1701,1719, 1572	1324,1486, 1636,1600, 1603	1574,1463, 1456,1605, 1295
Prosječan najkraći put	1511.15	1609.79	1530.38	1478.95
Najbolje rješenje	[0, 10, 11, 4, 2, 8, 16, 3, 6, 12, 7, 17, 13, 19, 14, 20, 9, 1, 5, 18, 15]	[16, 8, 2, 6, 12, 7, 3, 17, 13, 19, 14, 5, 15, 18, 1, 9, 20, 4, 11, 10, 0]	[0, 10, 11, 4, 2, 6, 12, 7, 8, 3, 17, 13, 19, 14, 5, 15, 18, 1, 9, 20, 16]	[15, 18, 5, 1, 9, 20, 14, 19, 13, 17, 16, 3, 7, 12, 6, 8, 2, 4, 11, 10, 0]

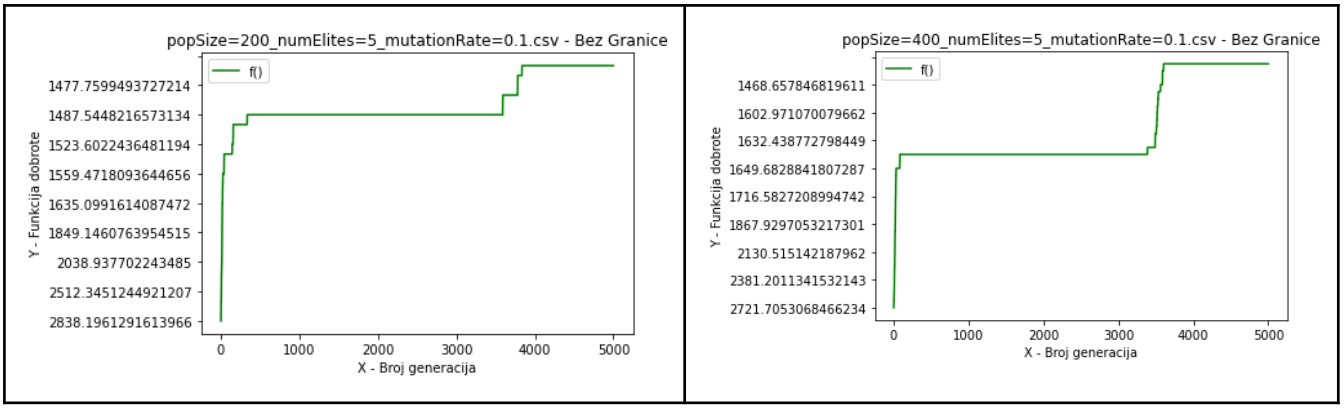




3.3. Ovisnost o veličini populacije (bez granica)

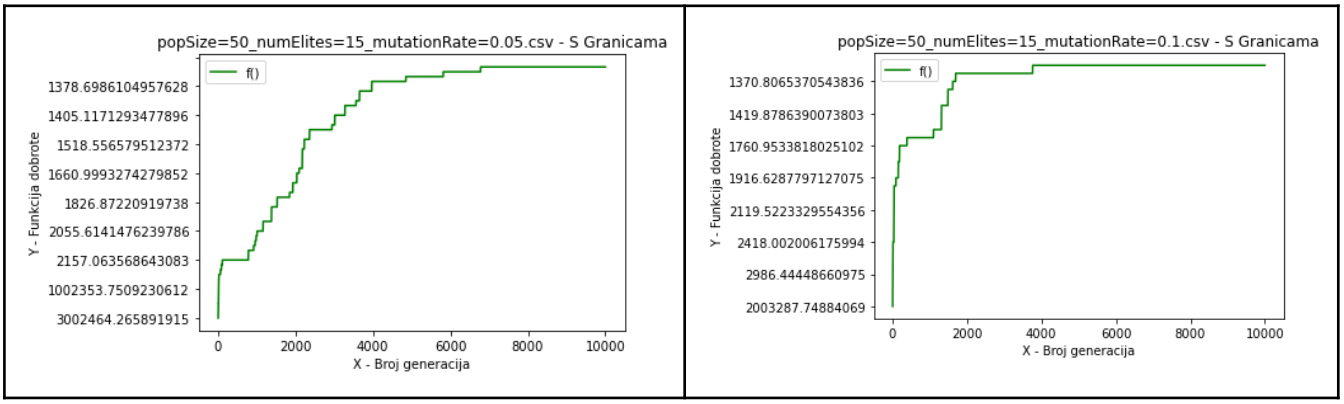
Broj elitnih članova	5			
Broj generacija	5000			
Mutacija	10%			
Veličina populacije	50	100	200	400
Broj generacija po završetku algoritma	1350,1724, 1703,1770, 1511	1373,1502, 1511,1489, 1679	1452,1458, 1631,1438, 1492	1653,1548, 1466,1407, 1474
Prosječan najkraći put	1611.86	1511.15	1494.65	1509.97
Najbolje rješenje	[0, 10, 11, 4, 20, 9, 1, 18, 15, 5, 14, 19, 13, 17, 16, 3, 8, 2, 7, 6, 12]	[0, 10, 11, 4, 2, 8, 16, 3, 6, 12, 7, 17, 13, 19, 14, 20, 9, 1, 5, 18, 15]	[15, 18, 5, 1, 9, 14, 16, 8, 2, 6, 12, 7, 3, 17, 13, 19, 20, 4, 11, 10, 0]	[12, 6, 7, 2, 8, 3, 16, 17, 13, 19, 5, 15, 18, 1, 9, 14, 20, 4, 11, 10, 0]

