SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6492

Usmjeravanje vozila uz pomoć evolucijskog algoritma

Laura Majer

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Nevjerojatni napredci u sklopovlju omogućili su računarstvu velike uspjehe. Bez obzira na to, neki se problemi zbog svoje složenosti još uvijek ne mogu riješiti tzv. "grubom silom". Za rješavanje takvih problema potrebna je kreativnost i apstrakcija.

Nalazeći inspiraciju u prirodnom svijetu moderni su algoritmi uspjeli nadmašiti konvencionalne metode i podići znanstvenu interdisciplinarnost na novu razinu. Od kretanje roja ili čopora i evolucije vrsta sve do padanja kapljice vode i hlađenja metala, fascinantno je koliko je pravilnosti u entropijskim sustavima. Posebno je zanimljiva evolucija, proces koji objedinjuje sva živa bića i omogućuje im opstanak i kontinuiranu prilagodbu. Upravo cijela teorija evolucije - od nasljeđivanja genoma, preko mutacije i rekombinacije pa sve do selekcije najpogodnijih jedinki - služi kao temelj evolucijskog računarstva, čiji su predstavnici genetski algoritmi, evolucijsko programiranje i evolucijske strategije.

Zbog broja mogućih kombinacija koji faktorijelno raste povećanjem dimenzije problema (primjerice većeg broja gradova koje je potrebno obići u poznatom problemu trgovačkog putnika), NP-teški problemi nisu rješivi deterministčkim metodama. Genetski algoritmi jedan su od načina usmjerenog slučajnog pretraživanja prostora rješenja s drastično manjom složenosti nasuprot pretraživanja svih mogućnosti. U ovom će se radu analizirati primjena genetskih algoritama u rješavanju jednog od NP-teških problema - važnom problemu usmjeravanja vozila.

U nastavku rada detaljnije će se opisati problem usmjeravanja vozila te genetski algoritmi. Nakon toga, opisat će se implementirani algoritam te će se raspraviti o dobivenim rezultatima na nekoliko konkretnih primjera problema usmjeravanja vozila. [**?**]

2. Problem usmjeravanja vozila (VRP)

2.1. Motivacija

Problem usmjeravanja vozila (eng. *Vehicle Routing Problem*, nadalje VRP) prvi je puta formaliziran 1959. godine u članku Danziga i Ramsera pod naslovom *The Truck Dispatching Problem*. [3] Radi se o problemu kojemu je cilj naći optimalan raspored i redoslijed obilaženja lokacija za određeni broj dostavnih vozila, uz moguće dodatne parametre i ograničenja.

Par desetljeća kasnije cestovna je dostava još uvijek najjednostavniji i najjeftiniji način isporuke robe i međunarodnog transporta te se to neće promijeniti u bliskoj budućnosti. Stoga je važnost VRP-a samo porasla od njegovih početaka. Osim toga, bitno je napomenuti da se način rješavanja problema usmjeravanja vozila može primijeniti i na ostale probleme koji se mogu preslikati na pretraživanje prostora stanja i permutaciju ruta. Primjeri izravne primjene osim dostavljačkih i transportnih službi uključuju i logistiku većih korporacija, organizaciju rada, usmjeravanje ruta kod mrežnih usmjeritelja i slično. Iako je intuitivno, bitno je napomenuti da se problem ne mora usko definirati samo za dostavljačke službe nego logistiku većih korporacija, organizaciju rada kod kojeg zaposlenici obilaze klijente i slično.

2.2. Definicija problema

Problem usmjeravanja vozila formalno je nadogradnja problema trgovačkog putnika (eng. *Travelling Salesman Problem*, TSP). Gdje se u TSP-u optimizira kretanje jednog putnika ili vozila, u VRP-u se istovremeno koordinira i optimizira putovanje više njih. Temelj formalne definicije VRP je graf $G(V, E)$, gdje V označava vrhove grafa, a E bridove. Formalni parametri osnovnog oblika VRP-a su:

- $V = \{v_0, v_1, \ldots, v_n\}$, gdje v_0 označava polaznu točku (eng. *depot*)
- $V' = V \backslash \{v_0\}$ označava set od n čvorova, tj. lokacija kupaca

- C, matrica ne-negativnih udaljenosti c_{ij} između čvorova
- D, vektor zahtjeva lokacija (može se raditi o količini dobara koja se dostavljaju, vremenskom prozoru unutar kojeg se dostava mora izvršiti i ostalo)
- R_i, ruta vozila i.
- m, broj vozila koja imaju identične karakteristike

Ispravno rješenje rješenje sastoji se od $R_1, \ldots, R_m$, podskupa od V, gdje je redoslijed obilaska permutacija. Cijena rute R_i dana je kao : $F(R_i) = \sum_{i=0}^{m} c_{i,i+1} + \sum_{i=1}^{m} \delta_i$, gdje $c_{i,i+1}$ označava udaljenost između dvije susjedne lokacije u ruti, a δ_i označava količinu dobara koje lokacija zahtijeva, vrijeme potrebno za istovar dobara ili neki drugi parametar koji je definiran ovisno o varijanti VRP-a.

Skup ruta $R_1, \ldots, R_m$ ispravan je ako je svaka lokacija posjećena točno jednom i zadovoljeni su dodatni zahtjevi D koji specificiraju gornje granice kapaciteta, udaljenosti ili vremenskog trajanja.

Konačno, ukupna cijena rješenja dana je kao $F_{VRP} = \sum_{i=1}^{m} F(R_i)$.

2.3. Složenost

Gore navedeni problem je NP-težak jer je faktorijelne složenosti. To je činjenica koja ograničava metode koje se efektivno mogu koristiti za rješavanje problema.
Evaluacija rješenja ovisi o vrsti VRP, tj. o dodatnim zahtjevima D. Intuitivno mjerilo kvalitete rješenja je zbroj pređenih udaljenosti svih vozila, iako je u sklopu nekih varijanti VRP-a potrebno uzeti u obzir i kašnjenje u dostavi ili ostala kršenja propisanih ograničenja. Neke od metoda evaluacije uključuju i primjenu računalnog vida, pri čemu se analizira preklapanje u rutama.

2.4. Varijante

Ovisno o dodatnim zahtjevima, osnovni problem usmjeravanja vozila može se proširiti na varijante od kojih su neke:

Capacitated Vehicle Routing Problem (CVRP), gdje svako vozilo ima kapacitet C koji može prenijeti, a svaka lokacija ima zahtjev za količinom dobara Q. Ukupna suma količina na svim lokacijama koje vozilo posjeti ne smije nadmašiti Q.

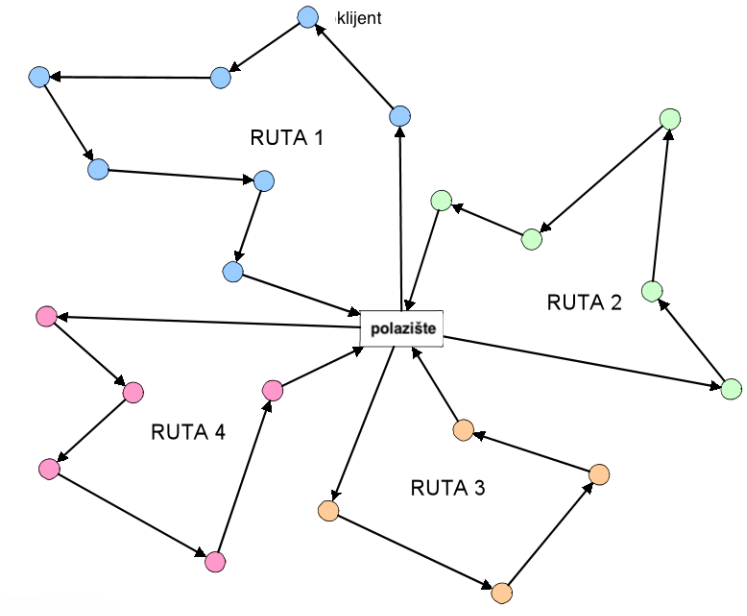
Vehicle Routing with Time Window, gdje se svaka dostava mora odviti unutar definiranog vremenskog okvira. Za dostavu izvan definiranog okvira određuje se dodatna cijena rješenja (koja rješenje čini manje kvalitetnim, s obzirom na to da je minimizacija cijene cilj VRP-a).

