SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU ELEKTROTEHNIČKI FAKULTET

Diplomski studij računarstva

Laboratorijska vježba 2

Rješavanje problema trgovačkog putnika koristeći genetski algoritam

Ivan Budoš, DRB

Osijek, 2022.

SADRŽAJ

UVOD

U drugoj laboratorijskoj vježbi primjenjuje se genetski algoritam kako bi se riješio problem trgovačkog putnika. Vježba sadrži dva zadatka.

Prvi zadatak je da se odredi najkraći put obilaska zadanih hrvatskih gradova (problem trgovačkog putnika) od kojih svaki treba posjetiti točno jednom.

Drugi zadatak je da se prilagodi algoritam tako da trgovački putnik tokom obilaska ne napušta teritorij Hrvatske odnosno da tokom rada genetski algoritam penalizira rješenja koja prelaze granicu.

Zbog stohastičke naravi genetskog algoritma svaki eksperiment potrebno je ponoviti najmanje pet puta i zabilježiti srednji rezultat. Najbolje pronađeno rješenje od svih obavljenih mjerenja crta se na karti Hrvatske.

Parametri genetskog algoritma mijenjaju se na slijedeće vrijednosti:

Populacija : 50, 100, 200, 400

Mutacija: 5%, 10%, 15%, 20%

Broj elitnih članova: 5, 10, 15, 20

OPIS PROBLEMA TRGOVAČKOG PUTNIKA

Problem trgovačkog putnika je odrediti najkraći put obilaska gradova uz uvjet da se svaki grad posjeti samo jednom. Problem se u ovoj vježbi rješava koristeći genetski algoritam.

Opis rješenja

Do rješenja ovog problema se može doći na više načina. Jedan od mogućih načina je pohlepni algoritam. On pretragu započinje od proizvoljnog početnog grada te za svaki slijedeći grad uzima onaj koji je najbliži trenutnom i koji još nije posjećen. Takav algoritam najčešće ne daje optimalno rješenje jer ne rukuje sa svim dostupnim podacima.

Drugi način je da se koristi brute force search. On je jednostavan, ali samim time i jako spor. Radi na način da isprobava sve permutacije i pamti najbolju kombinaciju. Ovakva pretraga jako brzo postane neefikasna i može se koristiti samo za mali broj gradova.

Genetski algoritam

Genetski algoritam je meta heuristička metoda optimiranja koja imitira prirodni evolucijski proces. Evolucija je robustan proces pretraživanja prostora rješenja. Po načinu djelovanja ubrajaju se u metode usmjerenog slučajnog pretraživanja prostora rješenja (guided random search techniques) u potrazi za globalnim optimumom.

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. Najčešći tipovi kodiranja su: binarni, vrijednosni, permutacijski i stablasti.

Binarno kodiranje – gen može poprimiti samo dvije vrijednosti : 0 ili 1.
Vrijednosno kodiranje – gen može poprimiti cjelobrojne/realne vrijednosti iz zadanog intervala.
Permutacijsko kodiranje – gen može poprimiti cjelobrojne vrijednosti tako da kromosom uvijek sadrži sve brojeve ali različiti raspored.
Stablasto kodiranje – gen je čvor stabla.

Tijekom rada genetski algoritam koristi genetske operatore za stvaranje novih populacija odnosno novih rješenja. Koriste se sljedeći genetski operatori:

REKOMBINACIJA (križanje)- Kombiniranje gena dva roditelja u svrhu stvaranja novih i boljih potomaka. Najčešća rekombinacija može biti:

Rekombinacija u jednoj točki- Slučajnim odabirom točke rekombinacije kod roditelja vrši se zamjena gena od te točke na dalje. 3

Rekombinacija u dvije (više) točaka- Točke rekombinacije se biraju slučajno i nakon njih se naizmjence vrši zamjena gena.

Uniformna rekombinacija- Slučajno se odabire svaki gen da li će biti odabran iz jednog roditelja ili drugog. Odnosno svaki gen ima vjerojatnost odabira od jednog roditelja ili drugog.

MUTACIJA mijenja vrijednost nasumično odabranog gena ili više gena i na taj način unosi nove informacije u populaciju i omogućuje izlazak iz lokalnog minimuma. Najčešće se baziraju na vjerojatnosti mutacije jednog gena. Postoji više tipova:

Jednostavna mutacija- slučajno se odabire gen unutar kromosona i mijenja ga se

Potpuna mutacija- slučajno se bira kromosom i zatim se ispremješta gene u njemu.

Ovisno o kojem kodiranju gena se radi, mutacija drugačije radi. Kod binarno kodiranih invertira bitove, kod vrijednosnih kodiranja mijenja vrijednost gena za određeni interval, kod permutacijskog kodiranja odabire dva gena i mijenja im mjesta, dok kod stablastih mijenja mjesto na stablu.

Da bi genetski algoritam provodio ove prethodno navedene genetske operatore 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 – Generacijski genetski algoritam u jednoj iteraciji raspolaže sa dvije populacije (što je ujedno i nedostatak generacijskog genetskog algoritma), jer se odabiru dobre jedinke iz stare populacije koje cine novu populaciju i nakon selekcije sudjeluju u procesu reprodukcije. Vrijeme preživljavanja jedinki je točno jedna generacija.

Eliminacijske – za razliku od generacijske selekcije, eliminacijske selekcije ne bira dobre kromosone za sljedeću populaciju, već loše koje treba eliminirati i reprodukcijom ih zamijeniti sa novima. Dakle, loši kromosoni umiru a njih nadomještaju djeca nastala reprodukcijom roditelja, tj. preživjelih kromosona. Nema stroge granice među generacijama, vrijeme preživljavanja jedinke ovisi o njenoj kvaliteti.

Koristeći bilo koje od ove dvije metode selekcije postoji opasnost da se dobro rješenje dobiveno nakon puno iteracija izgubi ukoliko ga genetski operatori izmjene. Stoga se javlja potreba za mehanizmom zaštite najbolje jedinke od bilo kakve izmjene ili eliminacije tijekom evolucijskog procesa. Takav mehanizam se naziva elitizam. Genetski algoritam s ugrađenim elitizmom iz generacije u generaciju, asimptotski teži ka globalnom optimumu, odnosno rješenju problema.

Funkcija dobrote ili funkcija ocjene kvalitete jedinke se u literaturi još naziva fitness funkcija, funkcija sposobnosti, funkcija cilja ili eval funkcija i u najjednostavnijoj interpretaciji ekvivalent je funkciji f koju treba optimizirati odnosno ona predstavlja naš problem koji pokušavamo riješiti. Što je dobrota jedinke veća, jedinka ima veću vjerojatnost preživljavanja i križanja. Funkcija dobrote je ključ za proces selekcije.

Rješenje problema trgovačkog putnika primjenom genetskog algoritma

Problem trgovačkog putnika moguće je riješiti primjenom genetskog algoritma. Za njegovo rješavanje koristi se permutacijsko kodiranje jer se obilazak gradova mora odviti bez ponavljanja. U kromosome se uvrštava dvadeset i jedan gen jer je sva dvadeset i jedan grada

potrebno obići određenim redoslijedom. U fitnes funkciji se zbrajaju udaljenosti između dva susjedna gena odnosno grada da bi dobili ukupnu udaljenost odnosno dobrotu koja pripada pojedinom kromosomu. Genetski algoritam radi do maksimalne dozvoljene generacije jer na početku nije poznato rješenje odnosno optimum kojeg se želi postići. Nema unaprijed određenog cilja koji prekida evoluciju genetskog algoritma. Pri dolasku do konačne generacije ispisuje se redoslijed ili grafički prikazuje obilazak gradova i postignuta udaljenost. Kromosom se formira kao niz cjelobrojnih gena čija vrijednost odgovara rednom broju posjećenih gradova.

Implementacija koda za automatsku izmjenu parametara

Prema uputama predloška implementirane su funkcije za računanje međusobnih udaljenosti gradova koje se spremaju u 2D polje tako da element (i, j) sadrži udaljenost između i-tog i j-tog grada. Pretpostavka iz predloška je da je udaljenost između dva meridijana 78.85 km, a između dvije paralele 110,64 km. Funkcija *udaljenost()* prikazana na slici 1 računa udaljenost između gradova.

Za rješenje drugog zadatka koordinate granica pohranjene su u XML datoteku te su implementirane funkcije koje provjeravaju sijeku li se dva pravca. Prilikom obilaska gradova povlači se pravac kroz dva grada te pravac kroz pripadajuće točke granica i gleda se sijeku li se ti pravci. Za to je zadužena funkcija *sjeciste()* na slici1.

```
def udaljenost(sirina1, duzina1, sirina2, duzina2):
    udaljenostDuzina = (duzina1 - duzina2) * 78.85
    udaljenostSirina = (sirina1 - sirina2) * 110.64
    return math.sqrt(udaljenostSirina ** 2 + udaljenostDuzina ** 2)

for i in range(IND_SIZE):
    for j in range(IND_SIZE):
        udaljenostGradova[i][j] = udaljenost(sirineGradova[i],
                                             duzineGradova[i],
                                             sirineGradova[j],
                                             duzineGradova[j])

def parametriPravca(x1, y1, x2, y2):
    a = (y2 - y1) / (x2 - x1)
    b = -a * x1 + y1
    return a, b

def sjeciste(p1, p2, p3, p4):
    a1, b1 = parametriPravca(p1[0], p1[1], p2[0], p2[1])
    a2, b2 = parametriPravca(p3[0], p3[1], p4[0], p4[1])

    if a1 == a2:
        return False

    x = (b2 - b1) / (a1 - a2)
    y = a2 * x + b2

    if (min(p1[0], p2[0]) <= x <= max(p1[0], p2[0])) \
            and (min(p1[1], p2[1]) <= y <= max(p1[1], p2[1])) \
            and (min(p3[0], p4[0]) <= x <= max(p3[0], p4[0])) \
            and (min(p3[1], p4[1]) <= y <= max(p3[1], p4[1])):
        return True
    return False
```

Slika 1. Funkcije *udaljenost()*, *parametriPravca()* i *sjeciste()*

Funkcija *prelazakGranice()* provjerava dali je došlo do prelaska granice prilikom obilaska gradova te ako je, rješenje se penalizira na način da se fitness funkcija poveća za 3000 pomoću funkcije *evaluateInd()*.

```
def prelazakGranice(koorPrvogGrada, koorDrugogGrada):
    for i in range(-1, len(koordinateGranice) - 1):
        if (max(koorPrvogGrada[0], koorDrugogGrada[0]) < min(koordinateGranice[i][0], koordinateGranice[i + 1][0])) \
                or (max(koorPrvogGrada[1], koorDrugogGrada[1]) < min(koordinateGranice[i][1], koordinateGranice[i + 1][1])) \
                or (min(koorPrvogGrada[0], koorDrugogGrada[0]) > max(koordinateGranice[i][0], koordinateGranice[i + 1][0])) \
                or (min(koorPrvogGrada[1], koorDrugogGrada[1]) > max(koordinateGranice[i][1], koordinateGranice[i + 1][1])):
            continue
        else:
            if sjeciste(koorPrvogGrada, koorDrugogGrada, koordinateGranice[i], koordinateGranice[i + 1]):
                return True
    return False


for k in range(IND_SIZE):
    for l in range(IND_SIZE):
        if k == l:
            matricaPrelaskaGranica[k][l] = False
        else:
            matricaPrelaskaGranica[k][l] = prelazakGranice(koordinateGradova[k], koordinateGradova[l])
```

Slika 2. Funkcija *prelazakGranice()*

Fitness funkcija se računa zbrajanjem udaljenosti između svakog grada kako bi se dobila dobrota pojedinog kromosoma.

```
def evaluateInd(individual):
    fit_val = 0.0

    for i in range(len(individual) - 1):
        fit_val += udaljenostGradova[individual[i]][individual[i + 1]]
        if borderCheck and matricaPrelaskaGranica[individual[i]][individual[i + 1]]:
            fit_val += 3000

    return fit_val,

def generateWorldImage(individual):
    img = QImage(620, 600, QImage.Format_ARGB32)
    img.fill(Qt.transparent)

    painter = QPainter(img)
    g_first = individual[0]
    g_last = individual[IND_SIZE - 1]
    x1, y1 = GlobToImgCoords(sirineGradova[g_first], duzineGradova[g_first])
    x2, y2 = GlobToImgCoords(sirineGradova[g_last], duzineGradova[g_last])
    painter.setBrush(Qt.green)
    painter.drawEllipse(x1 - 10, y1 - 10, 15, 15)
    painter.setBrush(Qt.blue)
    painter.drawEllipse(x2 - 10, y2 - 10, 15, 15)

    painter.setPen(QPen(Qt.black, 2, Qt.DashLine))
    for i in range(IND_SIZE - 1):
        x1, y1 = GlobToImgCoords(sirineGradova[individual[i]], duzineGradova[individual[i]])
        x2, y2 = GlobToImgCoords(sirineGradova[individual[i + 1]], duzineGradova[individual[i + 1]])
        painter.drawLine(x1, y1, x2, y2)

    painter.setPen(QPen(Qt.red, 2, Qt.DotLine))
    for i in range(len(sirineGranica)):
        if i == (len(sirineGranica) - 1):
            x1, y1 = GlobToImgCoords(sirineGranica[i], duzineGranica[i])
            x2, y2 = GlobToImgCoords(sirineGranica[0], duzineGranica[0])
        else:
            x1, y1 = GlobToImgCoords(sirineGranica[i], duzineGranica[i])
            x2, y2 = GlobToImgCoords(sirineGranica[i + 1], duzineGranica[i + 1])
        painter.drawLine(x1, y1, x2, y2)

    painter.end()

    return img
```

Slika 3. Računanje fitness funkcije te iscrtavanje putanje i granica

REZULTATI

Prvi zadatak zahtijevao je određivanje najkraćeg puta obilaska zadanih hrvatskih gradova uz uvjet da se svaki grad treba posjetiti točno jednom.

Drugi zadatak zahtijevao je prilagođavanje algoritma na način da trgovački putnik ne napušta teritorij Hrvatske odnosno da genetski algoritam penalizira rješenje koje prelazi granicu.

Zbog stohastičke naravi genetskog algoritma svaki eksperiment potrebno je ponoviti najmanje pet puta i zabilježiti srednji rezultat. Najbolje pronađeno rješenje od svih obavljenih mjerenja crta se na karti Hrvatske.