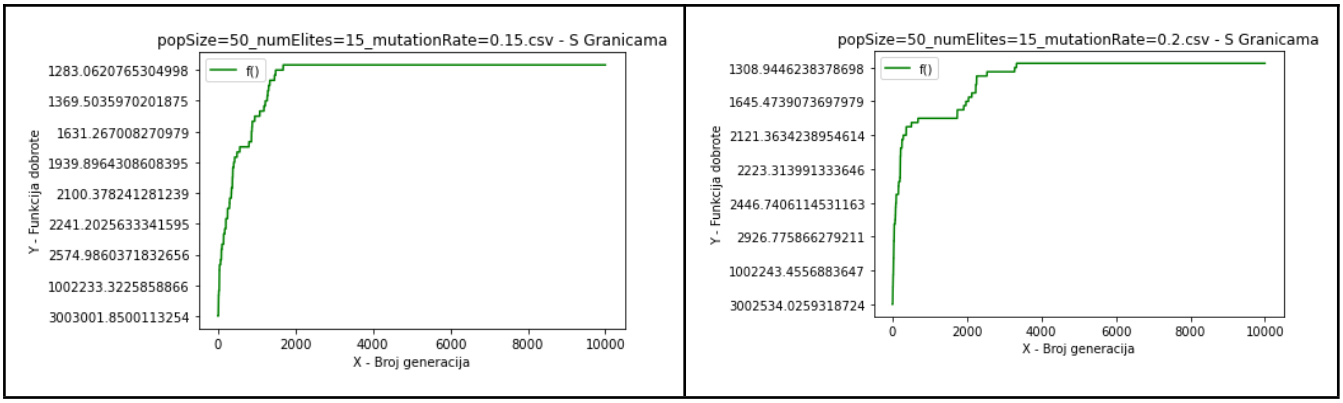




3.4. Ovisnost o postotku mutacije (s granicama)

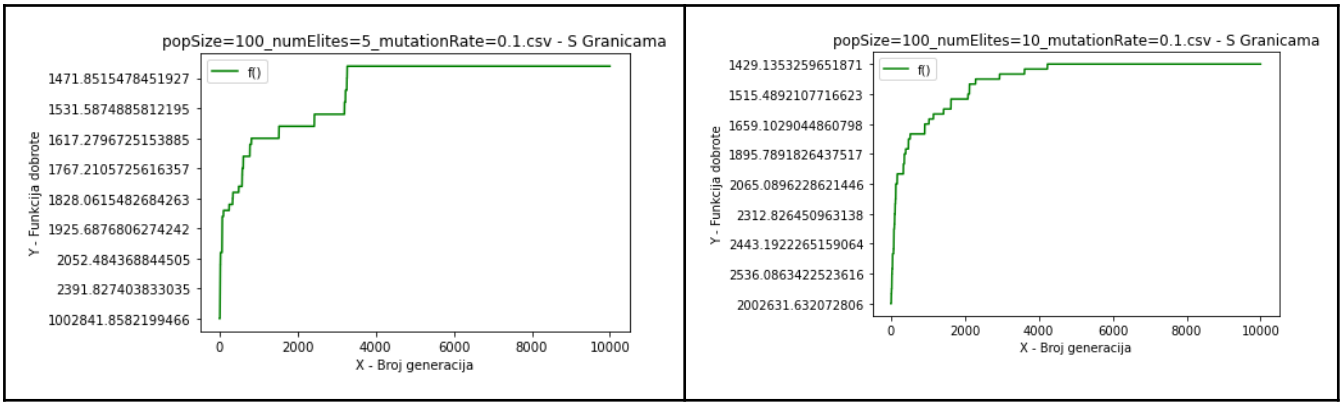
Populacija	50			
Broj generacija	10000			
Broj elitnih članova	15			
Mutacija	5%	10%	15%	20%
Broj generacija po završetku algoritma	1868,1592, 1261,1407, 1630	1607,1552, 1293,1835, 1577	1625,1612, 1575,1659, 1261	1433,1546, 1738,1295, 1528
Prosječan najkraći put	1552.10	1573.32	1546.84	1508.44
Najbolje rješenje	[15, 18, 5, 1, 9, 20, 14, 19, 13, 17, 16, 3, 8, 7, 12, 6, 2, 4, 11, 10, 0]	[0, 10, 11, 4, 2, 6, 12, 7, 8, 3, 17, 13, 19, 16, 20, 14, 9, 1, 5, 18, 15]	[0, 10, 11, 4, 2, 6, 12, 7, 8, 3, 16, 17, 13, 19, 14, 20, 9, 1, 5, 18, 15]	[0, 10, 11, 4, 2, 8, 6, 12, 7, 3, 16, 17, 13, 19, 14, 20, 9, 1, 5, 18, 15]