Multiple Depot Vehicle Routing Problem, gdje vozila mogu imati različito polazišno mjesto (eng. *depot*). Ova se varijanta pojavljuje kod većih sustava i moguće ju je razriješiti dijeljenjem na više manjih problema ovisno o polazišnoj točki, što bi rezultiralo drugačijim i vjerojatno suboptimalnim rješenjima.

Green Vehicle Routing Problem, gdje su korištena vozila na hibridni ili električni pogon i pritom rješavanja se u obzir moraju uzeti lokacije za punjenje baterija vozila i slična ograničenja. Cilj je rješenja minimizirati rute dostavnih vozila te emisiju C02 plinova u atmosferu.

U praksi je rijetko da se problem u potpunosti može opisati jednom od navedenih varijanti, već se u obzir mora uzeti više kriterija.

S obzirom na to da je višekriterijska selekcija rješenja daleko složenija od jednokriterijske, u ovom će radu biti razmatrana najšire primijenjiva varijanta, CVRP. Bitno je napomenuti da se kapacitet vozila i zahtjev lokacije može preslikati i na sljedeći način: kapacitet vozila C može predstavljati radno vrijeme zaposlenika, dok zahtjev lokacije Q može predstavljati vrijeme potrebno za obavljanje posla na toj lokaciji.



Slika 2.1: Jednostavan grafički prikaz rješenja VRP-a

2.5. Metode rješavanja

Potencijalni se načini rješavanja zbog složenosti VRP-a drastično razlikuju u primjenjivosti i uspješnosti. U nastavku su ukratko prikazani različiti pristupi problemu.

Egzaktne metode

Korištenje egzaktnih metoda kod rješavanja optimizacijskih problema daje garanciju da je rješenje zaista optimalno, no zbog kombinatorne eksplozije do koje dolazi uslijed povećanja dimenzija problema egzaktne metode poput *branch and bound* ili dinamičkog programiranja nisu pretjerano prikladne za rješavanje VRP-a. Način na koji se izbjegava faktorijelna složenost je korištenje nedeterminističkih metoda - heuristika i metaheuristika.

Heuristike

Cilj heuristika pronalazak je dovoljno dobrih rješenja optimizacijskih problema u dovoljno kratkom vremenu. Iako često ne nude garanciju optimalnosti, izvode se u polinomijalnoj složenosti.

Metaheuristike

Metaheuristike daljnje su poboljšanje u rješavanju optimizacijskih problema. Radi se o tematsko neovisnim i stoga široko primjenjivim metodama koje vode problemski specifične heuristike. Svoju strategiju metaheuristike mogu pripisati i raznim prirodnim pojavama i procesima, poput hlađenja tvari, kretanja roja ili evolucije. U nastavku su opisane neke od metaheuristika koje se koriste u rješavanju VRP-a.

Tabu pretraga jedna je od metaheuristika koje koriste lokalnu pretragu. Posebnost tabu pretrage je popis lokacija ili parametara za koje se zabranjuje ponovno obilaženje i korištenje. Osim toga, uvedeno je moguće obilaženje gorih rješenja, što sprječava zapinjanje u lokalnim optimumima.

Simulirano kaljenje jedna je od najstarijih metaheuristika. Svoju strategiju preuzima iz termodinamike te se kretanje prema rješenju odvija po modelima hlađenja materije.

Konačno, genetski algoritmi funkcioniraju na principu evolucije i genetike živih organizama. Upravo su oni odabrani za rješavanje problema u ovom završnom radu te su detaljnije opisani u nastavku.

3. Genetski algoritam

3.1. Definicija

Evolucija (lat. evolutio: razvoj, razvitak) je naziv za proces uslijed kojeg dolazi do promjena nasljednih osobina bioloških populacija tijekom velikog broja uzastopnih generacija. Teorija evolucije uspostavljena je sredinom 19. stoljeća, a kao inspiracija u računarstvu koristi se zadnjih pedesetak godina. Razlog tome je to što se proces evolucije može interpretirati i kao proces pretraživanja prostora stanja.

Genetski algoritam dio je genetskog računarstva. Spada u heurističke metode optimiranja i svojim karakteristikama imitira prirodni evolucijski proces. Jedinka u biološkoj populaciji predstavlja jedno moguće rješenje problema, dok je usmjeravanje prema kvalitetnijim rješenjima ostvareno pomoću procjene kvalitete jedinki i određenih selekcijskih algoritama. U nastavku je prikazana analogija između prirodnog evolucijskog procesa i genetskog algoritma koji se temelji na njemu. [4]

evolucija	**genetski algoritam**
jedinka	jedno rješenje problema
populacija	skup rješenja
mutacija genoma jedinke	unarni operatori
reprodukcija jedinki	operatori višeg reda
šanse za preživljavanje	dobrota

Tablica 3.1: Analogija između procesa evolucije i genetskih algoritama

3.2. Komponente algoritma

Prilikom odabira genetskog algoritma kao metode rješavanja problema bitno je uočiti da određene implementacijske odluke uvelike utječu na konačnu kvalitetu programa. Ključne komponente genetskog algoritma su sljedeće:

1. genotip, tj. struktura podataka koja će predstavljati rješenje
2. inicijalizacija početne populacije
3. funkcija dobrote, koja služi za evaluaciju kvalitete rješenja
4. operatori pomoću kojih se stvaraju nove jedinke (operatori križanja i mutacije)
5. postupak selekcije jedinki koje prelaze u sljedeću generaciju
6. uvjet zaustavljanja koji određuje koliko dugo se rješenje pretražuje

3.2.1. Genotip

Kvaliteta rješenja genetskog algoritma uvelike ovisi o odabiru strukture podataka koja predstavlja genom jedinke. Taj se odabir prvenstveno temelji na zahtjevima zadatka. Radni okviri i programi za GA nude već implementirane genotipe, gdje je su za svaki definirani i operatori križanja i mutacije.
Uobičajene strukture uključuju binarni kod, realan broj (koji se interpretira kao broj s pomičnom točkom s jednostrukom ili dvostrukom preciznošću), permutaciju cijelih brojeva te stablo.

3.2.2. Inicijalizacija populacije

Smjer u kojem će krenuti genetski algoritam uvelike određuje početna populacija jedinki. Ako je za problem moguće generirati mnoga nemoguća rješenja, prisutnost takvih u početnoj populaciji može kompromitirati konačno rješenje.
Tri su uobičajena načina za stvaranje početne populacije : jedinke se mogu generirati u odnosu na neko otprije poznato rješenje, generiranje može biti u potpunosti slučajno (u okviru restrikcija odabranog genotipa) ili se može koristiti pseudo-slučajno generiranje koje u obzir uzima poželjne karakteristike jedinke i omogućuje početak s "boljim" jedinkama.
Osim toga, bitno je odrediti veličinu populacije, koja u većini algoritama ostaje konstanta do završetka izvođenja.

3.2.3. Funkcija dobrote

U biološkom smislu "bolja" jedinka je ona bolje prilagođena na okolinu ili s velikim brojem poželjnih karakteristika. U kontekstu GA, jedinka (rješenje) je dobra ako zadovoljava zahtjeve problema te je svojim svojstvima blizu optimumu.
U optimizacijskom problemu može se tražiti minimalna ili maksimalna legalna vrijednost, te je u skladu s tim jedinka "bolja" što je rezultat funkcije dobrote za tu jedinku

manji ili veći, ovisno o definiciji problema.