Parametri genetskog algoritma mijenjaju se na slijedeće vrijednosti:

Populacija : 50, 100, 200, 400

Mutacija: 5%, 10%, 15%, 20%

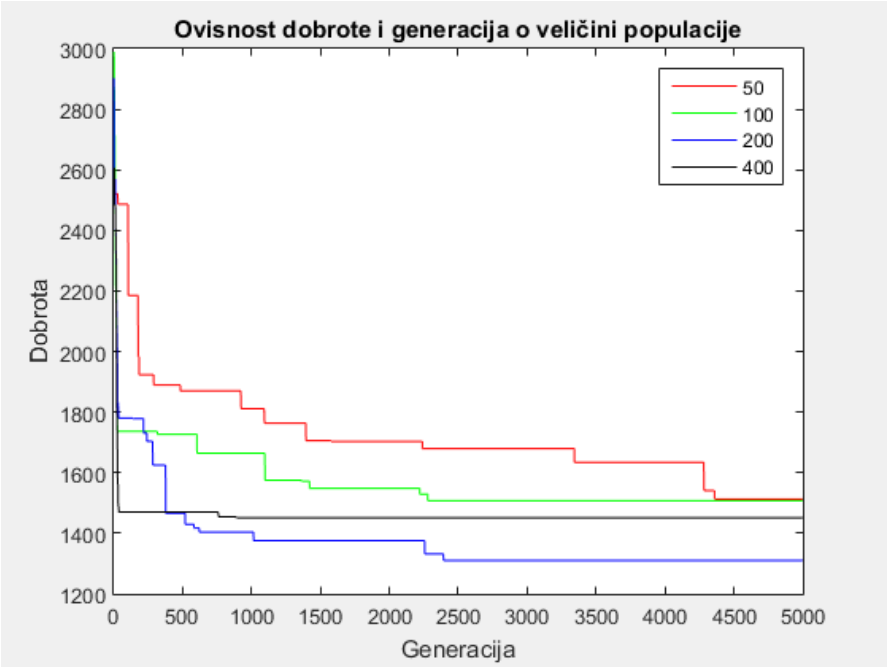
Broj elitnih članova: 5, 10, 15, 20

BEZ UVJETA ZA GRANICE

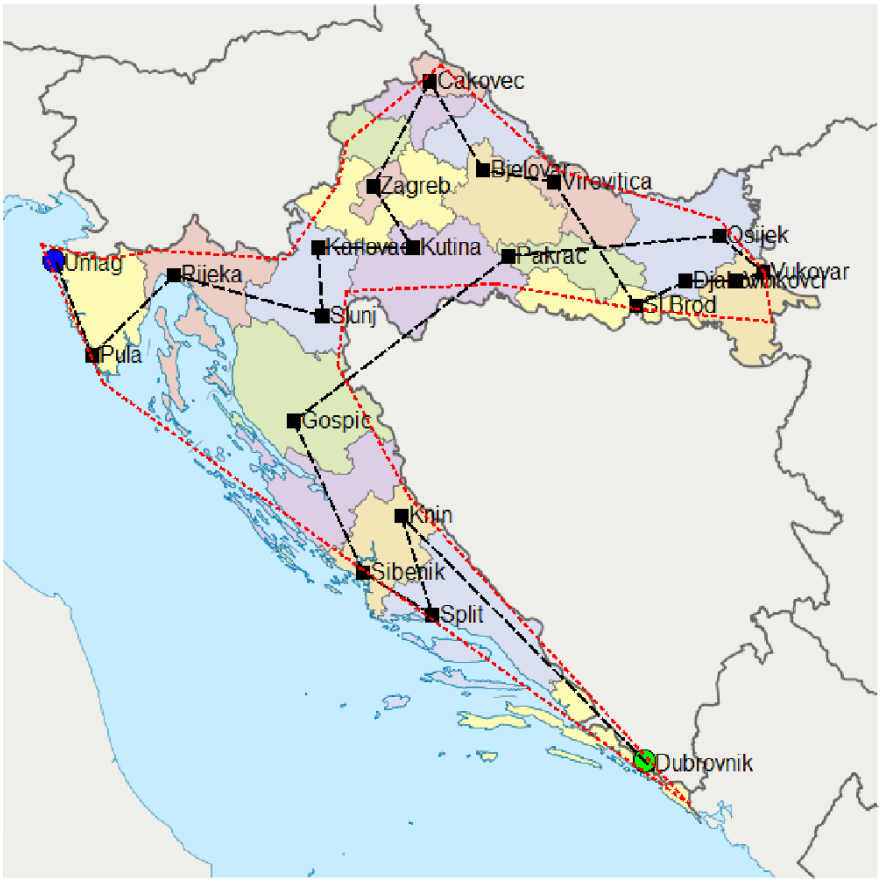
OVISNOST O VELIČINI POPULACIJE

Broj generacija	5000			
Broj elitnih članova	5			
Postotak mutacije[%]	5%			
Veličina populacije	50	100	200	400
Rješenje	1785.99, 1885.36, 1462.48, 1658.28, 1512.66	1524.55, 1658.56, 1489.56, 1530.04, 1603.84	1736.70, 1644.33, 1373.21, 1329.51, 1309.96	1418.55, 1323.49, 1600.30, 1501.33, 1450.96
Najkraći put	1462.48	1489.56	1309.96	1323.49
Prosječan najkraći put (medijan)	1658.28	1530.04	1373.21	1450.96

Tablica 1. Ovisnost o veličini populacije

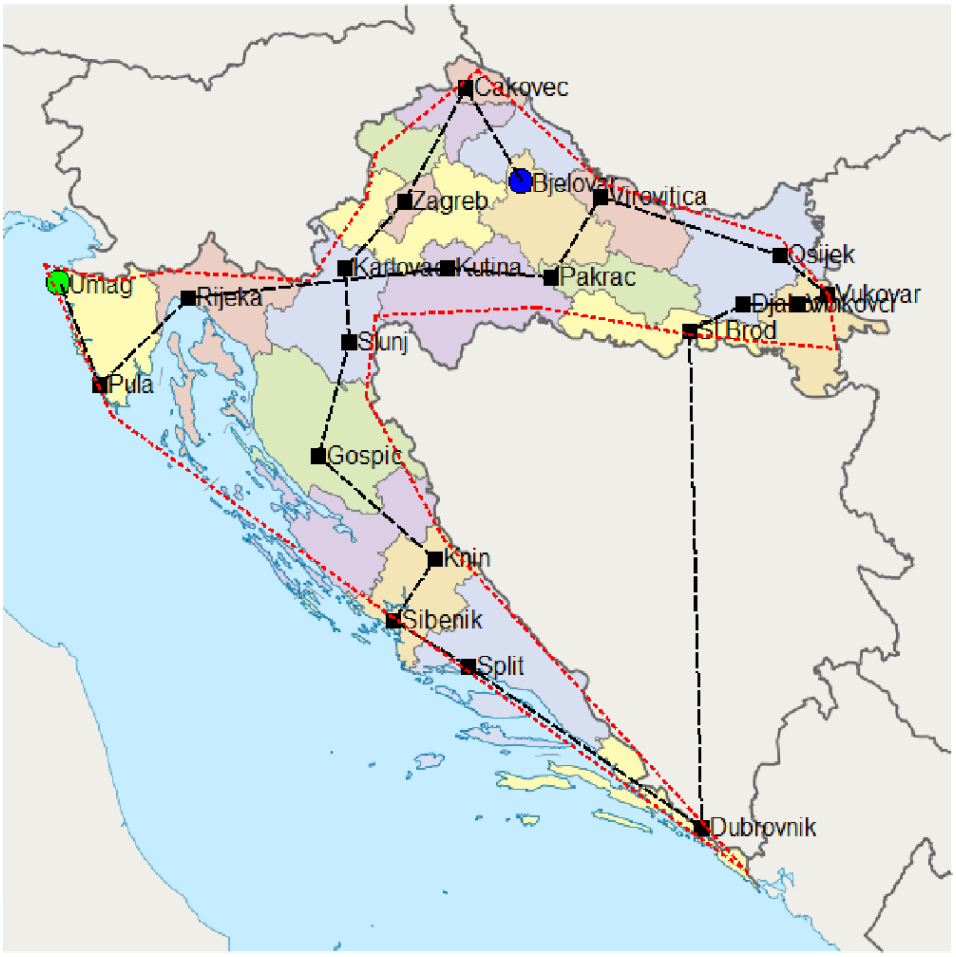


Slika 4. Prikaz ovisnosti dobrote i generacija o veličini populacije



Slika 5. Najbolje rješenje za populaciju 50

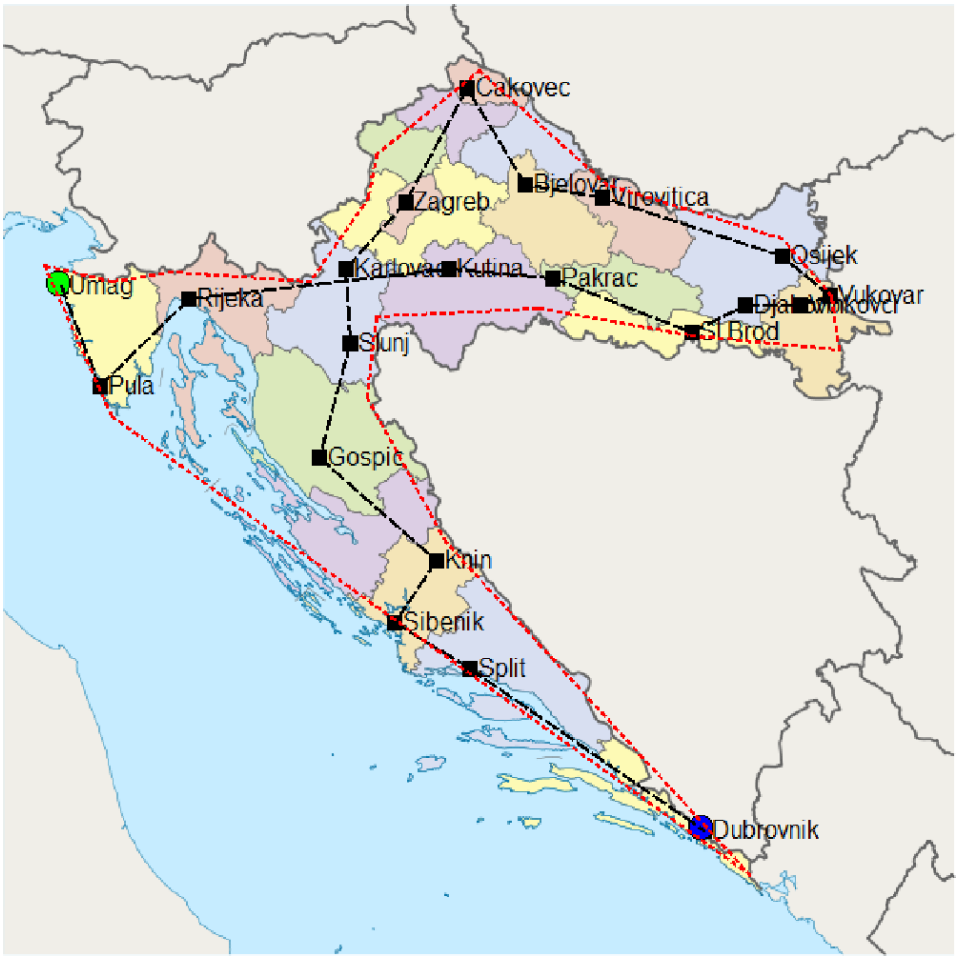
Najbolje rješenje = [0, 4, 10, 11, 2, 20, 5, 15, 18, 1, 9, 14, 19, 13, 17, 16, 3, 8, 7, 6, 12] (1462.48km)



Slika 6. Najbolje rješenje za populaciju 100

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

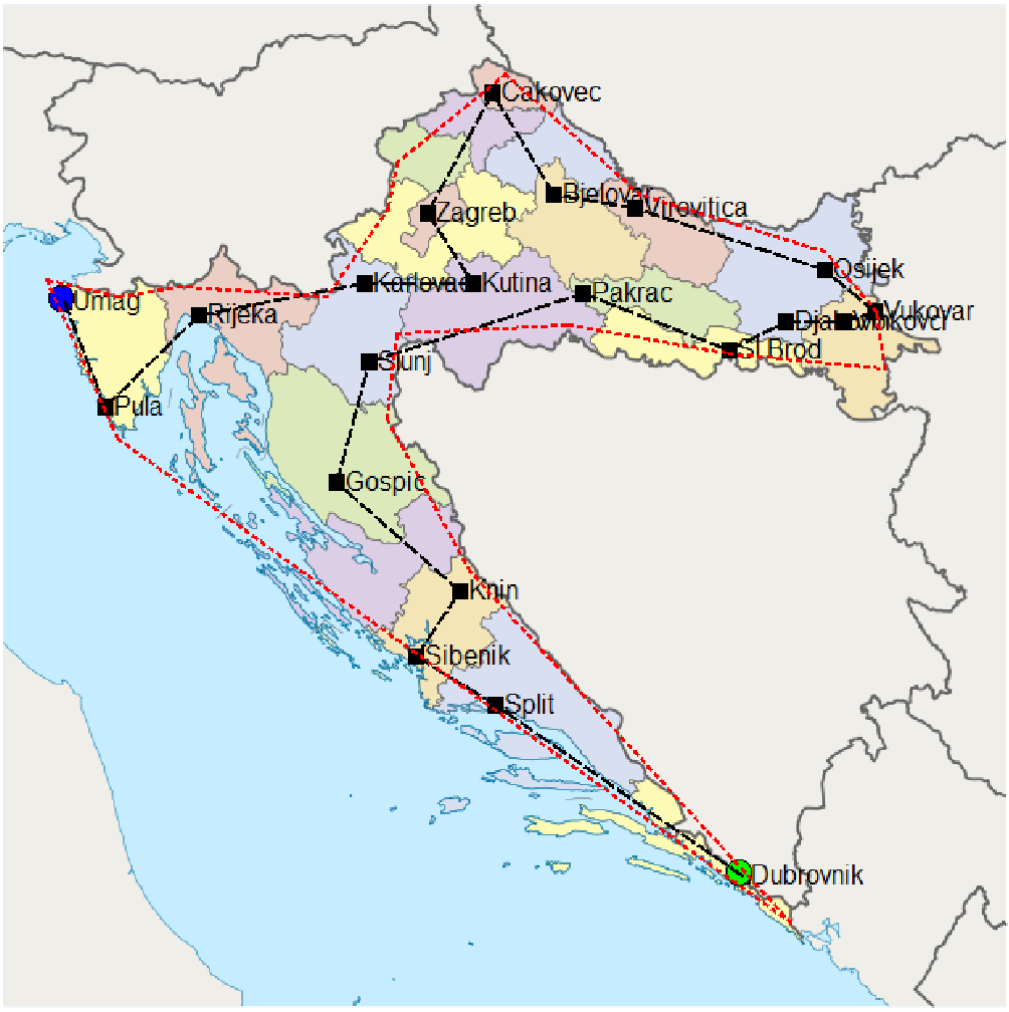
(1504.54km)