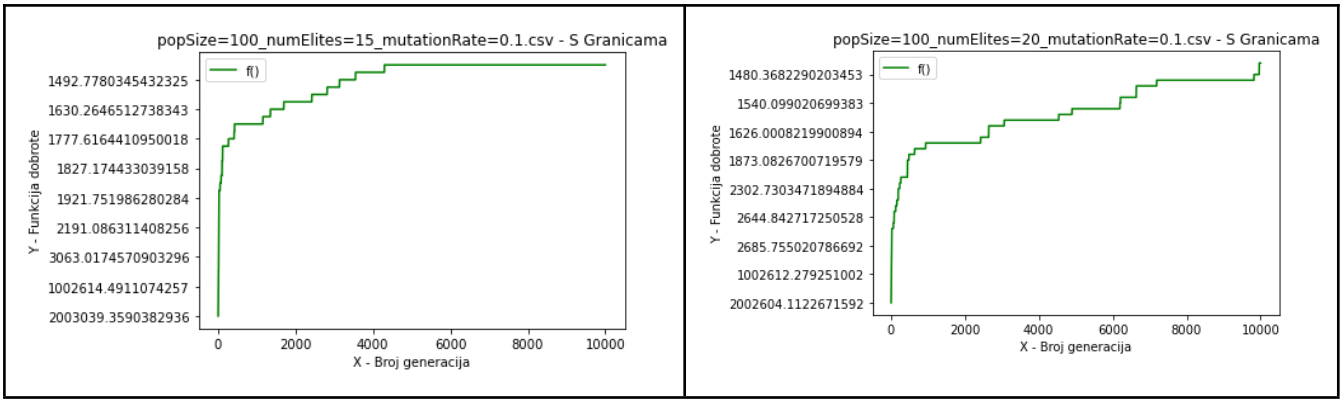




3.5. Ovisnost o broju elitnih članova(s granicama)

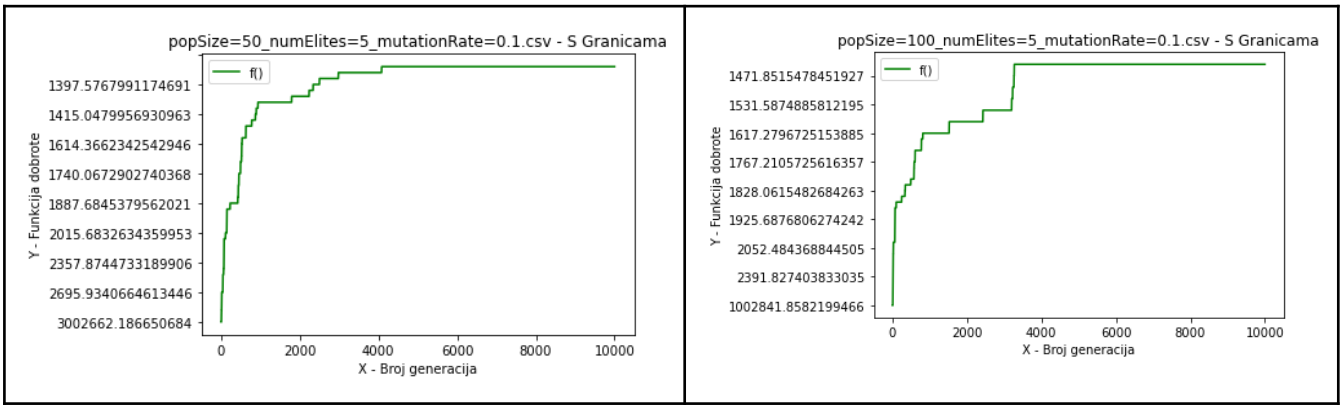
Populacija	100			
Broj generacija	10000			
Mutacija	10%			
Broj elitnih članova	5	10	15	20
Broj generacija po završetku algoritma	1645,1458, 1513,1484, 1695	1659,1429,156 1,1663,1779	1641,1487,204 7,1653,1662	1529,1550,1298 ,1461,1592
Prosječan najkraći put	1559.44	1618.50	1698.56	1486.57
Najbolje rješenje	[15, 18, 5, 1, 9, 14, 20, 16, 6, 12, 7, 3, 17, 13, 19, 8, 2, 4, 11, 10, 0]	[0, 10, 11, 4, 8, 20, 9, 1, 18, 15, 5, 14, 19, 13, 17, 16, 3, 2, 7, 6, 12]	[0, 10, 11, 4, 2, 8, 20, 19, 16, 6, 12, 7, 3, 17, 13, 14, 9, 1, 5, 18, 15]	[15, 18, 5, 1, 9, 20, 19, 14, 13, 17, 16, 3, 8, 7, 12, 6, 2, 4, 11, 10, 0]