3.2.4. Operatori križanja i mutacije

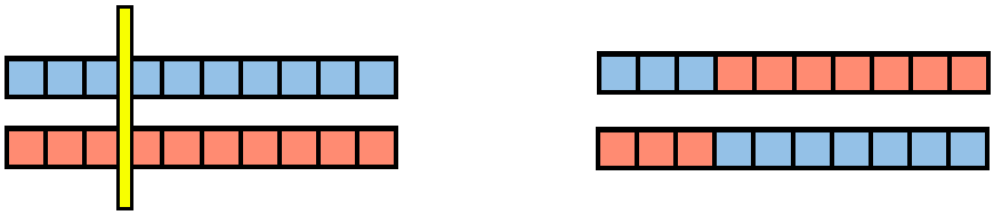
Potomci dobiveni spolnim razmnožavanjem uvijek su genetski različiti od roditeljskih jedinki te se i međusobno razlikuju. Razlog tome je rekombinacija gena.

U kontekstu algoritma, promjene u genotipu omogućuju dolazak do rješenja. Preko relativno malih, te relativno slučajnih legalnih promjena unutar odabranih jedinki, genetski materijal se iz generacije u generaciju mijenja te se sve više bliži optimumu (ako su korišteni parametri te operatori ispravni i prilagođeni problemu). Operatori se dijele na unarne, koji stvaraju novu jedinku mijenjajući manji dio genetskog materijala (operatori mutacije) te operatore višeg reda, koji kreiraju nove jedinke kombinirajući osobine dvaju jedinki (operatori križanja). Najčešće se unutar parametara algoritma određuje koja je vjerojatnost za mutaciju, a učestalost križanja i odabir jedinki koje ulaze u križanje ovisi o korištenom algoritmu selekcije.

Primjeri

Jedna od učestalih metoda križanja je uniformno križanje. Na početku postupka slučajnim se odabirom selektira točka presjeka unutar genotipa. Ako se primjerice radi o permutaciji, odabrat će se mjesto između dva indeksa permutacije. Nakon toga, dobiveni novi genotip nastat će od odlomka prvog roditelja ispred presjeka te drugog roditelja nakon presjeka te obrnuto za drugu jedinku koja se analogno generira. Prednost ove metode, osim njene jednostavnosti, činjenica je da se njenim korištenjem istovremeno mogu generiraju dva različita potomka. Primjer uniformnog križanja za permutaciju duljine 10 prikazan je na Slici 3.2.. Jedna je roditeljska jedinka prikazana plavom, dok je druga prikazana narančastom bojom. Nakon odabrane točke presjeka (koja je u ovom primjeru između indeksa 2 i 3 u permutaciji), kombinacijom materijala iz roditeljskih jedinki generiraju se dva potomka s različitim genomom[1]. Prilikom korištenja ovog operatora križanja, kao i za sve ostale operatore, treba obratiti pažnju na restrikcije genotipa te samog problema.

[1] Ako je genetski materijal prije ili nakon presjeka u obje roditeljske jedinke jednak, generiraju se dva jednaka potomka. Prilikom programskog ostvarenja uniformnog križanja ova se posljedica može spriječiti dodatnim provjerama.



Slika 3.1: Ilustrativni prikaz uniformnog križanja

Mutacija znatno ovisi o korištenom genotipu. Za pojednostavljenje primjera, u nastavku će se razmatrati binarni kod. Korištenje binarnog koda omogućuje promjenu u jednom bitu bez razmatranja ostalih bitova, što nije slučaj kod nekih učestalih genotipa. Mutacija se na binarnom kodu provodi u dva koraka - za početak se slučajnim odabirom selektira jedan bit koji će se promijeniti, te se zatim njegova vrijednost mijenja u suprotnu (ako je odabrani bit prije mutacije imao vrijednost 0, nakon mutacije vrijednost će biti 1 i obrnuto).

Na Slici 3.2. prikazan je primjer mutacije u binarnom kodu. Siva boja predstavlja vrijednost 0 (false), dok crvena predstavlja vrijednost 1 (true). Strelicom je prikazana slučajno odabrana točka mutacije. Desno od početne jedinke prikazana je mutirana jedinka s promijenjenim 8. bitom.

Slika 3.2: Ilustrativni prikaz mutacije u binarnom kodu

3.2.5. Selekcija

Prelaskom iz generacije u generaciju potrebno je odabrati jedinke koje preživljavaju, tj. odraditi postupak selekcije. Svrha selekcije čuvanje je te prenošenje dobrih svojstava na sljedeću generaciju jedinki. No, pohlepan odabir isključivo najboljih jedinki u generaciji mogao bi rezultirati zapinjanjem u lokalnom optimumu. U nastavku su opisani neki uobičajeni selekcijski algoritmi za GA.

Jednostavna selekcija (eng. *Roulette wheel selection*) metoda je u kojoj je vjerojatnost odabira jedinke proporcionalna s njenom dobrotom. Iako bolje jedinke imaju

veću šansu za preživljavanje, njihov opstanak nije siguran kao ni eliminacija slabijih jedinki. Rezultat toga je selekcija koja omogućuje određenu dozu raznolikosti u odabranoj novoj generaciji.

Eliminacijska selekcija djeluje suprotno od jednostavne selekcije. Na svakoj se generaciji obavlja selekcija, vjerojatnost odabira jedinke obrnuto je proporcionalna njenoj dobroti te se odabrane jedinke brišu iz populacije.

Turnirska selekcija metoda je korištena u ovoj implementaciji. Nakon odabira k jedinki (najčešće 3), najgora od njih mijenja se djetetom ostalih jedinki. Nova jedinka se zatim mutira te u tom obliku ubacuje u populaciju.

3.2.6. Uvjet zaustavljanja

Prilikom pokretanja genetskog algoritma bitno je odrediti kad će završiti s izvođenjem. Moguće je ograničiti broj evaluacija, koji definira koliko će se puta pokrenuti algoritam selekcije. Osim toga, može se zadati maksimalan broj uzastopnih generacija za koji se dopušta stagnacija najbolje vrijednosti funkcije dobrote, što omogućuje zaustavljanje algoritma koji je zapeo. Manje uobičajen uvjet zaustavljanja je vremensko ograničenje izvođenja.
Iako ograničenja uvelike ovise o okviru unutar kojeg se algoritam primjenjuje, minimum od 100 tisuća evaluacija smatra se prihvatljivim za procjenu kvalitete algoritma.

3.3. Tijek izvođenja algoritma

Na početku izvođenja algoritma potrebno je inicijalizirati početnu populaciju. Nakon toga, slijedi proces koji se ponavlja sve dok nije ispunjen jedan od postavljenih uvjeta zaustavljanja. Proces se sastoji od djelovanja genetskih operatora selekcije, križanja i mutacije nad populacijom jedinki. Tijekom selekcije loše jedinke odumiru dok bolje opstaju te se u sljedećem koraku križaju, što je ekvivalent razmnožavanju. Križanjem se prenose određena svojstva roditelja na djecu, a mutacijom se slučajnom promjenom gena mijenjaju svojstva jedinke. Ponavljanjem tog postupka iz generacije u generaciju postiže se sve veća prosječna dobrota. U nastavku je prikazan pseudokod opisanog tijeka algoritma.

```
Genetski algoritam {

      t = 0
      generiraj početnu populaciju potencijalnih rješenja P(0);
      sve dok (! zadovoljen uvjet završetka)
      {
            t = t + 1;
            selektiraj P'(t) iz P(t-1);
            križaj jedinke iz P'(t) i djecu spremi u P(t);
            mutiraj jedinke iz P(t);
      }
      ispiši rješenje;
}
```

Slika 3.3: Pseudokod genetskog algoritma

4. Praktični rad

4.1. ECF

Evolutionary Computational Framework, kraće ECF, radni je okvir za programski jezik C++ razvijen na Fakultetu elektrotehnike i računarstva u Zagrebu koji se koristi za genetsko računarstvo.

Da bi se ECF koristio za genetski algoritam, potrebno je unutar datoteke *xml* formata definirati parametre algoritma - prvenstveno genotip, selekcijski algoritam, te datoteku preko koje se obavlja učitavanje podataka, no i ostale karakteristike algoritma poput načina inicijalizacije populacije, operatora mutacije i križanja, te vjerojatnosti mutacije. Dodatne mogućnosti uključuju ispis najboljih rezultata pokretanja (eng. *Best of run*) te sveukupnu statistiku, koja uključuje minimalan, maksimalan te prosječan rezultat funkcije dobrote, kao i broj evaluacija i generacija.