Slika 7. Najbolje rješenje za populaciju 200

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

(1309.96km)



Slika 8. Najbolje rješenje za populaciju 400

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

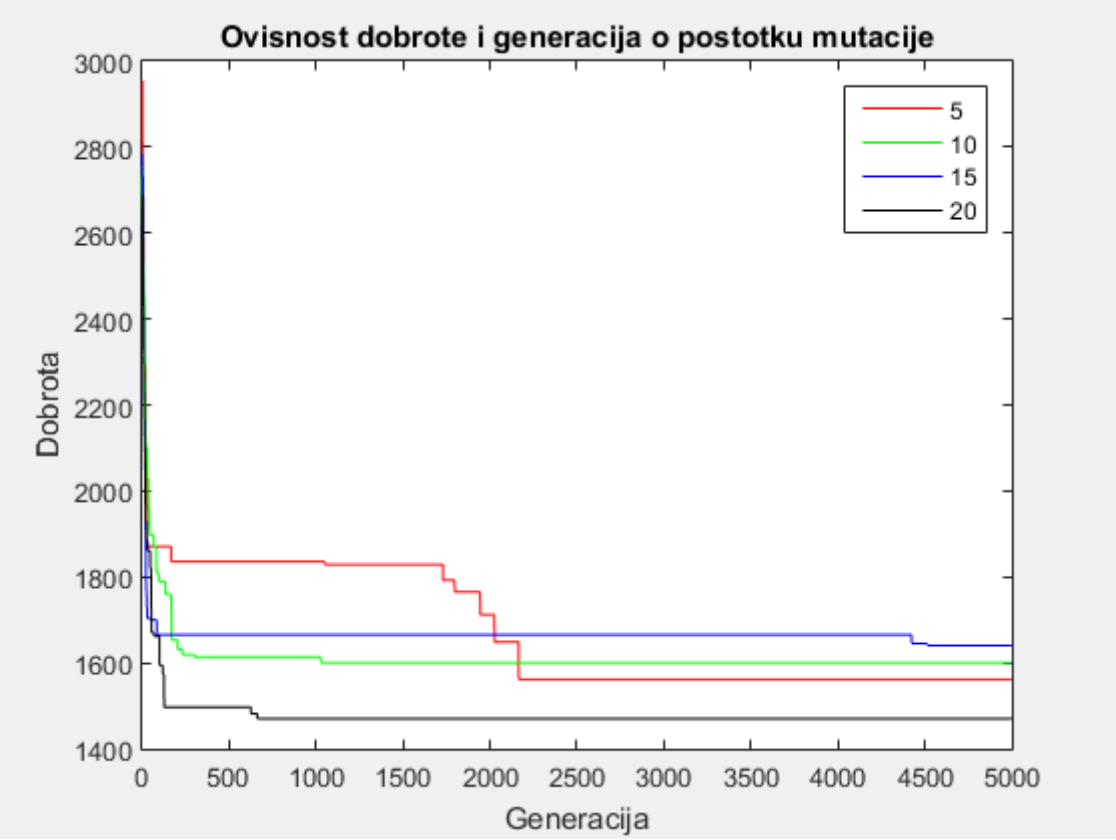
(1323.49km)

OVISNOST O POSTOTKU MUTACIJE

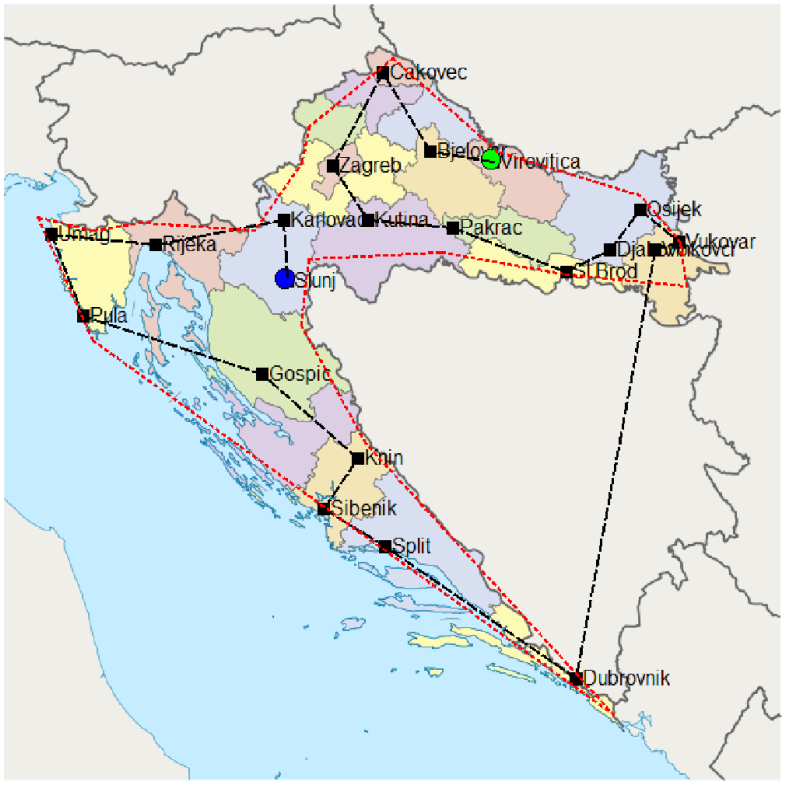
Broj generacija	5000			
Broj elitnih članova	5			
Populacija	100			
Mutacija	5%	10%	15%	20%
Rješenje	1524.55, 1658.56, 1489.56, 1530.04, 1603.84	1480.38, 1527.56, 1491, 1396.04, 1599.78	1489.66, 1433.94, 1297.33, 1522.21, 1640.32	1585, 1572.29, 1635.81, 1553.74, 1470.15

Najkraći put	1489.56	1396.04	1297.33	1470.15
Prosječan najkraći put (medijan)	1530.04	1491	1489.66	1572.29

Tablica 2. Ovisnost o postotku mutacije



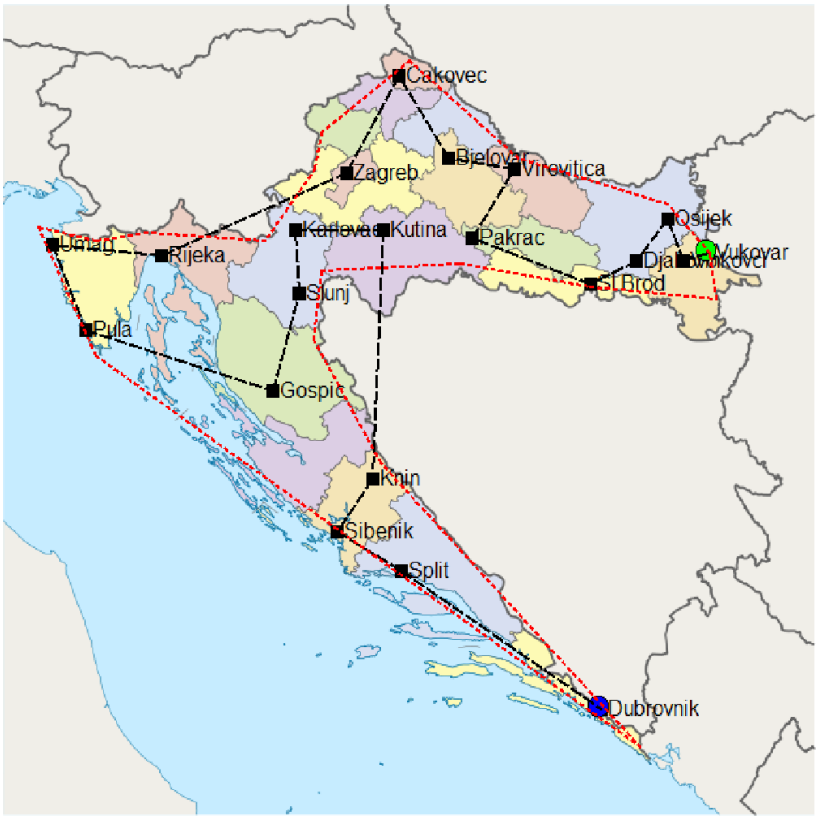
Slika 9. Prikaz ovisnosti dobrote i generacija o postotku mutacije



Slika 10. Najbolje rješenje za mutaciju 5%

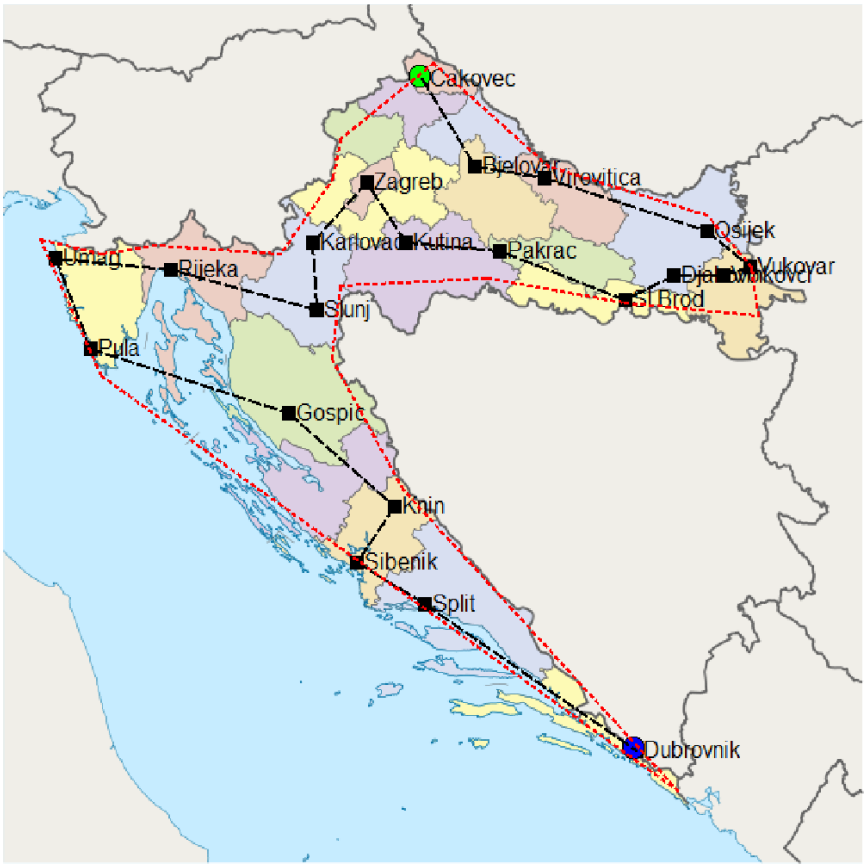
Najbolje rješenje = [14, 19, 13, 17, 16, 20, 9, 1, 5, 15, 18, 0, 10, 11, 4, 2, 6, 12, 7, 3, 8]

(1504.54km)



Slika 11. Najbolje rješenje za mutaciju 10%

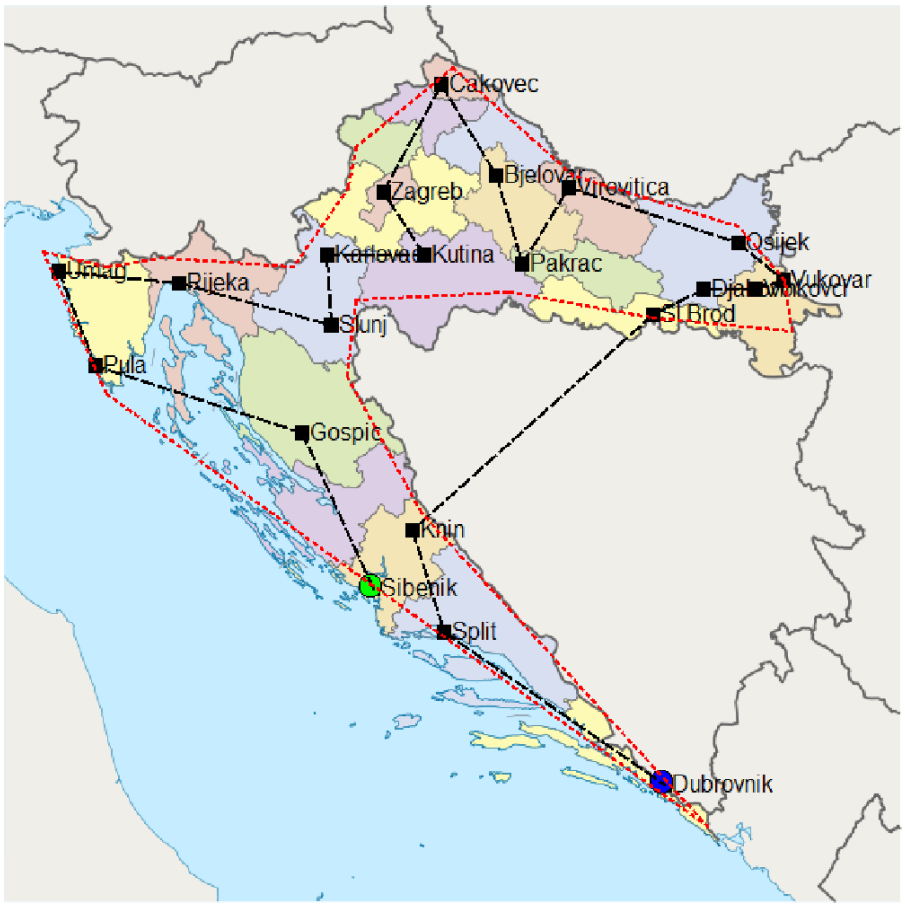
Najbolje rješenje = [15, 18, 5, 1, 9, 20, 14, 19, 13, 17, 7, 12, 6, 2, 8, 3, 16, 4, 11, 10, 0]
(1396.04km)



Slika 12. Najbolje rješenje za mutaciju 15%

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

(1297.33km)