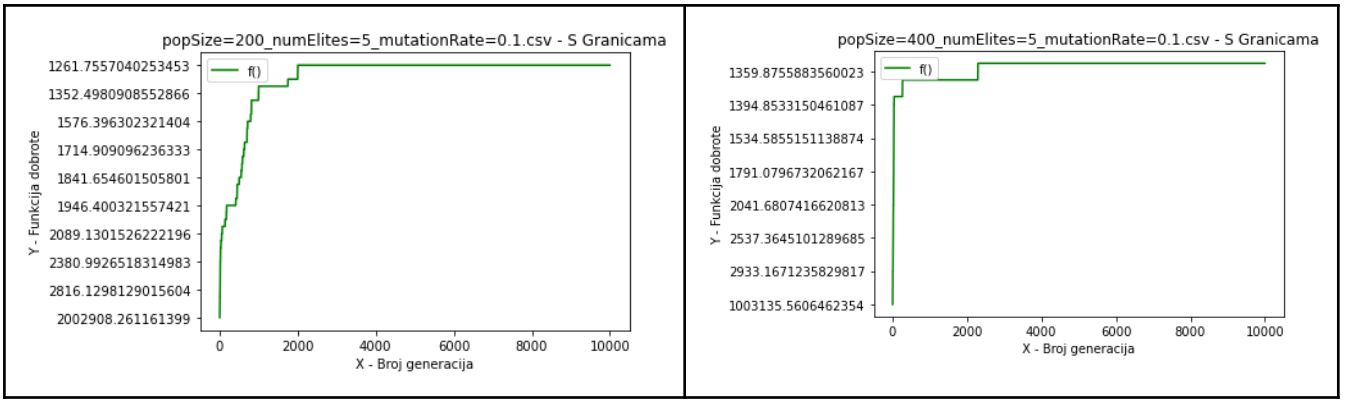




3.6. Ovisnost o veličini populacije (s granicama)

Broj elitnih članova	5			
Broj generacija	10000			
Mutacija	10%			
Veličina populacije	50	100	200	400
Broj generacija po završetku algoritma	1825,1352,1355, 1991,1552	1645,1458,151 3,1484,1695	1332,1303, 1551,1358, 1261	1604,1455, 1326,1563, 1530
Prosječan najkraći put	1615.50	1559.44	1361.49	1496.18
Najbolje rješenje	[0, 10, 11, 4, 2, 6, 12, 7, 8, 16, 20, 9, 1, 18, 15, 5, 14, 19, 13, 17, 3]	[15, 18, 5, 1, 9, 14, 20, 16, 6, 12, 7, 3, 17, 13, 19, 8, 2, 4, 11, 10, 0]	[15, 18, 5, 1, 9, 20, 14, 19, 13, 17, 16, 3, 8, 7, 12, 6, 2, 4, 11, 10, 0]	[0, 10, 11, 4, 2, 6, 12, 7, 8, 3, 17, 16, 20, 5, 15, 18, 1, 9, 14, 19, 13]





3.7. Funkcija dobrote (*fitness function*)

```
distanceMatrix = [[0 for x in range(len(sirine))] for y in range(len(duzine))]

for i in range (len(sirine)):
    for j in range (len(duzine)):
        #Širina = širina1-širina2*110.64
        #Dužina = dužina1-dužina2+78.85
        #Udaljenost = sqrt(širina^2 + dužina^2)
        currSirina = (sirine[i] - sirine[j]) * 110.64
        currDuzina = (duzine[i] - duzine[j]) * 78.85
        distance = math.sqrt((currSirina*currSirina)+(currDuzina*currDuzina))
        distanceMatrix[i][j] = distance


def evaluateInd(individual):
    fit_val = 0.0 #starting fitness is 0
    for i in range(len(individual) - 1):
        fit_val += distanceMatrix[individual[i]][individual[i + 1]]
    return fit_val,
```