Sve spomenute komponente mogu se samostalno programski ostvariti, no unutar radnog okvira dostupan je određeni broj gotovih komponenti. Osim toga, za demonstraciju svojstava ECF-a priloženo je par primjera koji uključuju i rješenje problema trgovačkog putnika (eng. *Travelling salesman problem*, TSP), čija je generalizacija problem usmjeravanja vozila koji se analizira u ovom radu. [5]

4.2. Dohvat podataka

Datoteka koja predstavlja specifičnu instancu VRP-a određenog je formata. Postoji nekoliko uobičajenih formata za opisivanje problema. U formatu korištenom za ovaj rad unutar datoteke naveden je broj lokacija, kapacitet vozila (jednak za svako vozilo) a za svaku lokaciju njene koordinate te količina dobara koju zahtijeva. Osim toga naveden je indeks polazišta. Sve korištene datoteke preuzete su s [1].

```
NAME : A-n32-k5
COMMENT : (Augerat et al, Min no of trucks: 5, Optimal value: 784)
TYPE : CVRP
DIMENSION : 32
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 100
NODE_COORD_SECTION
 1 82 76
 2 96 44
 3 50 5
 4 49 8
 5 13 7
 6 29 89
 7 58 30
 8 84 39
 9 14 24
 10 2 39
 11 3 82
 12 5 10
 13 98 52
 14 84 25
 15 61 59
 16 1 65
 17 88 51
 18 91 2
 19 19 32
 20 93 3
 21 50 93
 22 98 14
 23 5 42
 24 42 9
 25 61 62
 26 9 97
 27 80 55
 28 57 69
 29 23 15
 30 20 70
 31 85 60
 32 98 5
DEMAND_SECTION
1 0
2 19
3 21
4 6
5 19
6 7
7 12
8 16
9 6
10 16
11 8
12 14
13 21
14 16
15 3
16 22
17 18
18 19
19 1
20 24
21 8
22 12
23 4
24 8
25 24
26 24
27 2
28 20
29 15
30 2
31 14
32 9
DEPOT_SECTION
 1
 -1
EOF
```

Slika 4.1: Prikaz formata ulazne datoteke [1]

4.3. Programski ostvarene komponente

4.3.1. Genotip

U ovoj je implementaciji kao genotip korištena permutacija, no u dva različita oblika (gdje oblik u ovom kontekstu predstavlja veličinu permutacije, operatore križanja i mutacije te evaluaciju kvalitete jedinke).

Bitno je napomenuti da se unutar cijele implementacije, uključujući i dva oblika permutacije, kao startna lokacija uzima ona na indeksu 0 te taj indeks nije prisutan u permutaciji. Razlog tome je taj da je u definiciji problema usmjeravanja vozila određeno da ruta počinje i završava na polazišnoj lokaciji.

Prva implementirana permutacija (u nastavku *duga* permutacija) sadrži indekse lokacija u rasponu [1,$n-1$] te indekse vozila u rasponu [n,m] gdje n predstavlja broj lokacija, a m broj dostupnih dostavnih vozila. Duljina duge permutacije je slijedno $n+m-1$. Ruta nekog vozila definirana je indeksima lokacija koji su navedeni nakon indeksa vozila. Redoslijed obilaženja lokacija jednak je redoslijedu indeksa u permutaciji. U ovoj su permutaciji rute vozila jedinstveno definirane, što olakšava postupak evalucije jedinke, no jedan je od razloga zašto se stvaraju mnoga nemoguća rješenja. Daljnji razlozi bit će obrazloženi u analizi rezultata.
Inicijalizacija početne populacije u obzir uzima ispravan oblik ove permutacije, no ne i heuristiku. Prillikom generiranja jedinki izračunava se prosječan broj lokacija koje svako vozilo mora posjetiti, te se indeksi vozila postavljalju s tolikim razmakom. Iako je takva metoda stvaranja smanjila vjerojatnost da generirana jedinka predstavlja nemoguće rješenje, ta mogućnost nije se u potpunosti eliminirala. Bez korištenja podataka o udaljenostima i potražnjama specifične lokacije, nemoguće je odrediti hoće li ruta biti ispravna ili ne.

Na Slici 4.2. prikazan je primjer *duge* permutacije s parametrima n = 8, m = 3. S obzirom na parametre, lokacije će imati indekse od 1 do 7, dok će vozila biti označena indeksima 8,9 i 10. Ruta za pojedino vozilo definirana je indeksima koji se nalaze između indeksa dotičnog vozila i sljedećeg indeksa koji predstavlja vozilo. Primjerice, ruta koju će po ovoj permutaciji obići vozilo označeno indeksom 8 je početna lokacija - lokacija 1 - lokacija 3 - početna lokacija[1]. Na jednak se način određuju rute i za ostala dva vozila u ovom primjeru. Na slici 4.3. prikazana je ista permutacija s naznačenim

[1]U definiciji problema usmjeravanja vozila konačno odredište vozila jednako je polazišnoj lokaciji.

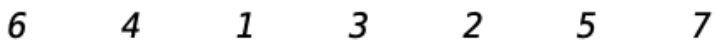
pripadajućim rutama vozila, gdje je indeks vozila zaokružen te su lokacije koje to vozilo obilazi podcrtane jednakom bojom.

Slika 4.2: Primjer duge permutacije (n=8, m=3)

Slika 4.3: Primjer duge permutacije s naznačenim rutama

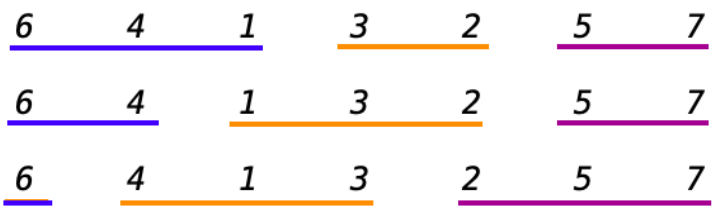
Druga implementirana permutacija (u nastavku *kratka* permutacija) sadrži isključivo indekse lokacija u rasponu $[1, n-1]$. Za razliku od *duge* permutacije, ovdje rute za pojedino vozilo nisu eksplicitno definirane, što znači da se ista permutacija može interpretirati kao nekoliko različitih ruta. Određivanje ruta izvodi se unutar funkcije dobrote i temelji se na kapacitetu pojedinog vozila.

Na Slici 4.4. prikazan je primjer *kratke* permutacije s parametrom n=8. Iz same permutacije nije moguće odrediti koliko vozila je namijenjeno za obilazak lokacija jer indeksi vozila ni na koji način nisu vidljivi. Ruta se za svako vozilo određuje ovisno o potražnji lokacije te preostalom kapacitetu vozila. Broj vozila stoga ovisi isključivo o zahtjevima lokacija te se u parametrima programa zadaje maksimum vozila, ne egzaktan broj, iako su problemi najčešće takvi da zahtjev odgovara maksimumu. U ovom je primjeru broj vozila postavljen na m=3.



Slika 4.4: Primjer kratke permutacije (n=8)

Permutacija na Slici 4.4. mogla bi definirati bilo koje od rješenja prikazanih na Slici 4.5. Unutar rješenja, pojedina je ruta označena crtom jednake boje ispod slijednih lokacija. Primjerice, ako bi raspored lokacija i vozila bio u skladu s prvim retkom Slike 4.5. prvo bi vozilo obišlo tri lokacije, dok bi drugo, kao i treće vozilo, obišlo dvije lokacije. Iako se to ne čini efikasno, rješenje prikazano u trećem redu Slike, gdje prvo vozilo obilazi samo jednu lokaciju (označenu indeksom 6), također prikazuje moguće rješenje koje indicira da je zahtjev lokacije 6 veći od zahtjeva ostalih lokacija.



Slika 4.5: Primjeri mogućih ruta kratke permutacije (n=8)

Prilikom inicijalizacije populacije jedinke se generiraju u potpunosti nasumično.

4.3.2. Funkcija dobrote

Implementirane su dvije funkcije dobrote, jedna za svaku od permutacija. Cilj funkcije dobrote je što točnije evaluirati jedinku.