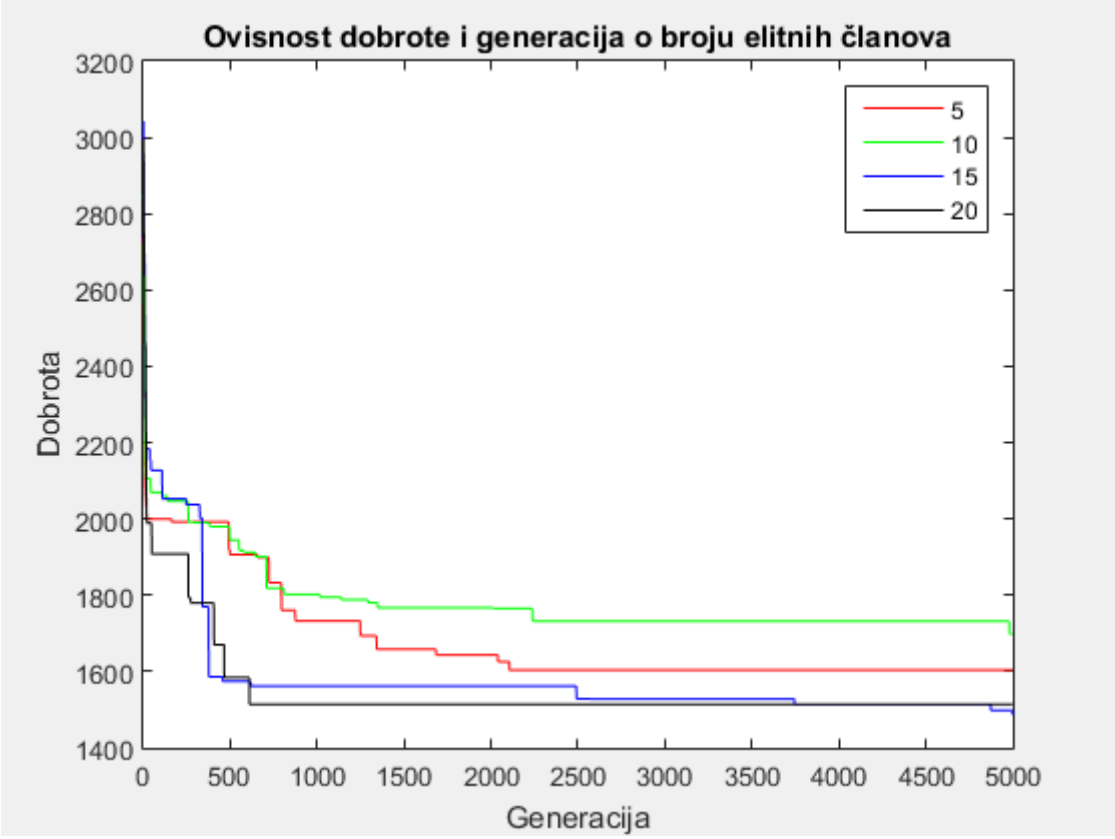
Slika 13. Najbolje rješenje za mutaciju 20%

Najbolje rješenje = [11, 2, 6, 12, 7, 8, 3, 16, 17, 13, 19, 20, 14, 5, 15, 18, 1, 9, 4, 10, 0] (1470.15km)

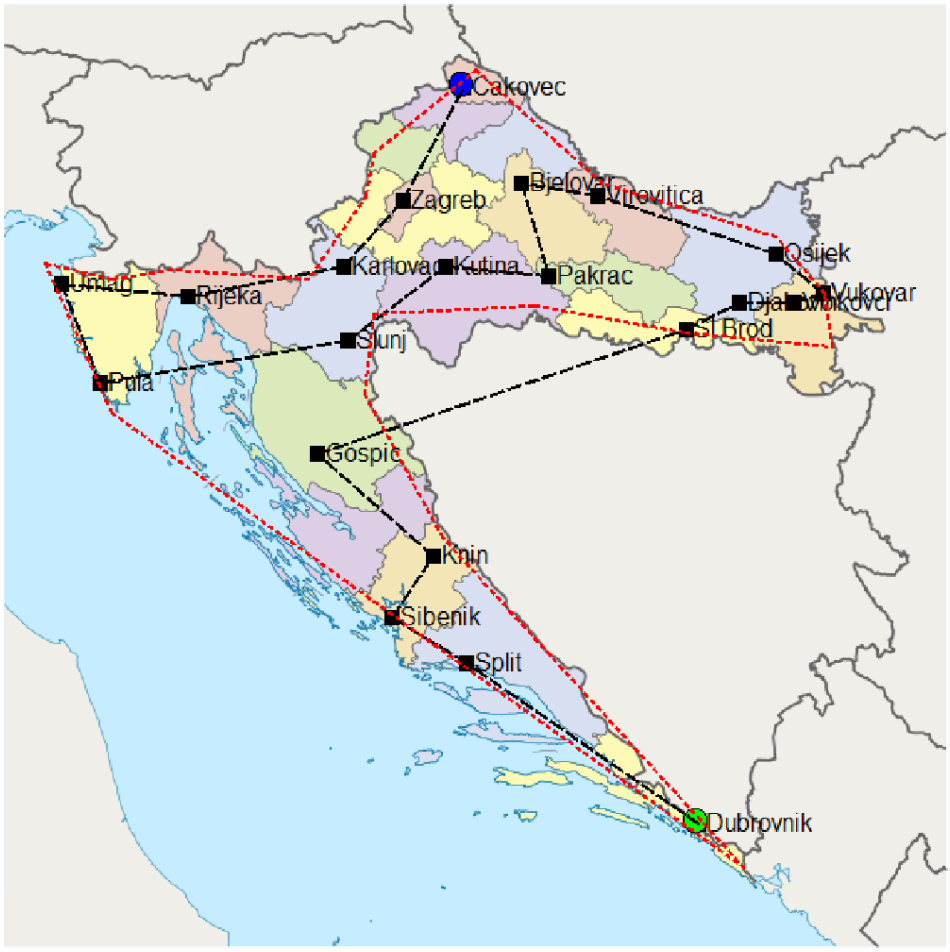
OVISNOST O BROJU ELITNIH ČLANOVA

Broj generacija	5000			
Populacija	100			
Postotak mutacije[%]	5%			
Broj elitnih članova	5	10	15	20
Rješenje	1524.55, 1658.56, 1489.56, 1530.04, 1603.841	1515.41, 1558.13, 1513.77, 1683.48, 1698.19	1528.08, 1581.94, 1616.88, 1557.56, 1486.79	1652.70, 1727.68, 1565.51, 1628.30, 1513.96
Najkraći put	1489.56	1513.77	1486.79	1513.96
Prosječan najkraći put (medijan)	1530.04	1558.13	1557.56	1628.30

Tablica 3. Ovisnost o broju elitnih članova



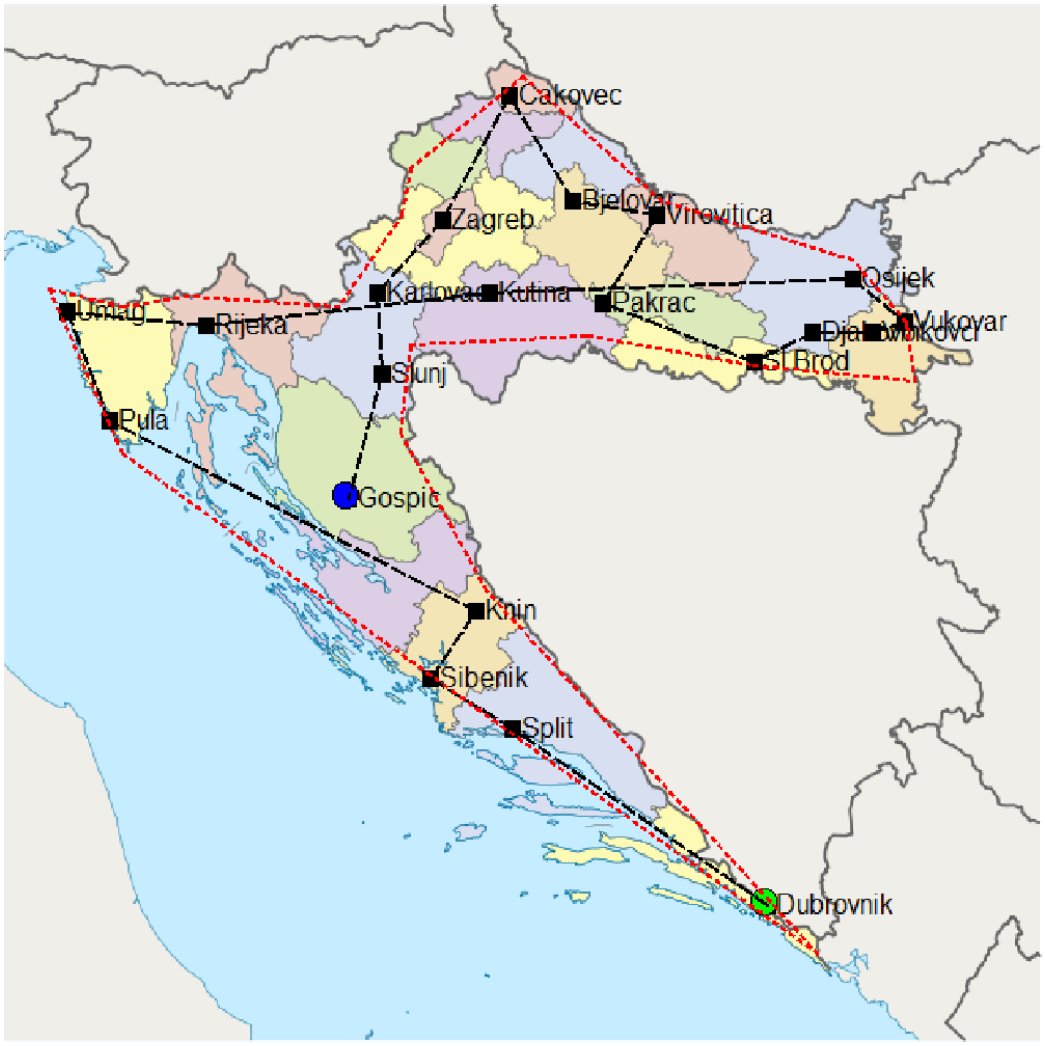
Slika 14. Prikaz ovisnosti dobrote i generacija o broju elitnih članova



Slika 15. Najbolje rješenje za broj elitnih članova 5

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

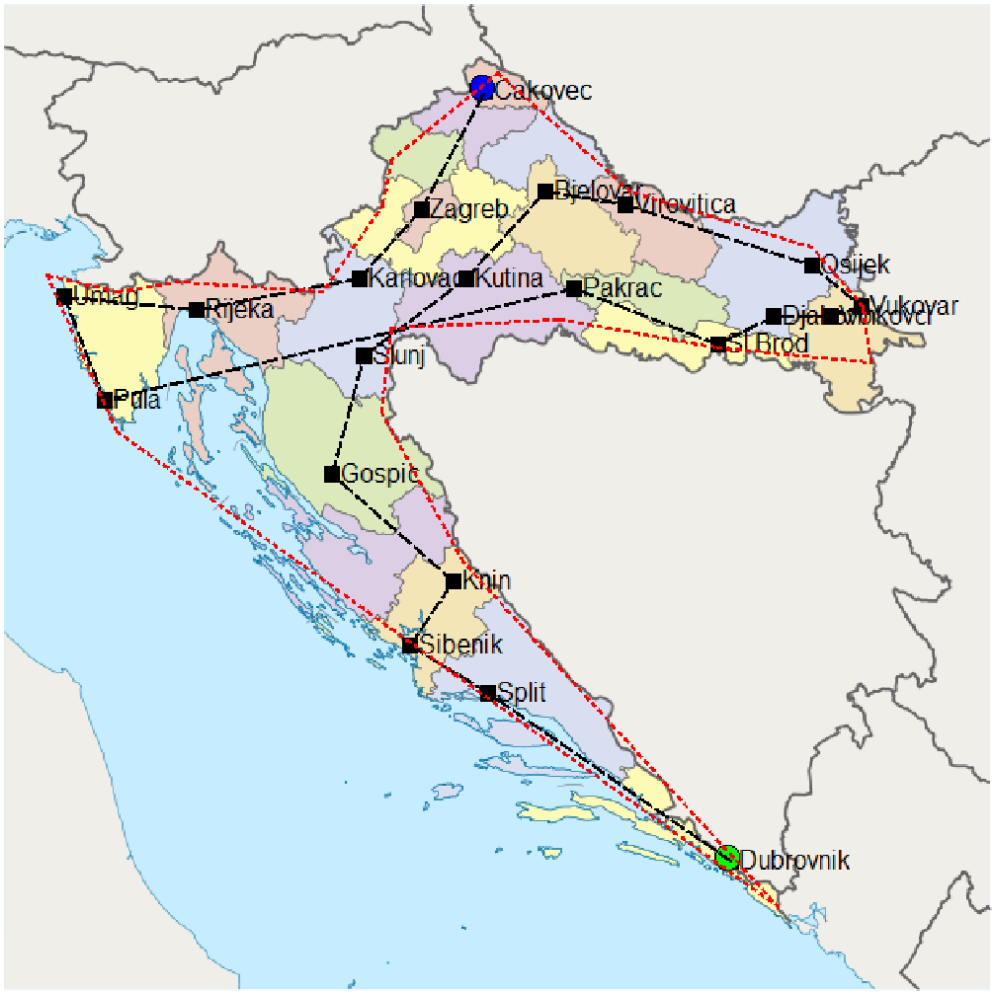
(1489.56km)