Funkcija dobrote bez uvjeta granice

```
#####################################################
#         Definiranje granica za uvjet.
slovPoint1 = (45.48,13.52)
slovPoint2 = (45.48,15.37)
slovPoint3 = (46.38,16.28)

bosPoint1 =  (45.16,18.1)
bosPoint2 =  (45.14,15.7)
bosPoint3 =  (42.7,18.2)
slovBorder1 = LineString([slovPoint1,slovPoint2])
slovBorder2 = LineString([slovPoint2,slovPoint3])
bosBorder1 = LineString([bosPoint1,bosPoint2])
bosBorder2 = LineString([bosPoint2,bosPoint3])
```

```
def checkForSlovenia(width1,width2,length1,length2):
    intersection = False
    citiesLine = LineString([(width1,length1),(width2,length2)])
    if(citiesLine.intersects(slovBorder1) or citiesLine.intersects(slovBorder2)):
        intersection = True
    return intersection

def checkForBosnia(width1,width2,length1,length2):
    intersection = False
    citiesLine = LineString([(width1,length1),(width2,length2)])
    if(citiesLine.intersects(bosBorder1) or citiesLine.intersects(bosBorder2)):
        intersection = True
    return intersection
#####################################################
for i in range (len(sirine)):
    for j in range (len(duzine)):
        #Širina = širina1-širina2*110.64
        #Dužina = dužina1-dužina2+78.85
        #Udaljenost = sqrt(širina^2 + dužina^2)
        currSirina = (sirine[i] - sirine[j]) * 110.64
        currDuzina = (duzine[i] - duzine[j]) * 78.85
        distance = math.sqrt((currSirina*currSirina)+(currDuzina*currDuzina))
        if(checkForSlovenia(sirine[i],sirine[j],duzine[i],duzine[j])):
            distance = 999999
        if(checkForBosnia(sirine[i],sirine[j],duzine[i],duzine[j])):
            distance = 999999
        distanceMatrix[i][j] = distance

def evaluateInd(individual):
    fit_val = 0.0 #starting fitness is 0
    for i in range(len(individual) - 1):
        fit_val += distanceMatrix[individual[i]][individual[i + 1]]
    return fit_val,
```

Funkcija dobrote uz uvjet granice s BiH i Slovenijom

Rješenje za provjeru granica izveli smo tako što smo povukli dužinu između gradova,odnosno približno gradovima najbližih slovenskoj granici (Umag, Karlovac, Čakovec), po sljedećim koordinatama:

Umag	Karlovac	Čakovec
(45.48,13.52)	(45.48,15.37)	(46.38,16.28)

Tako su dužine koje obilježavaju granicu sa Slovenijom dužine Umag-Karlovac te Karlovac-Čakovec.

Za granicu s Bosnom i Hercegovinom smo definirali slično pravilo, dakle sve putanje (dužine koje povučemo između dva grada) koje se sijeku s granicama (dužine koje čine relacije Dubrovnik-Slunj te Slunj-Slavonski Brod) se odbacuju odnosno postavljamo vrijednost dobrote te putanje na 999999 što nadalje algoritam zanemaruje pri svom izvršavanju.

Dubrovnik	Slunj	Slavonski Brod
(42.7,18.2)	(45.14,15.7)	(45.16,18.1)

4. ZAKLJUČAK

Nakon izvršavanja genetskog algoritma, koristeći se različitim kombinacijama parametara za navedenih problem trgovačkog putnika, možemo zaključiti da se izmjenom parametara izravno utječe na vrijeme potrebno za pronalazak rješenja (što je moguće vidjeti na grafovima funkcija dobrote).

Najbolje rješenje [0, 10, 11, 4, 2, 6, 12, 7, 8, 3, 16, 17, 13, 19, 14, 20, 9, 1, 5, 18, 15] (za uvjet bez granica) dobili smo za sljedeću kombinaciju parametara :

Broj Generacija	Veličina populacije	Mutacija	Elitnih članova
5000	400	15%	20

Kada je u pitanju algoritam koji u obzir uzima granice sa Bosnom i Hercegovinom i Slovenijom tada je najbolje rješenje [15, 18, 5, 1, 9, 20, 14, 19, 13, 17, 16, 3, 8, 7, 12, 6, 2, 4, 11, 10, 0] uz sljedeću kombinaciju parametara:

Broj Generacija	Veličina populacije	Mutacija	Elitnih članova
10000	400	20%	5

Vidimo da povećanjem broja elitnih članova dobivamo malo bolja rješenja. Na razliku u vremenu znatnije utječe postotak mutacije što također vidimo iz grafova (povećanjem postotka mutacije dobijamo nešto lošija rješenja).

Promjena veličine populacije uz zadržavanje parametara mutacije i broja elitnih članova se dobija nešto bolje rješenje.