Rješenja se u ovom problemu procjenjuju s obzirom na dva kriterija:

1. **rute moraju biti ostvarive**, tj. zbroj potražnja lokacija koje vozilo obilazi mora biti manji ili jednak njegovu kapacitetu
2. **ukupan zbroj duljina** svih ruta mora biti **čim manji**

Prvi je uvjet obavezan, što znači da sve jedinke koje predstavljaju nevažeća rješenja u funkciji dobrote na neki način trebamo diskreditirati. S obzirom na to da će se unutar selekcije birati jedinke s minimalnom vrijednosti funkcije dobrote, nevažećim permutacijama dodjeljuje se maksimalna vrijednost koju je u programskom jeziku C++ moguće prikazati cijelim brojem, INT_MAX (što iznosi 2.14748e+09). Na taj se način označava da rute dobivene iz permutacije nisu prikladne za rješavanje problema. Ako je dakle na kraju izvođenja algoritma kao najbolja vrijednost populacije procijenjena upravo INT_MAX, to je indikator da se nakon unaprijed određenog broja generacija nije našla ni jedna jedinka s izvedivim rješenjem.

***Duga* permutacija** jednoznačno definira rute što olakšava proces evaluacije. Unutar funkcije dobrote permutacija se najprije rotira tako da je indeks nekog vozila na prvom mjestu (što olakšava daljnje računanje, a ne narušava rezultat) te se nakon toga slijedno zbrajaju udaljenosti svih lokacija smještenih između indeksa dva vozila. Ako se kapacitet vozila premaši prije kraja računanja njegove rute, računanje se prekida i kao vrijednost se vraća INT_MAX.

Određivanje ruta za ***kratku*** **permutaciju** nije toliko jednostavno. Postoji više različitih načina na koji se rute mogu odrediti, gdje neki uključuju računanje minimalne udaljenosti (što bi znatno povećalo vremensku složenost cjelokupnog algoritma). U ovoj je implementaciji odabrana varijanta s manjom vremenskom složenosti koja se oslanja na kapacitete vozila za izračunavanje mogućih ruta. Tom se metodom kao ruta vozila uzima maksimalan broj lokacija koje može obići s obzirom na preostali kapacitet i potražnju specifične lokacije. Kroz permutaciju se prolazi slijedno, te se u slučaju prekoračenja kapaciteta vozila započinje nova ruta, tj. računa se ruta za sljedeće vozilo. U nastavku je za dodatno pojašnjenje prikazan pseudokod rješenja.

```
indeks = 0;
preostali_kapacitet = MAX_KAPACITET;
broj_preostalih_vozila = MAX_BROJ_VOZILA-1;
sve dok(indeks <= zadnji indeks) {

      ako (potražnja[indeks]<preostali_kapacitet) {
            dodaj udaljenost;
            preostali_kapacitet -= potražnja[indeks];
            ++indeks;
      } inače  {
            završi rutu;
            --broj_preostalih_vozila;
            ako (broj_preostalih_vozila == 0) {
                  vrati MAX_INT;
            }
            započni novu rutu;
            preostali_kapacitet = MAX_KAPACITET;
      }
}

vrati udaljenost;
```

Slika 4.6: Pseudokod funkcije dobrote za kratku permutaciju

Kao dodatno poboljšanje funkcije dobrote za kratku permutaciju uveden je parametar $double\ check$ koji se zadaje pri inicijalizaciji programa. Ako je vrijednost parametra različita od nule, provjera se ne obavlja samo "unaprijed" (od početka do kraja permutacije) nego i "unatrag" (od kraja do početka permutacije). Nakon toga se vrijednosti te dvije provjere uspoređuju i kao rezultat funkcije vraća se manja vrijednost. Ovakav postupak ne samo da smanjuje vjerojatnost nemogućih rješenja, nego omogućava filtriranje boljih rezultata.

4.4. Korištene komponente ECF-a

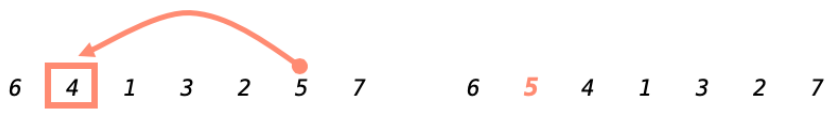
4.4.1. Selekcija

Algoritam selekcije korišten u ovoj implementaciji GA je turnirska selekcija veličine 3, što znači da se u jednom krugu selekcije iz populacije nasumično odabiru tri jedinke. Među odabranim jedinkama, ona s najgorom dobrotom zamijenjuje se s produktom križanja ostale dvije jedinke. Nova jedinka se zatim mutira (s vjerojatnošću mutiranja zadana u parametrima programa) te se sprema u populaciju.
Turnirska selekcija jedna je od selekcija implementiranih u ECF-u koje su prikladne za sve , pa za njeno korištenje nije bilo potrebno ništa dodatno implementirati.

4.4.2. Operatori križanja i mutacije

Za sve genotipe unaprijed dostupne unutar ECF-a dostupno je i nekoliko operatora križanja i mutacije. S obzirom na to da je jedan od raspoloživih genotipova permutacija, što je upravo format oba implementirana genotipa, kao operatori korišteni su operatori mutacije i križanja za permutaciju iz ECF-a.
Odabrano je 8 operatora križanja te 3 operatora mutacije, te je vjerojatnost njihovog korištenja postavljena tako da je jednako vjerojatno koji će se od operatora odabrati. Ovaj pristup pokazao se efektivnijim od korištenja samo jednog operatora. [6]
Korišteni operatori mutacije su:

1. **operator umetanja**, gdje se nasumično odabiru dva indeksa te se element na prvom indeksu umeće na drugi indeks. Ostali elementi pomiču se jedno mjesto u desno. Ova je mutacija prikazana na Slici 4.7.. Nakon nasumično odabranih indeksa, gdje se u ovom slučaju prvo odabire 5 i nakon toga 1, element se pomiče s prvog indeksa na drugi dok se elementi pomiču za jedno mjesto u desno.



Slika 4.7: Ilustrativni prikaz mutacije umetanjem

2. **operator inverzije**, gdje se isječak permutacije između dva nasumično odabrana indeksa invertira. Na Slici 4.8. prikazan je operator inverzije između indeksa 0 i 3. Provođenje ove mutacije na kratkoj permutaciji rezultirati će u promjeni

redoslijeda rute, ako odabrani indeksi označuju početak i kraj rute ili pak u potpunoj promjeni rasporeda lokacija između vozila. S obzirom na to da mutacija potencijalno utječe na nekoliko elemenata, njeno djelovanje može drastično izmijeniti jedinku, što može imati i pozitivan i negativan učinak.

Slika 4.8: Ilustrativni prikaz mutacije umetanjem

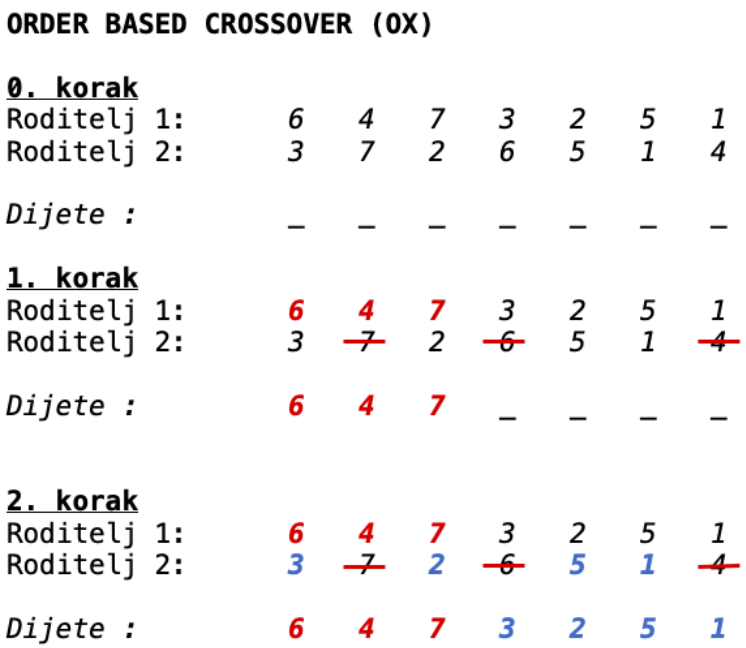
3. **operator zamjene**, gdje elementi dva nasumično odabrana indeksa mijenjaju mjesta. Efekt ove mutacije prikazan je na Slici 4.9., gdje su odabrani indeksi 2 i 6.