Slika 16. Najbolje rješenje za broj elitnih članova 10

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

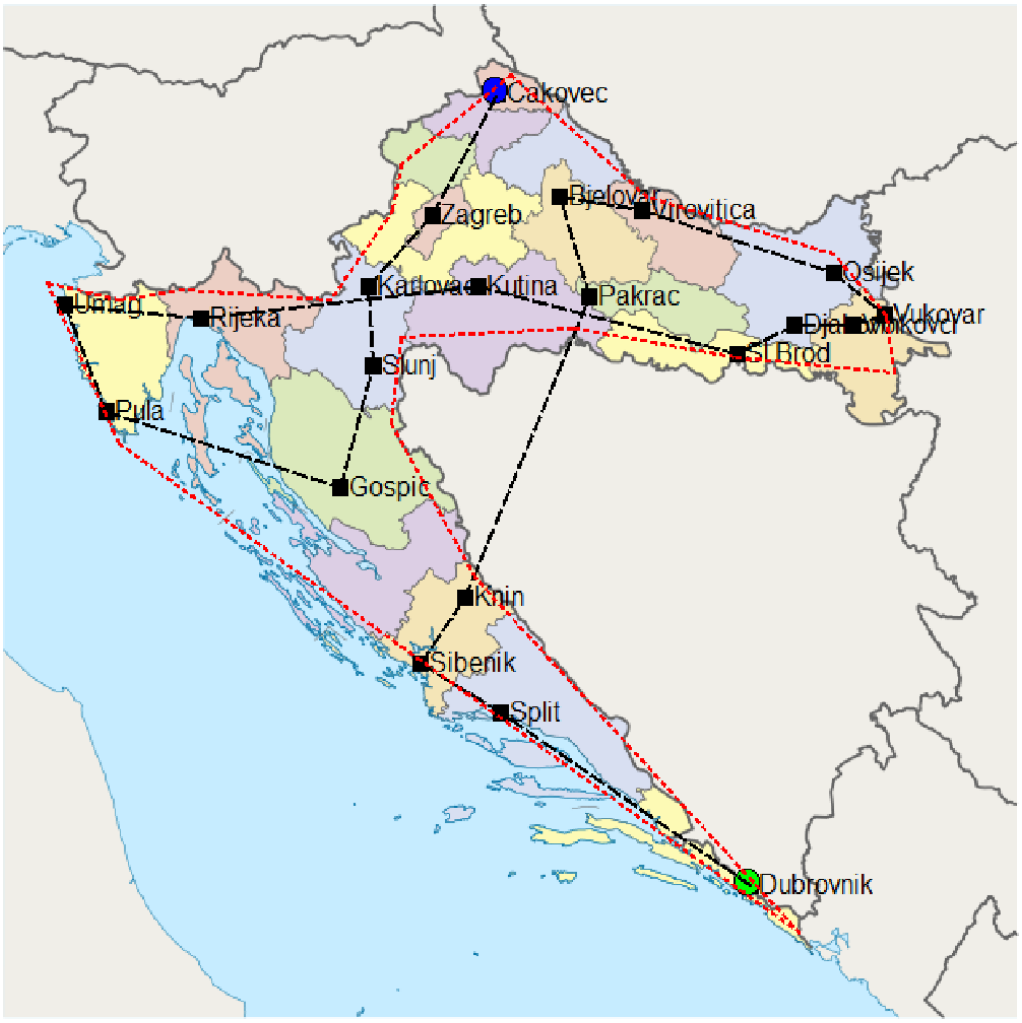
(1513.77km)



Slika 17. Najbolje rješenje za broj elitnih članova 15

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

(1486.79km)



Slika 18. Najbolje rješenje za broj elitnih članova 20

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

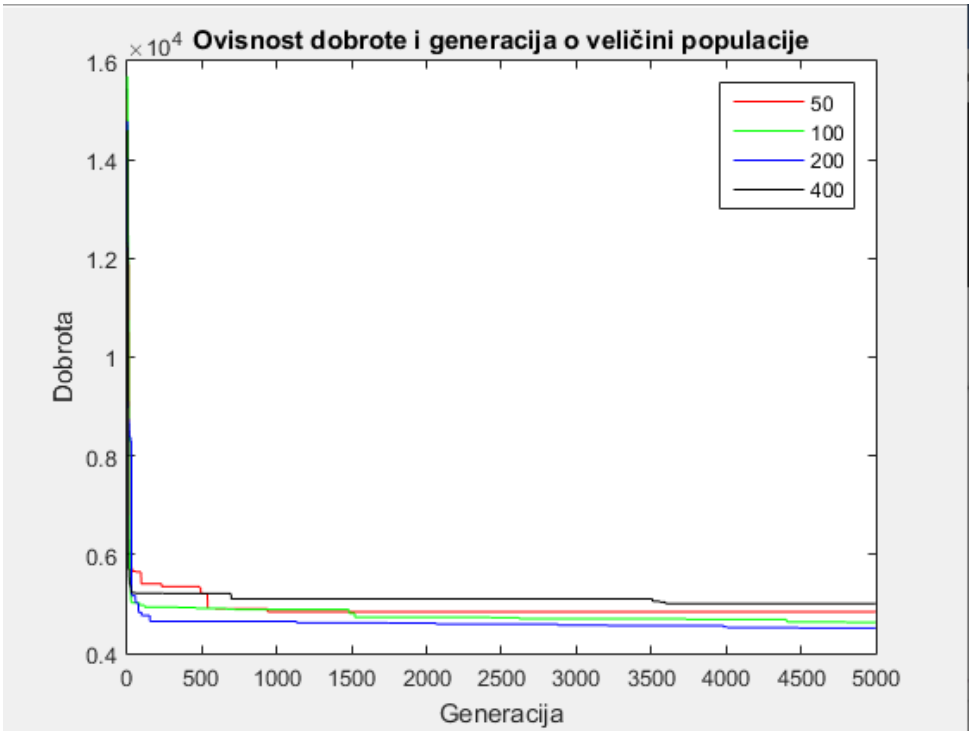
(1513.96km)

UZ UVJET GRANICE

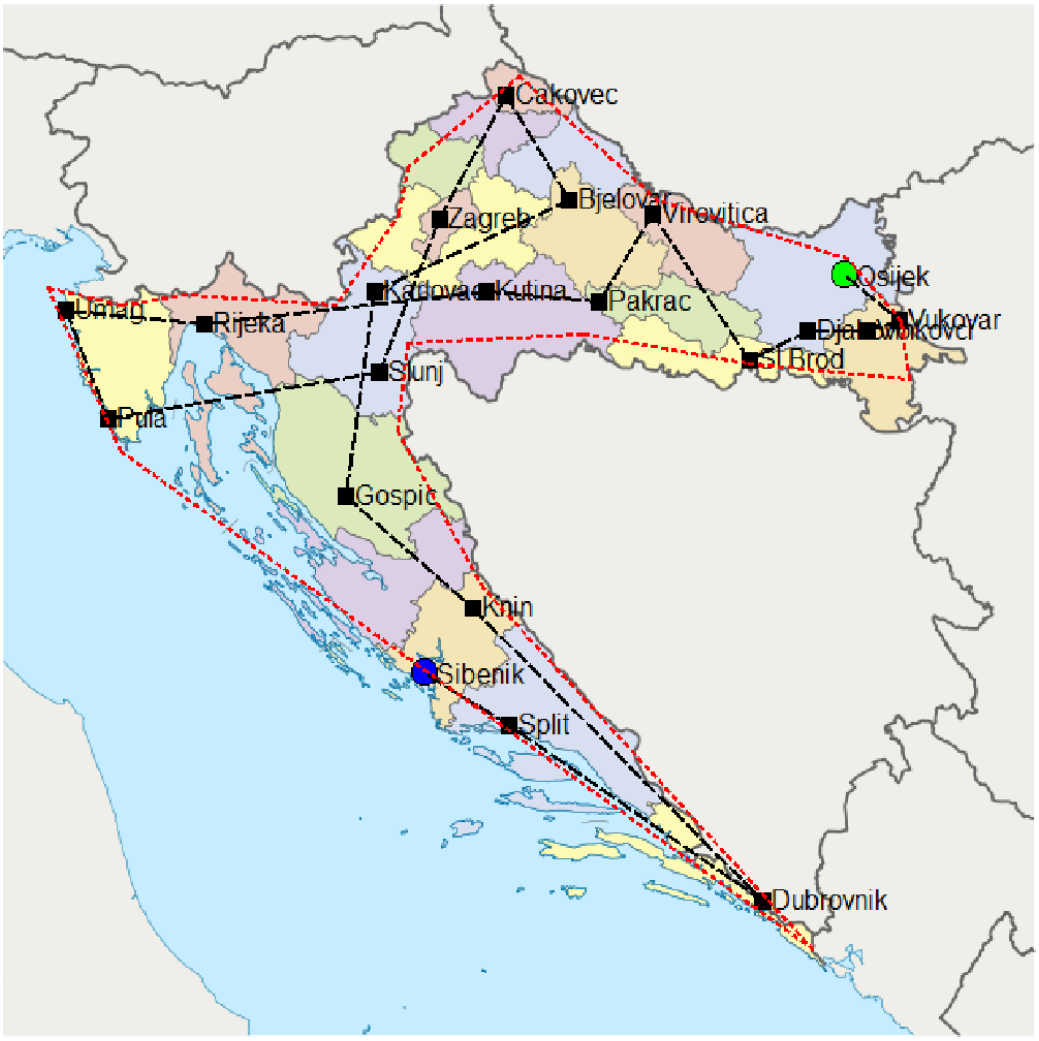
OVISNOST O VELIČINI POPULACIJE

Broj generacija	5000			
Broj elitnih članova	5			
Postotak mutacije[%]	5%			
Veličina populacije	50	100	200	400
Rješenje	5750.74, 4675.87, 8025.26, 5220.705, 4838.341	4802.76, 4751.58, 5175.35, 4969.27, 4625.29	4705.98, 4637.563, 5366.18, 5069.38, 4509.82	4735.91, 4783.49, 4681.35, 4735.91, 5003.96
Najkraći put	4675.87	4625.29	4509.82	4681.35
Prosječan najkraći put (medijan)	5220.7	4802.76	4705.98	4735.91

Tablica 4. Ovisnost o veličini populacije

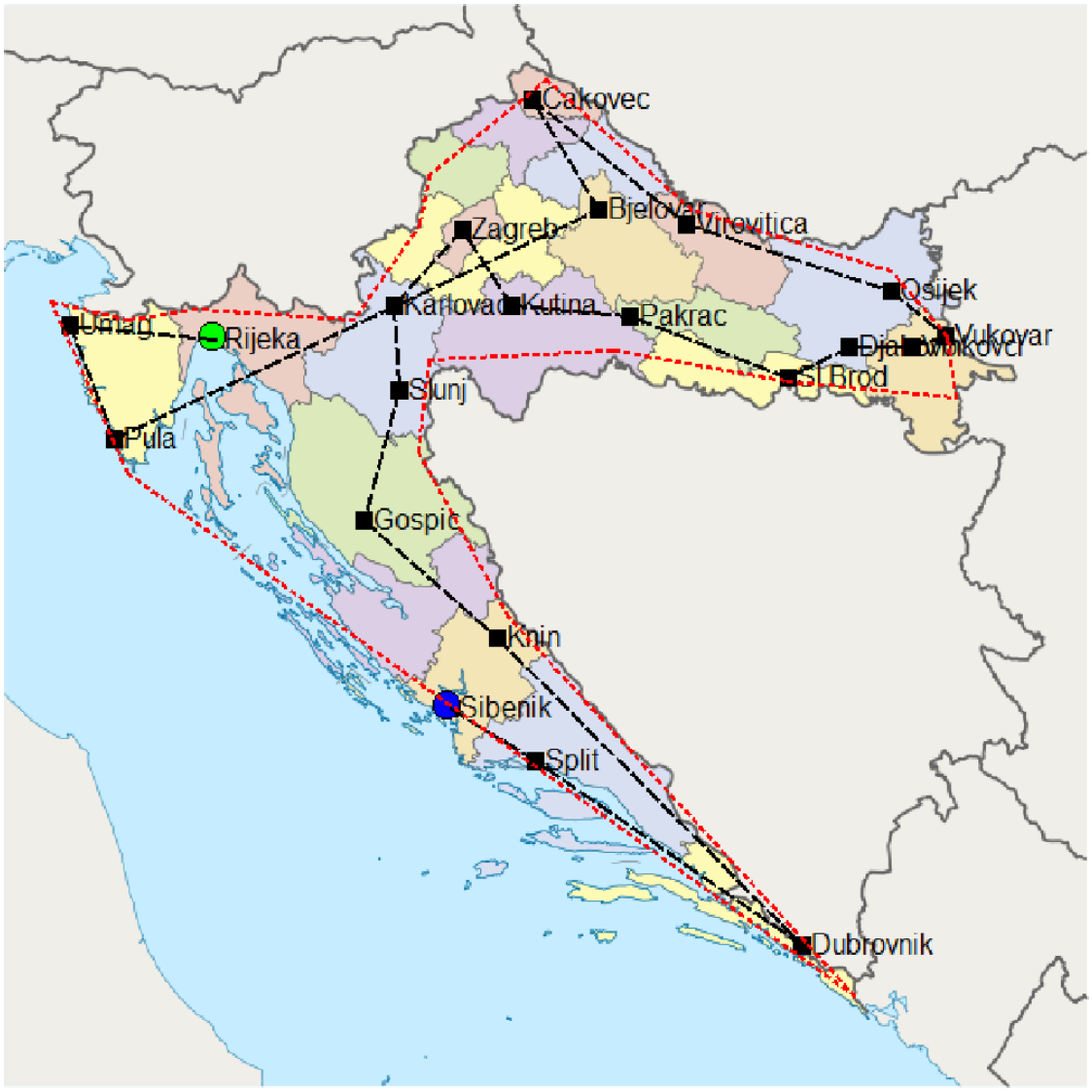


Slika 19. Prikaz ovisnosti dobrote i generacija o veličini populacije



Slika 20. Najbolje rješenje za populaciju 50

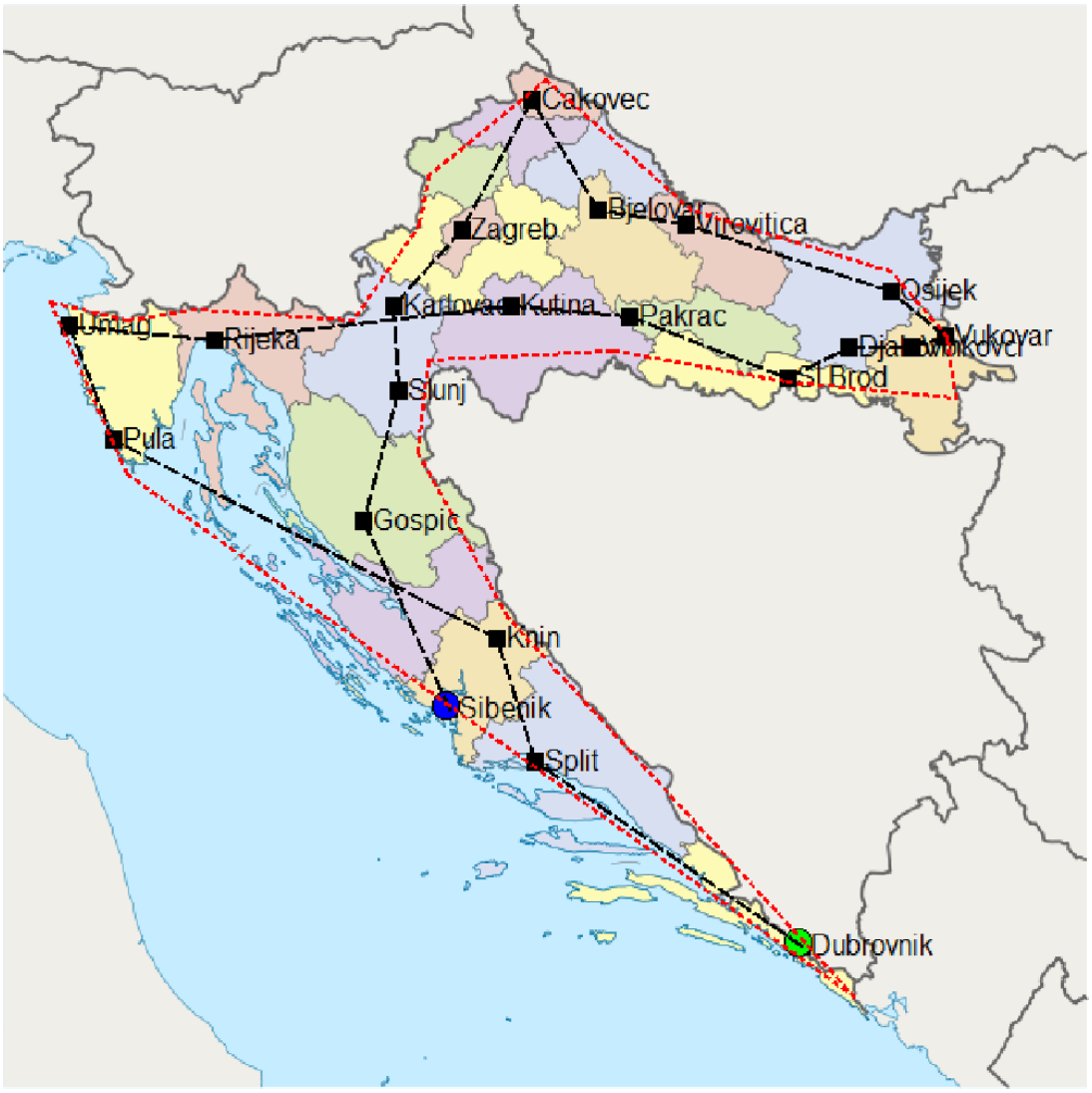
Najbolje rješenje = [5, 15, 18, 1, 9, 14, 20, 16, 7, 12, 6, 8, 17, 13, 19, 3, 2, 4, 0, 10, 11] (4675.87km)