Slika 4.9: Ilustrativni prikaz mutacije umetanjem

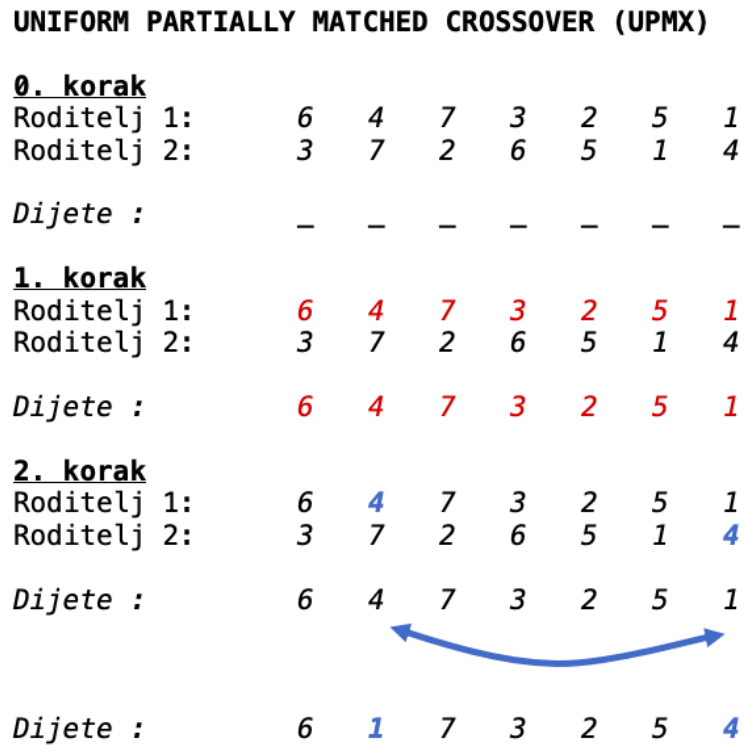
U programskom ostvarenju korišteno je osam dostupnih operatora križanja, no u ovom poglavlju detaljnije će biti opisana dva operatora. Za daljnja pojašnjenja vidjeti dokumentaciju korištenog radnog okvira (dostupna na [5]).

Korišteni operatori križanja su:

1. *order based crossover (OX)*, jednostavan i brz operator križanja prikazan na Slici 4.10.. Dijete se iz dvije roditeljske jedinke generira tako da se iz prvog roditelja odaberu dva indeksa te se sekvenca iz permutacije između ta dva indeksa kopira u dijete na isto mjesto. Zatim se od početka do kraja drugog roditelja kopiraju vrijednosti koje se još ne nalaze u dijetetu (zanemaruju se prethodno kopirane vrijednosti iz prvog roditelja).
2. *uniform partially matched crossover (UPMX)*, koji je prikazan na Slici 4.11.. Prvi korak u izvođenju operatora je inicijalizacija djeteta na vrijednosti prvog roditelja. Zatim se $n/3$ (gdje n označava duljinu permutacije) puta bira slučajni indeks iz prvog roditelja te se u drogm roditelju traži element koji je jednak elementu koji se nalazi na odabranom indeksu u prvom roditelju. Ako indeksi nisu jednaki u djetetu će elementi koji se nalaze na ta dva indeksa promijeniti mjesta. Osim toga, tijekom izvođenja pamte se već odabrani indeksi.



Slika 4.10: Ilustrativni prikaz order based križanja (OX)



Slika 4.11: Ilustrativni prikaz uniform partially matched križanja (UPMX)

S obzirom na to da se operatori križanja i mutacije nisu ni na koji način prilagođavali zadatku, dva oblika genotipa imati će poprilično različite rezultate i uspješnost. *Duga* permutacija, u kojoj su indeksi vozila izmiješani s indeksima lokacija, osjetljivija je na svaku promjenu rasporeda indeksa vozila te je korištenje neprilagođenih operatora križanja i mutacije riskantan potez. *Kratka* permutacija je, s obzirom na svoju strukturu i činjenicu da se od ECF-ovog modela permutacije razlikuje u samo jednoj stvari (ECF-ova permutacija sadrži nulu, dok ju kratka permutacija ne sadrži), daleko bolje prilagođena već ugrađenim općenitim operatorima križanja i mutacije.

5. Rezultati

5.1. Korišteni parametri

Za evaluaciju ukupne kvalitete implementacije i usporedbu dvije vrste genotipa korištena su tri konkretna problema. Dimenzije problema znatno se razlikuju, što omogućuje usporedbu na različitoj razini složenosti. Odabrani su problemi s 33, 48 te 65 lokacija.
Da bi uspoređivanje bilo konzistentno za svaki su se problem te odabrani genotip koristili jednaki parametri:

- **veličina populacije:** 50
- **broj generacija:** 3000
- **vjerojatnost mutacije:** 0.3
- **broj pokretanja:** 100

Pri pokretanju nije korišten parametar koji zaustavlja izvođenje nakon dugotrajne stagnacije najbolje vrijednosti. Nakon izvedenih 100 pokretanja programa, statistička analiza odradila se pomoću skripte napisane u programskom jeziku Perl.

5.2. Analiza rezultata

Osim općenite analize rezultata u nastavku će se uspoređivati rezultati dva implementirana genotipa - duge i kratke permutacije, te dva načina generiranja ruta i evaluiranja jedinki za kratku permutaciju - jednostruke i dvostruke provjere. Prilikom uspoređivanja kvalitete genotipa korišteni su rezultati jednostruke provjere.

5.2.1. Usporedba genotipova

Na problemu najmanje dimenzije (n=33,m=5) razlika između kvalitete dva genotipa nije toliko očita. Iako duga permutacija nakon 100 pokretanja programa nije

pronašla optimum, odstupanje od optimuma iznosi samo 7 kilometara ili 1%. No, s povećanjem dimenzije, a time i kompleksnosti problema, mane duge permutacije sve su vidljivije u rezultatima.

Za problem koji obuhvaća 48 lokacija odstupanje od optimuma u slučaju duge permutacije bilo je 8%, dok je za kratku permutaciju iznosio manje od 4%. Osim toga, od 100 pokretanja programa duga permutacija pronašla je valjano rješenje u samo 21 pokretanju, dok je kratka permutacija našla valjano rješenje u svakom pokretanju.

Ipak, tek je na primjeru s 65 lokacija slaba upotrebljivost duge permutacije postala očita. Nakon 100 pokretanja nije nađeno ni jedno valjano rješenje. Kratka permutacija je, s druge strane, pronašla valjano rješenje u svakom pokretanju te je najbolji rezultat odstupao od optimuma za otprilike 8%.

	33 lokacije	48 lokacija	65 lokacija
optimum	**661**	**1073**	**1174**
duga permutacija	668	1166	-
kratka permutacija (jednostruko)	661	1112	1270

Tablica 5.1: Usporedba dva genotipa

Ovakav disbalans u rezultatima očekivan je zbog toga što operatori križanja nisu bili prilagođeni formatu duge permutacije. Nakon križanja indeksi vozila često su završili preblizu jedan drugome, što je za ostala vozila značilo da moraju obići više lokacija, nadmašujući njihov kapacitet. Posljedica toga je vrlo lako stvaranje jedinke s nemogućim rješenjem što je drastično smanjilo uspješnost algoritma u pronalaženju odgovarajućih ruta.

5.2.2. Usporedba metoda evaluacije

Kao daljnje poboljšanje kratke permutacije uvedena je dvostruka provjera ruta vozila. Ona omogućava da se ista jedinka procijeni u dva smjera. Konačna dobrota jedinke bira se između dva dobivena rezultata. No, da bi se utvrdilo koliko je ova modifikacija utjecala na kvalitetu rezultata, potrebno je usporediti rezultate nakon 100 pokretanja na svim dimenzijama problema.

Za problem dimenzije 33 korištenje jednostruke i dvostruke provjere rezultiralo je jednakim uspjehom - u oba slučaja pronađen je optimum od 661 kilometra. U problemu s 48 lokacija razlika između dva najbolja rezultata iznosi 13 kilometara što je 1% vri-

jednosti optimuma. Na najvećem problemu razlika je zanemariva te iznosi samo 2 kilometra.

	33 lokacije	48 lokacija	65 lokacija
optimum	**661**	**1073**	**1174**
kratka permutacija - jednostruko	661	1112	1270
kratka permutacija - dvostruko	661	1099	1268

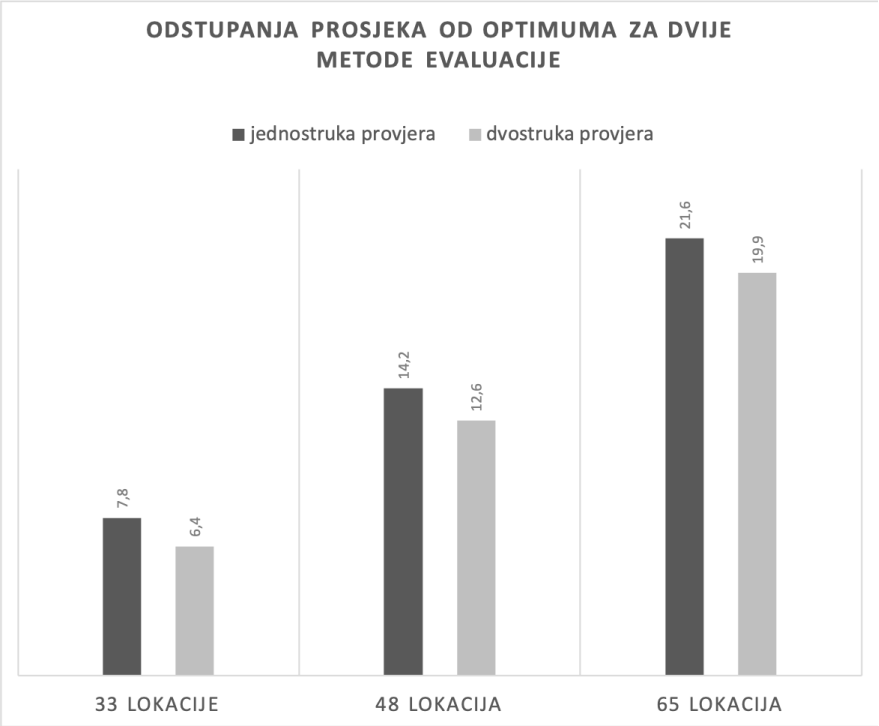
Tablica 5.2: Usporedba najboljih rezultata dvije metode evaluacije rješenja

Sudeći po ovim rezultatima, ne postoji drastična razlika između dvije metode evaluacije rješenja. Ipak, dodatna statistika iskoristiva za usporedbu može biti i prosječan rezultat nakon 100 pokretanja. Iako usporedba na temelju prosjeka nije uobičajena za genetske algoritme, dodatna usporedba metoda evaluacija ne može biti na odmet. U Tablici 5.3. prikazano je odstupanje prosječnog rezultata nakon 100 pokretanja i optimuma za konkretan problem.

	33 lokacije	48 lokacija	65 lokacija
optimum	**661**	**1073**	**1174**
kratka permutacija - jednostruko	7.8%	14.2%	21.6%
kratka permutacija - dvostruko	6.4%	12.6%	19.9%

Tablica 5.3: Usporedba prosječnih rezultata dvije metode evaluacije rješenja

Na temelju ovih podataka razlike su očitije, no potrebno je izvesti detaljnije testiranje da se utvrdi u kolikoj mjeri dodatna evaluacija pridonosi boljim rezultatima algoritma. Podaci iz Tablice 5.3. grafički su prikazani na Slici 5.1., gdje su usporedno prikazana postotna odstupanja prosjeka od optimuma za sva tri analizirana problema.



Slika 5.1: Grafički prikaz postotnog odstupanja između prosjeka i optimuma

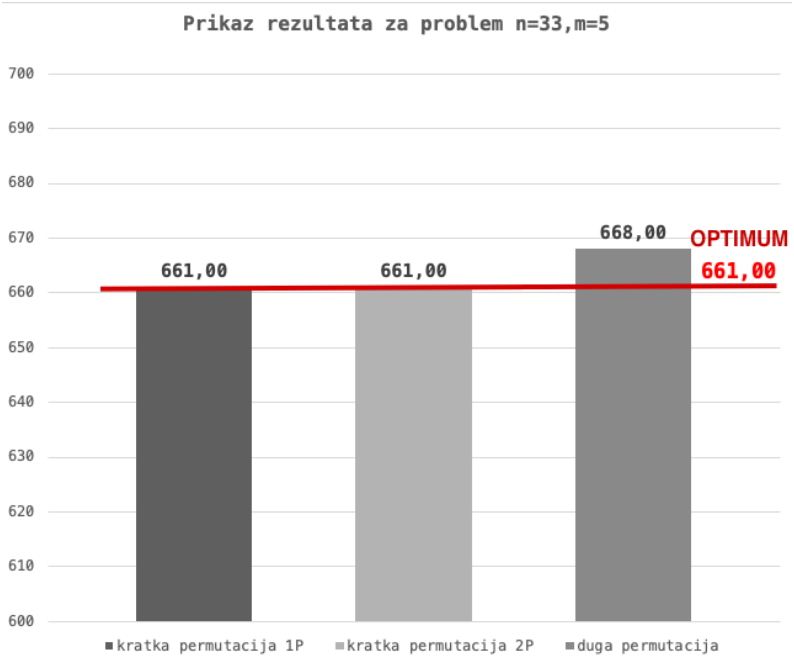
5.2.3. Sveukupna analiza

U nastavku su prikazani grafički prikazi najboljih rezultata nakon 100 pokretanja za tri različite varijante[1] korištene u implementaciji - *dugu* permutaciju, *kratku* permutaciju s jednostranom provjerom te *kratku* permutaciju s dvostranom provjerom. Rezultati su zbog preglednosti prikazani na tri odvojena grafička prikaza, svaki za jednu dimenziju problema.

Napomena: U svim grafičkim prikazima koji slijede horizontalna crvena crta s naznačenom vrijednosti označava vrijednost optimuma za promatranu dimenziju problema.

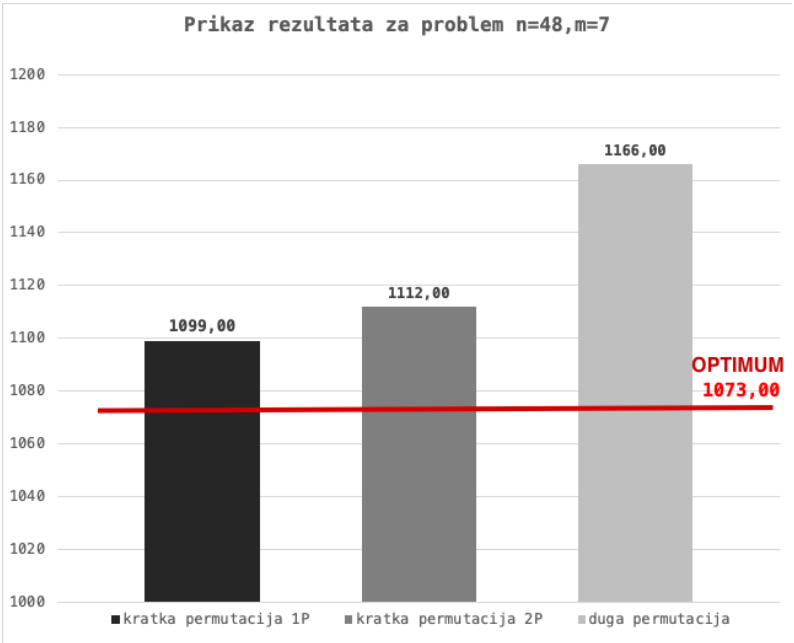
[1]Ovdje nije korišten pojam *genotip* s obzirom na to da se na istim grafičkim prikazima analiziraju rezultati istog genotipa (*kratke* permutacije), no različitog načina evaluacije jedinki (jednostruka i dvostruka provjera).