Slika 21. Najbolje rješenje za populaciju 100

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

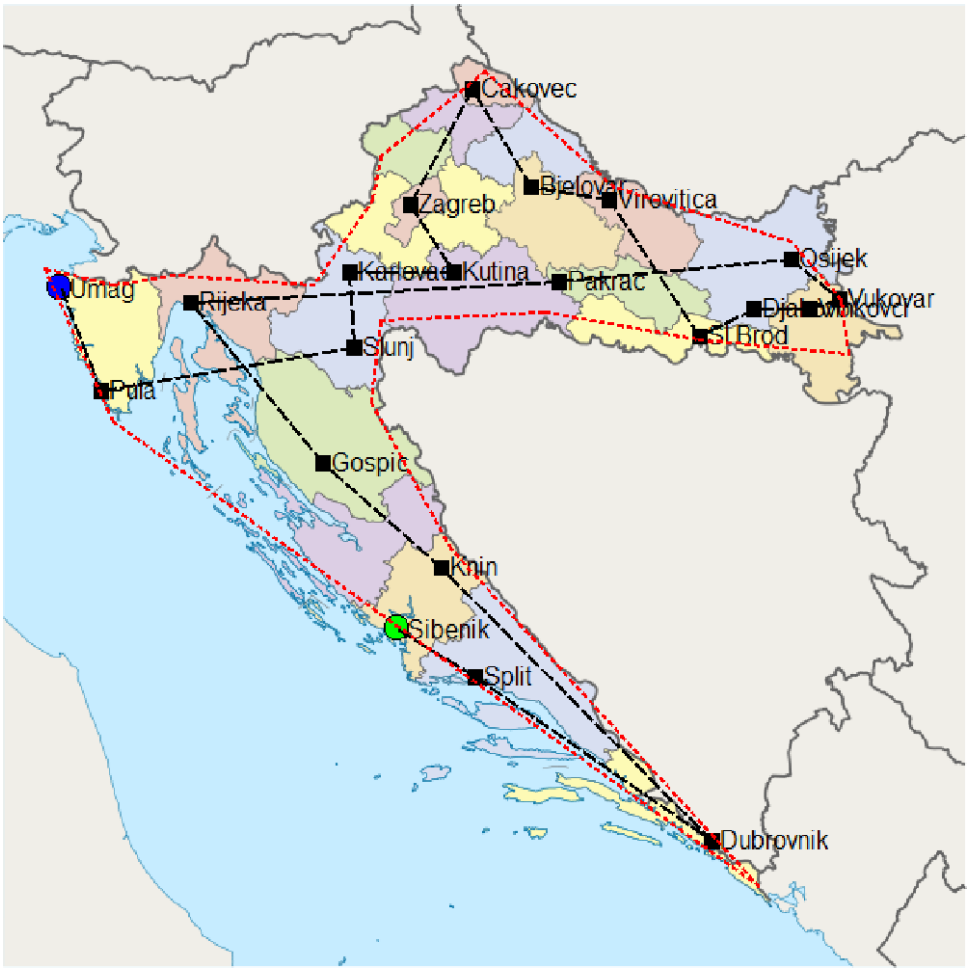
(4625.29km)



Slika 22. Najbolje rješenje za populaciju 200

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

(4509.82km)



Slika 23. Najbolje rješenje za populaciju 400

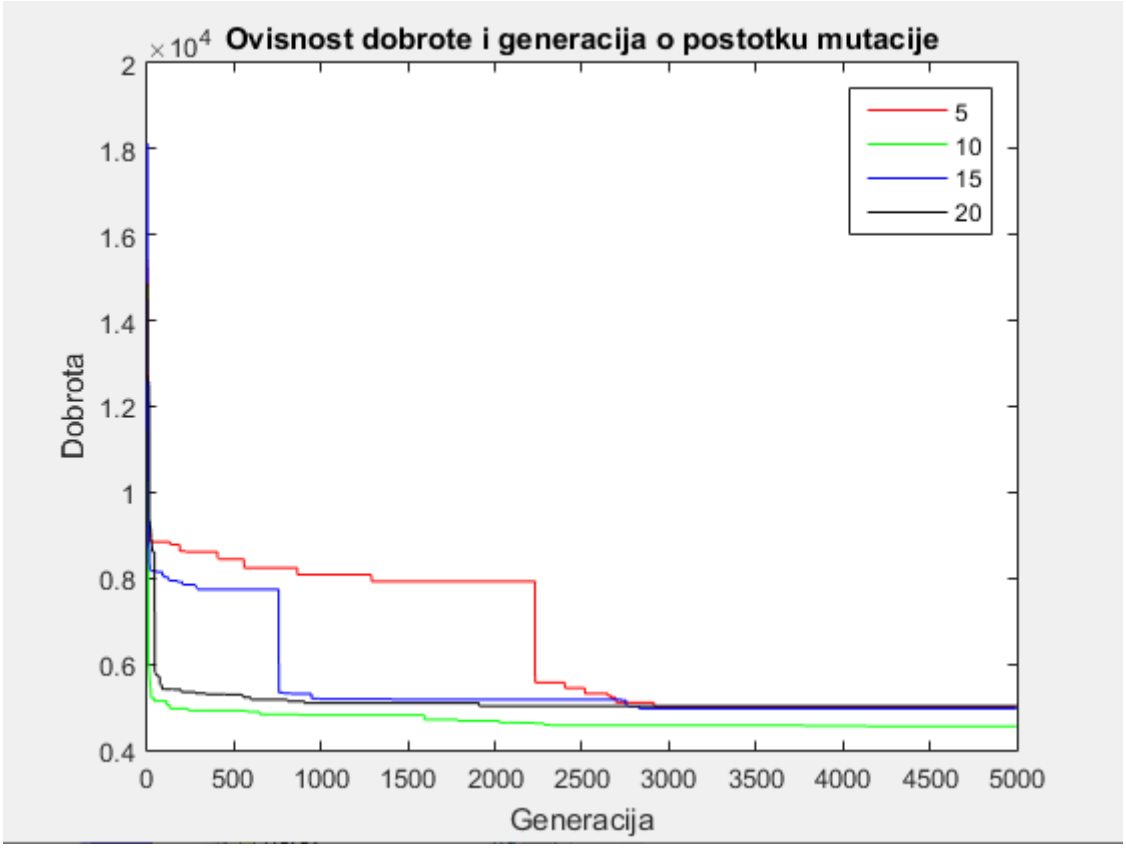
Najbolje rješenje = [11, 10, 0, 4, 2, 7, 20, 5, 15, 18, 1, 9, 14, 19, 13, 17, 16, 3, 8, 6, 12]

(4681.35km)

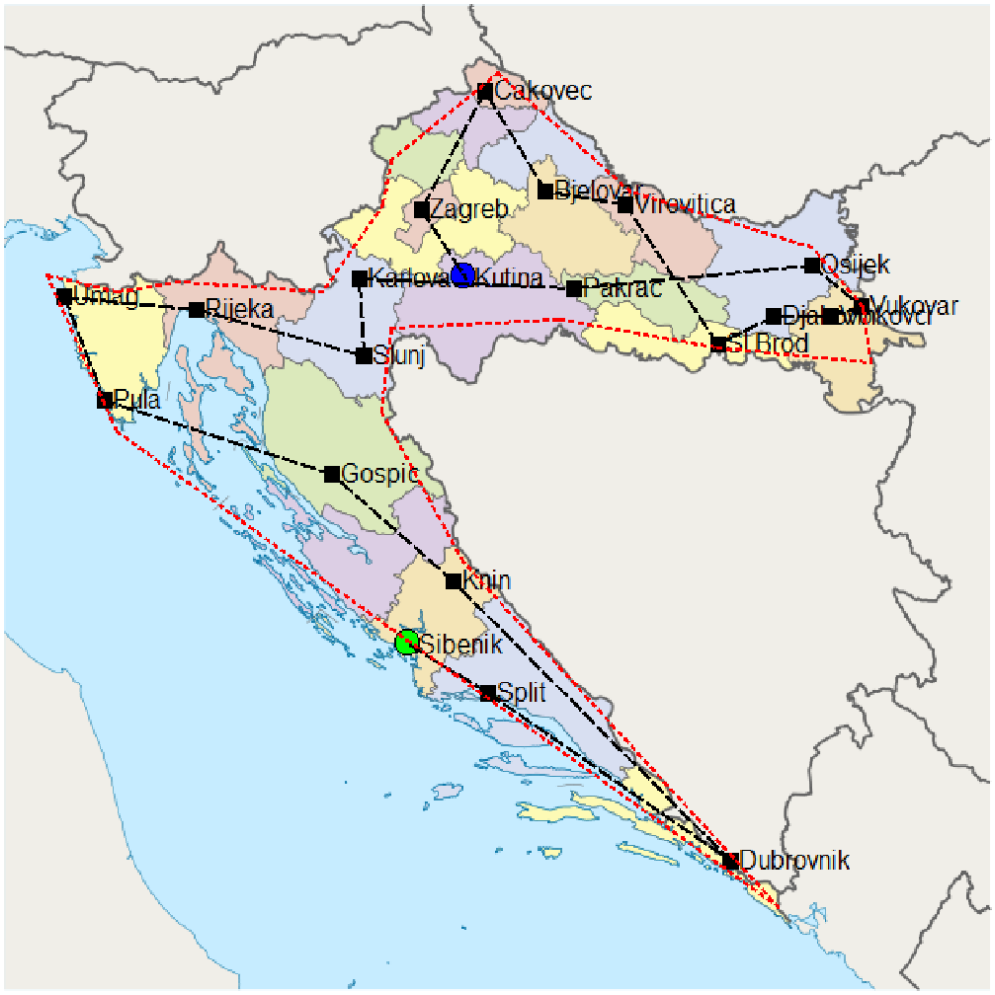
OVISNOST O POSTOTKU MUTACIJE

Broj generacija	5000			
Broj elitnih članova	5			
Populacija	100			
Mutacija	5%	10%	15%	20%
Rješenje	5171.77, 7706.88, 4664.60, 4574.98, 5005.04	4874.75, 4618.27, 4479.46, 4883.32, 4546.61	4760.54, 4715.67, 4917.74, 4546.61, 4959.37	4774.12, 4463.48, 4896.29, 4495.05, 5019.57
Najkraći put	4574.98	4479.46	4546.61	4463.48
Prosječan najkraći put (medijan)	5005.04	4618.27	4760.54	4774.12

Tablica 5. Ovisnost o postotku mutacije



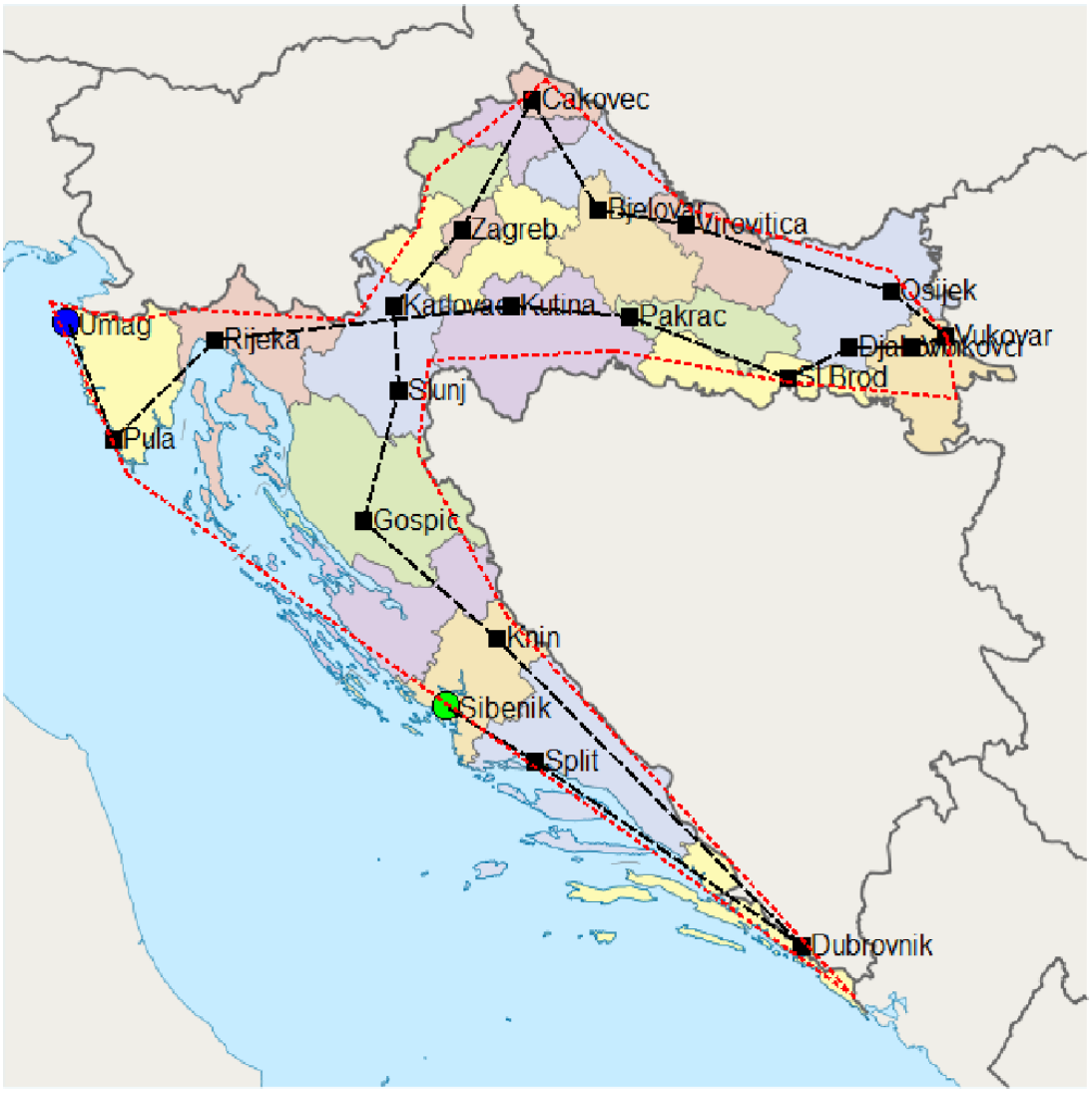
Slika 24. Prikaz ovisnosti dobrote i generacija o postotku mutacije



Slika 25. Najbolje rješenje za mutaciju 5%

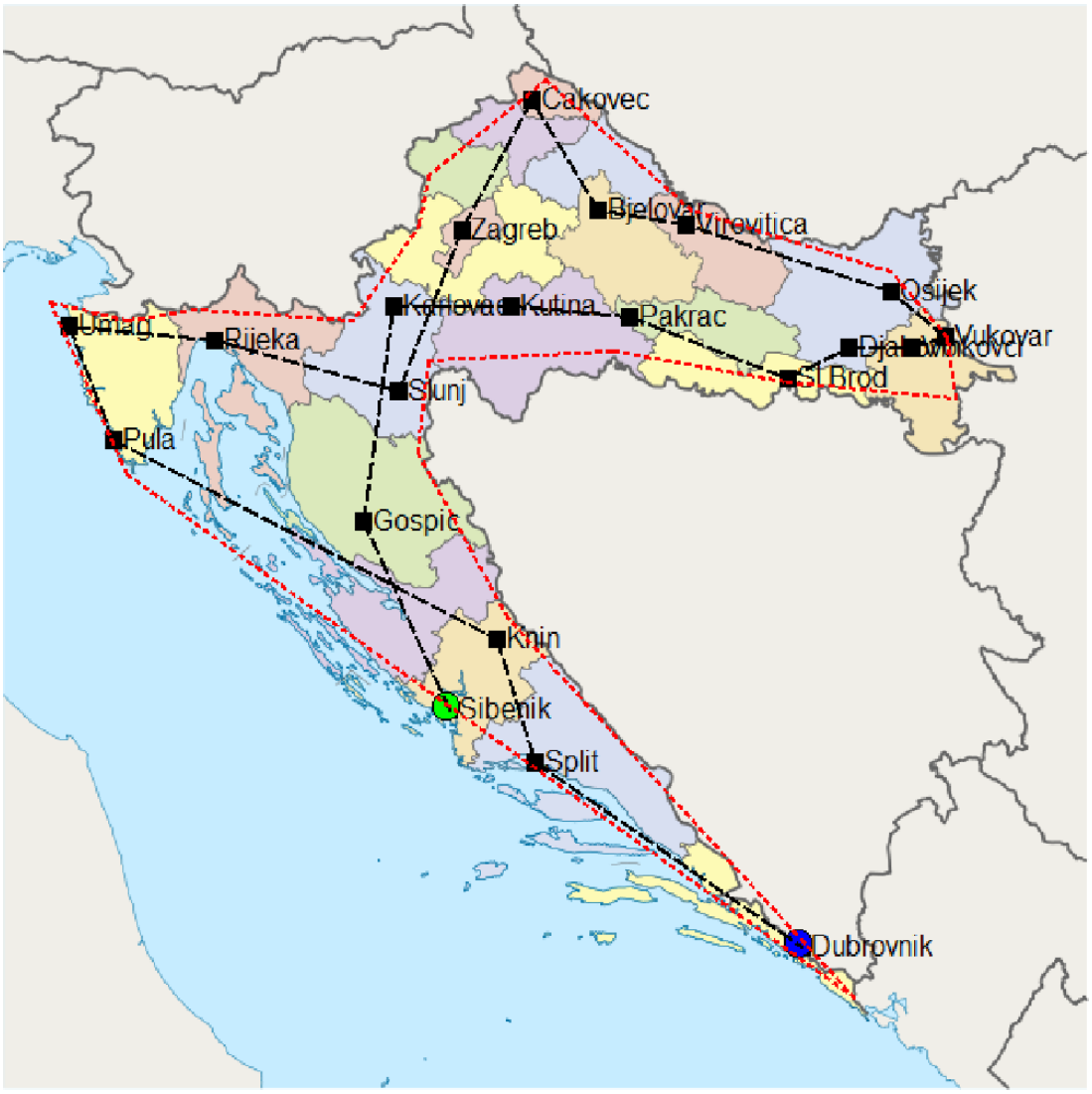
Najbolje rješenje = [11, 10, 0, 4, 2, 6, 12, 7, 8, 3, 20, 5, 15, 18, 1, 9, 14, 19, 13, 17, 16]

(4574.98km)



Slika 26. Najbolje rješenje za mutaciju 10%

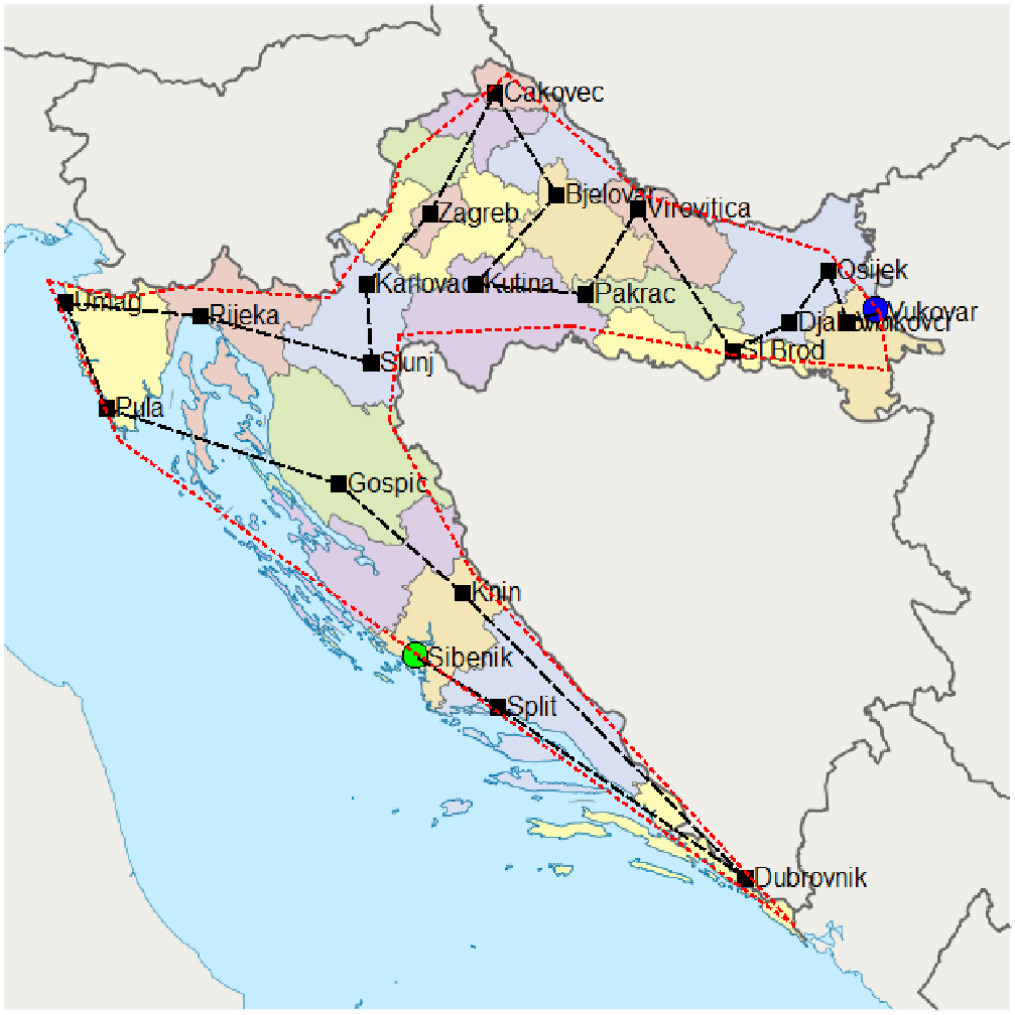
Najbolje rješenje = [11, 10, 0, 4, 2, 8, 3, 17, 13, 19, 14, 5, 15, 18, 1, 9, 20, 16, 7, 6, 12] (4479.46km)



Slika 27. Najbolje rješenje za mutaciju 15%

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

(4546.61km)



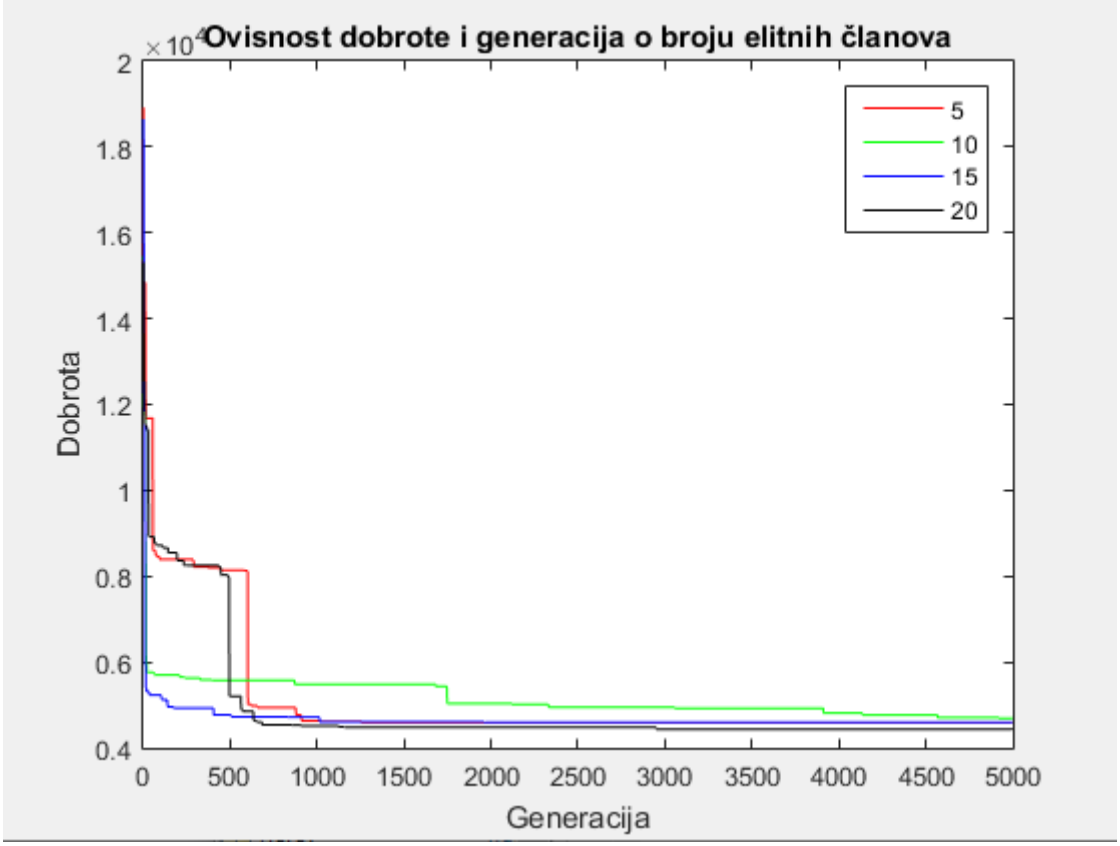
Slika 28. Najbolje rješenje za mutaciju 20%

Najbolje rješenje = [11, 10, 0, 4, 2, 6, 12, 7, 8, 3, 17, 13, 19, 16, 20, 14, 9, 1, 5, 18, 15] (4463.48km)

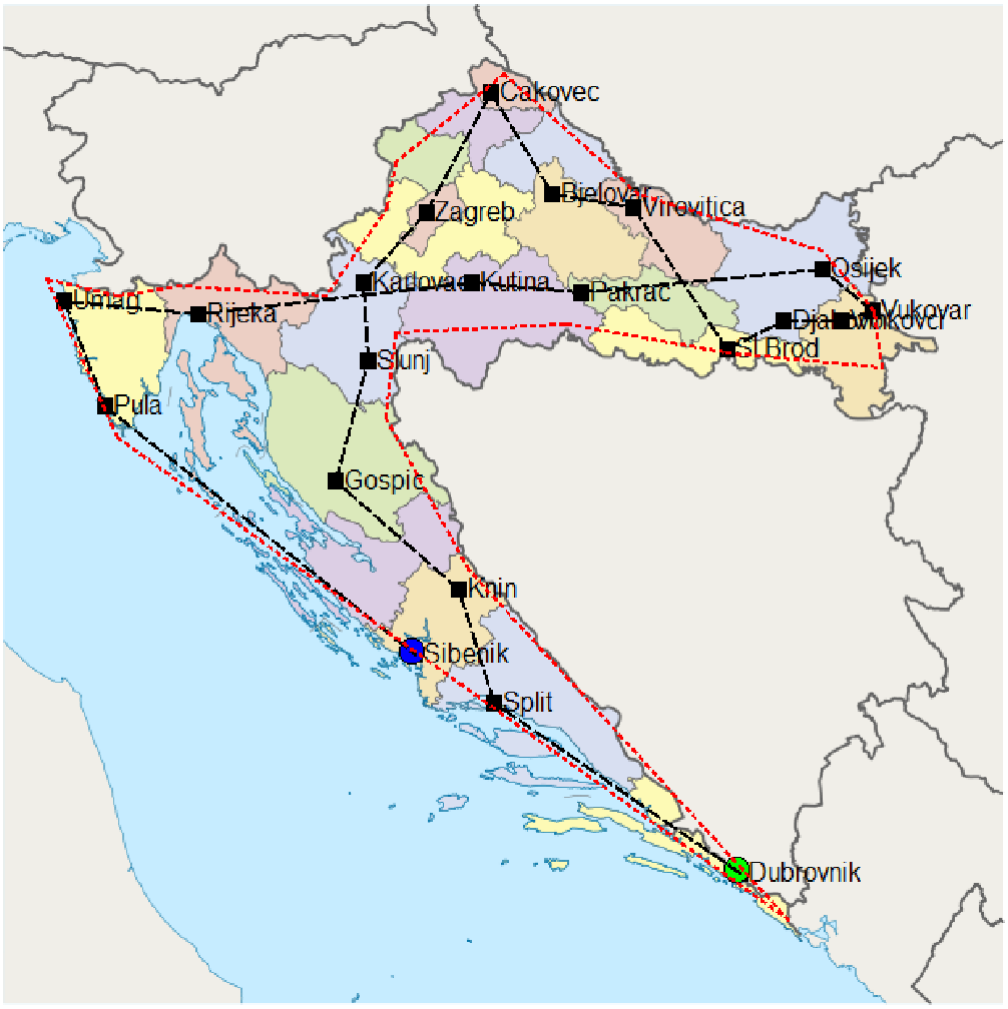
OVISNOST O BROJU ELITNIH ČLANOVA

Broj generacija	5000			
Populacija	100			
Postotak mutacije[%]	5%			
Broj elitnih članova	5	10	15	20
Rješenje	4524.51, 7582.26, 4775.52, 5942.86, 4588.70	5099.55, 4431.24, 4687.44, 4774.61, 4688.55	5217.47, 5121.02, 4514.54, 4881.69, 4595.23	4565.73, 5329.38, 4915.40, 4801.39, 4431.24
Najkraći put	4524.51	4431.24	4514.54	4801.39
Prosječan najkraći put (medijan)	4775.52	4688.55	4881.69	4431.24