Tako su na Slici 5.2. prikazani najbolji rezultati nakon 100 pokretanja za problem dimenzije n=33, m=5.



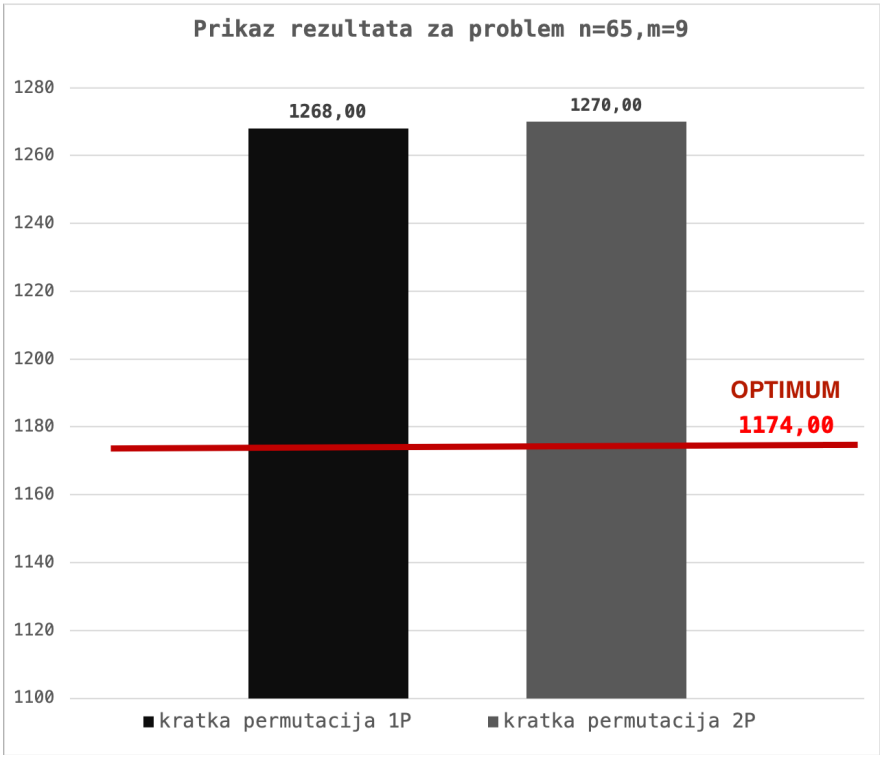
Slika 5.2: Grafički prikaz rezultata za problem n=33, m=5

Nadalje, na Slici 5.3. prikazani su najbolji rezultati nakon 100 pokretanja za problem dimenzije n=48, m=7.



Slika 5.3: Grafički prikaz rezultata za problem n=48, m=7

Konačno, na Slici 5.4. prikazani su najbolji rezultati oba načina evaluacije za *kratku* permutaciju nakon 100 pokretanja za problem dimenzije n=65,m=9. Rezultat *duge* permutacije nije vidljiv s obzirom na to da nakon 100 pokretanja nije nađeno ni jedno legalno rješenje.



Slika 5.4: Grafički prikaz rezultata za problem n=65, m=5

5.2.4. Daljnja poboljšanja

Za genetske algoritme svojstveno je određeno odstupanje od optimuma, čak i za potpuno optimizirane implementacije. Razlog tome je element slučajnosti u pretraživanju te uvjeti za prestanak algoritma. Ipak, uz manje ili veće intervencije ova konkretna implementacija mogla bi postizati bolje rezultate.

Prvi korak pri optimizaciji *duge* permutacije bio bi uvođenje prilagođenih operatora mutacije i križanja. Nakon toga uslijedilo bi testiranje rješenja i optimizacija parametara. Što se tiče *kratke* permutacije, mogućnost uvođenja novih ili poboljšanja dosadašnjih operatora mogla bi utjecati na kvalitetu rješenja. No, uvođenje novog načina evaluacije rješenja, koji bi uključivao i heuristiku te udaljenosti možda bi trebao biti prvi korak u daljnjem poboljšanju.

6. Zaključak

Važnost problema usmjeravanja vozila nije se smanjila od njegove prve formalizacije u 50-tim godinama 20. stoljeća. U međuvremenu pojavili su se mnogi pristupi rješavanju ovog NP-teškog problema. Za ovaj je rad kao metoda rješavanja odabran genetski algoritam.

Crpeći svoju inspiraciju iz teorije evolucije vrsta i populacija, genetski algoritam pretražuje prostor rješenja tako da skup rješenja predstavljen populacijom međusobno kombinira i nasumično mutira. Očuvanje kvalitetnijih rješenja rezultira približavanjem optimalnom rješenju. Pri implementaciji GA vrlo je bitno prikladno odabrati strukturu podataka koja će predstavljati rješenje. U ovom je radu kao struktura podataka korištena permutacija, tj. lista, te je implementirana u dva različita formata - *dugoj* i *kratkoj* varijanti.

Uspoređivanjem rezultata postalo je jasno da složenost problema usmjeravanja vozila uvelike ovisi o broju lokacija koje se posjećuju. Dok su za najmanji isprobani problem rezultati bili približno jednaki, na problemima veće dimenzije nedostaci jedne od izvedbi postali su jasni. Stoga je presudna dobra prilagođenost problemu ne samo genotipa, već i operatora križanja i mutacije. Ozbiljniji projekti vezani uz genetske algoritme stoga moraju realizirati prilagođene operatore te detaljno ispitati koja kombinacija operatora i parametara daje najpovoljnije rezultate.

LITERATURA

[1] *VRP-REP*. URL http://www.vrp-rep.org/datasets.html.

[2] Tonci Caric i Hrvoje Gold (I-Tech). *Vehicle Routing Problem*. In-Teh, 2008. URL https://bib.irb.hr/datoteka/433524.Vehnicle_Routing_Problem.pdf.

[3] G. B. Dantzig i J. H. Ramser. The truck dispatching problem. *Management Science, Vol. 6, No. 1 (Oct., 1959), pp. 80-91*, 1959. URL https://andresjaquep.files.wordpress.com/2008/10/2627477-clasico-dantzig.pdf.

[4] Marin Golub et al. *Genetski algoritam - Prvi dio*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2004.

[5] Domagoj Jakobović et al. *ECF Documentation*. URL http://ecf.zemris.fer.hr/documentation.html.

[6] Krunoslav Puljić i Robert Manger. *Comparison of eight evolutionary crossover operators for the vehicle routing problem*. Odjel za matematiku, Sveučilište u Zagrebu, 2013.

Usmjeravanje vozila uz pomoć evolucijskog algoritma

Sažetak

U ovom je radu obrađen problem usmjeravanja vozila. Opisane su karakteristike problema i glavne sastavnice genetskih algoritama, metode koja je korištena za rješavanje problema. Implementirano je rješenje u programskom jeziku C++, temeljeno na radnom okviru ECF. Kao genotip korištene su dvije verzije permutacije. Kvaliteta rješenja isprobana je i analizirana na 3 primjera različite dimenzije.

Ključne riječi: problem usmjeravanja vozila, genetski algoritam, selekcija, genotip, NP-težak problem

Vehicle routing using evolutionary algorithms

Abstract

In this paper the vehicle routing problem was discussed. The problem's characteristics were described as well as the main components and principles of genetic algorithms. A problem solution was implemented using the ECF framework and programming language C++. Two different genotypes were used and discussed. The solution's accuracy and possible upgrades were tested and discussed on three VRP examples of different complexity.

Keywords: vehicle routing problem, VRP, genetic algorithm, selection, genotype, NP-hard problem