Tablica 6. Ovisnost o broju elitnih članova



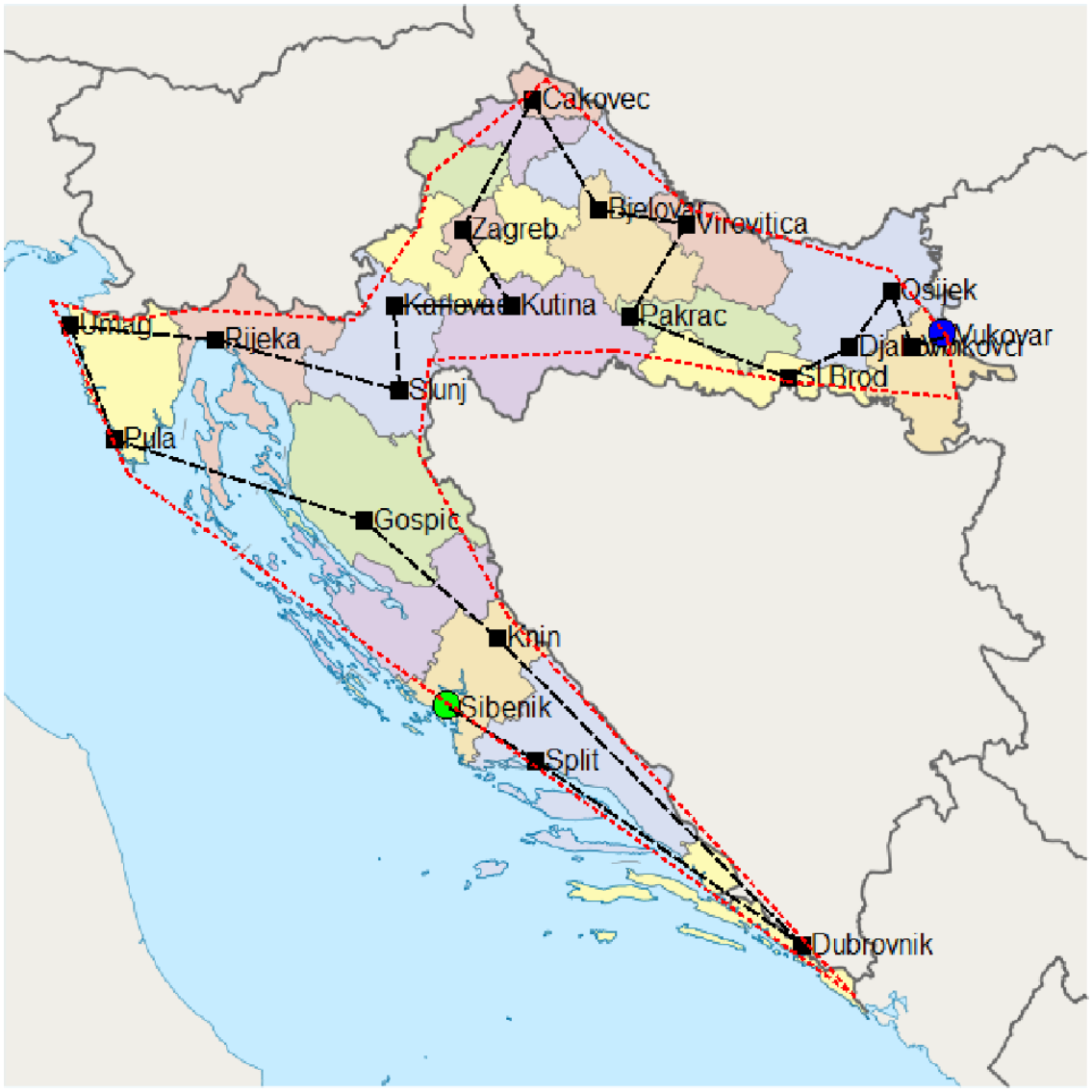
Slika 29. Prikaz ovisnosti dobrote i generacija o broju elitnih članova



Slika 30. Najbolje rješenje za broj elitnih članova 5

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

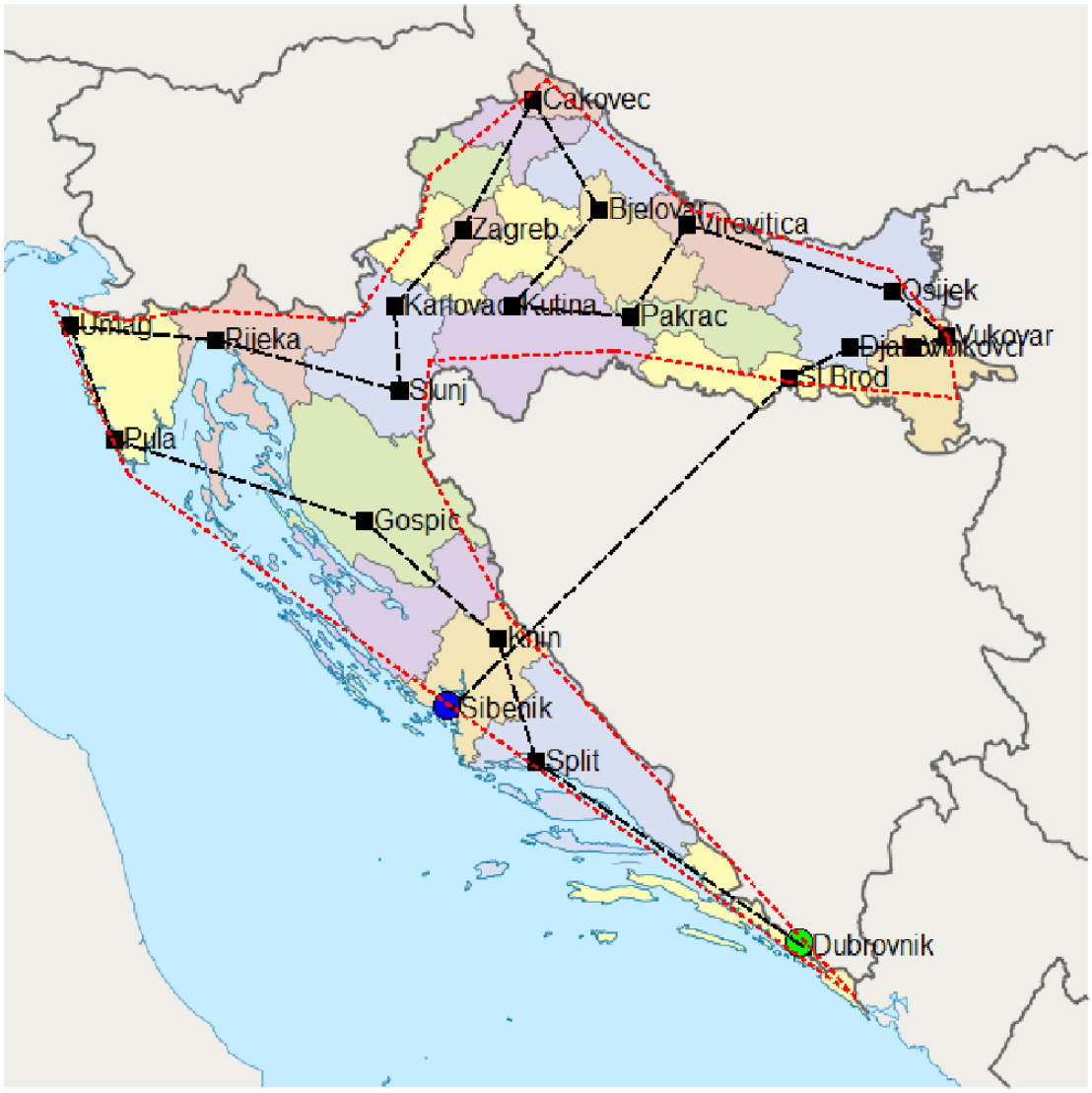
(4524.51km)



Slika 31. Najbolje rješenje za broj elitnih članova 10

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

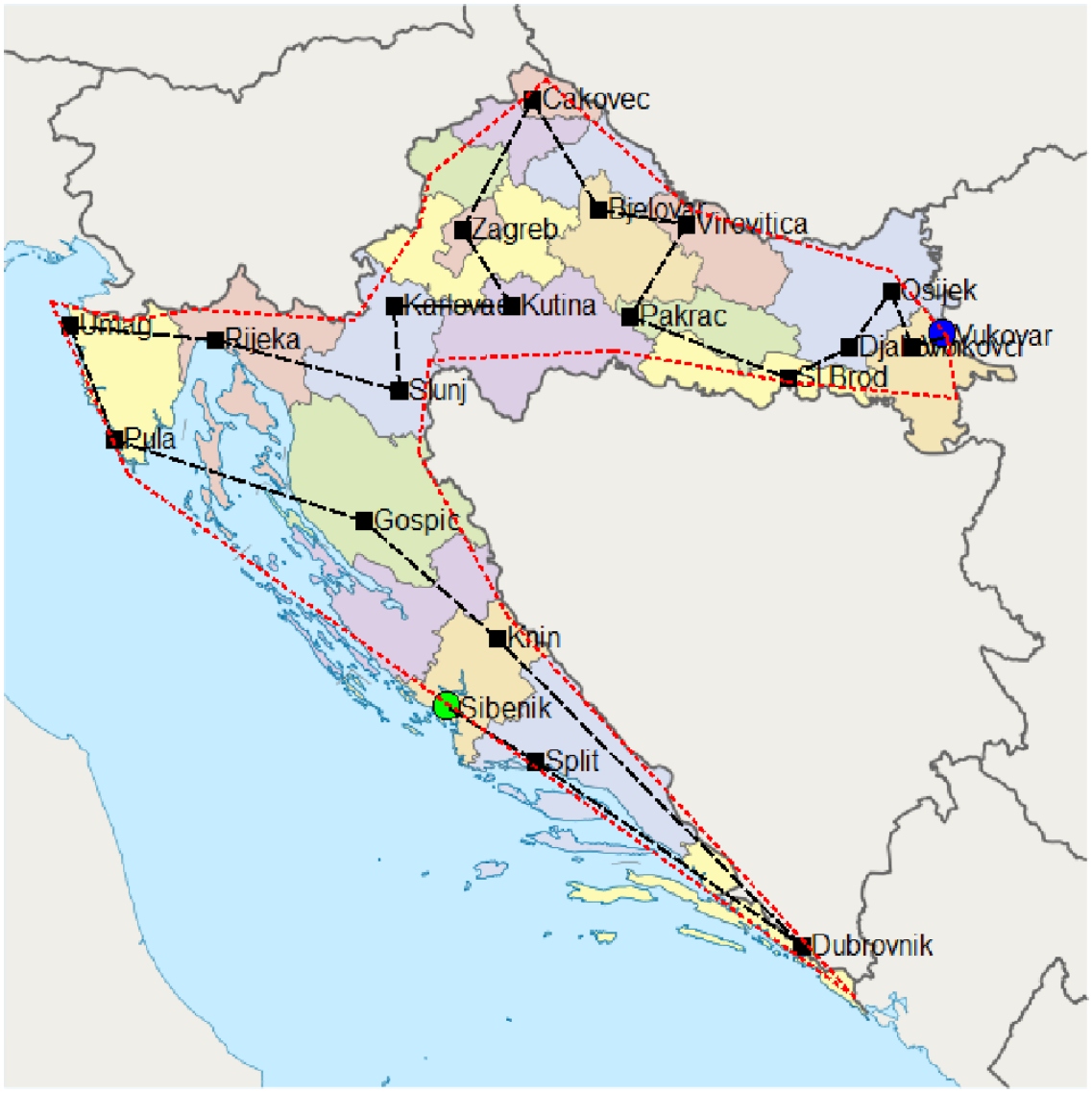
(4431.24kms)



Slika 32. Najbolje rješenje za broj elitnih članova 15

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

(4514.54km)



Slika 33. Najbolje rješenje za broj elitnih članova 20

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

(4431.24km)

ZAKLJUČAK

Iz rezultata možemo zaključiti da je populacija utječe na brzinu pronalaska rješenja dok mutacija utječe na kvalitetu rješenja. Povećavanjem populacije povećava se i šansa da neka od jedinki sadrži rješenje.

Ukoliko bi se mutacija previše povećala, one dobre jedinke bi se izmijenile, a to nije pozitivna stvar.

Isto tako je bitno broj elitnih članova postaviti dobro. Premali broj elitnih članova dovodi do gubitka dobrih jedinki dok prevelik broj elitnih članova omogućava preživljavanje onih loših jedinki koje zatim utječu na točnost i brzinu pronalaska rješenja. U ovom slučaju broj elitnih članova je bio optimalan oko 10 i 15.

Svaka izmjena parametara je u algoritmu odrađena pet puta no i dalje je teško vidjeti kako koji parametar utječe na brzinu pronalaska rješenja. Za detaljniji uvid potrebno je provesti pokus više od pet puta.