SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6984

OPTIMIZACIJA PROBLEMA USMJERAVANJA VOZILA

Domagoj Lokner

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6984

OPTIMIZACIJA PROBLEMA USMJERAVANJA VOZILA

Domagoj Lokner

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6984

Pristupnik:	**Domagoj Lokner (0036510438)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Optimizacija problema usmjeravanja vozila**

Opis zadatka:

Proučiti problem usmjeravanja vozila te različite varijante tog problema. Opisati metode i načine rješavanja problema usmjeravanja vozila. Ostvariti simulator za odabranu varijantu problema usmjeravanja vozila. Odabrati i ostvariti algoritme za rješavanje problema usmjeravanja vozila. Ocijeniti učinkovitost odabranih metoda na skupu problema usmjeravanja vozila iz literature. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 12. lipnja 2020.

SADRŽAJ

1. Uvod

Problem raspoređivanja jedan je od najčešćih problema kombinatorne optimizacije. Kako živimo u svijetu koji je okružen vremenskim okvirima, svjesni smo kako je efikasno raspoređivanje našega vremena važna komponenta o kojoj na veliko utječe kvaliteta života kojeg ćemo voditi. Naravno, raspoređivanje ne moramo samo promatrati u okvirima vremena, organizacija prostora je također problem s kojim se svakodnevno susrećemo.

Kako je razvojem čovječanstva život čovjeka postao kompleksniji tako su i procesi raspoređivanja postali složeniji. Ovaj rad će se baviti jednim problemom raspoređivanja koji je poznat kao problem usmjeravanja vozila. Glavno pitanje na koje pokušavamo odgovoriti tražeći rješenje ovoga problema bit će kako uspješno rasporediti flotu vozila kojoj je zadaća obići određen broj kupaca, a da pri tome nastojimo optimizirati određene kriterije.

Ovaj rad će pobliže objasniti problem i ponuditi dvije metode koje će pokušati odgovoriti na njegove izazove i ponuditi rješenje za efikasno usmjeravanje vozila.

2. Problem usmjeravanja vozila

Problem usmjeravanja vozila (engl. *Vehicle routing problem*), to jest *VRP* je problem kombinatorne optimizacije kojim se pokušava optimizirati put flote vozila čija je zadaća poslužiti određen broj klijenata. Već iz ovako grube definicije problema jasna je praktična korist koju bismo ostvarili pronalaskom optimalnog načina na koji bi problem bio rješiv te će u tome biti sadržana glavna motivacija rada.

Usmjeravanje vozila spada u kategoriju *NP-teških* problema, to jest problem je težak barem koliko i najteži problem u kategoriji *NP* problema[1]. Zahvaljujući napretku tehnologije i računarske znanosti danas je moguće doskočiti ovakvim izazovima.

Problem se prvi puta pojavljuje 1959. godine u studiji Georgea Dantziga i Johna Ramsera gdje je bio primijenjen na problem dostave benzina. Kao dio ove studije također je predložen i prvi algoritamski pristup rješavanja VRP-a.

Problem usmjeravanja vozila možemo proučavati kao poopćenu varijantu problema trgovačkog putnika (engl. *Travelling salesaman problem*), to jest TSP, u kojoj na raspolaganju možemo imati više od jednog trgovačkog putnika. TSP je najčešće spominjan problem kombinatorne optimizacije, a za cilj ima odrediti redoslijed obilaska lokacija pri čemu nastoji minimizirati sve troškove pri čemu se nikada ne smije vratiti na već posjećenu lokaciju. Cilj problema usmjeravanja vozila je isti, no ono za želju ima i minimizaciju broja vozila potrebnih za posjet svih kupaca.

Gornja definicija VRP-a nameće jedno važno pitanje. Što znači minimizirati troškove puta? Ovo pitanje je osnova na kojoj će se zasnovati različite varijante problema, ovisno o tome koje komponente će utjecati na trošak puta vozila.

2.1. Formalna definicija problema

Kako bismo lakše pristupili rješavanju problema uvest ćemo formalne oznake koje će detaljno opisivati problem.

[1] NP je klasa kompleksnosti u koju spadaju svi problemi koji su rješivi u neodređenom polinomijalnom vremenu.

V skup vrhova, to jest klijenata/poslova (v_0, ..., v_n), pri čemu vrh v_0 predstavlja depo

V' skup vrhova kojem pripadaju isključivo klijenti (v_1, ..., v_n)

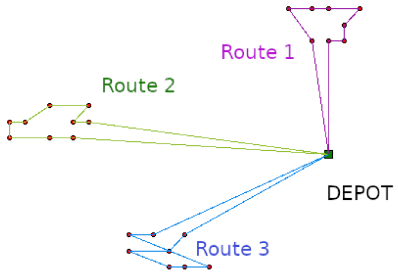
A = $\{(v_i, v_j)|v_i, v_j \in V; i \neq j\}$ set udaljenosti među vrhovima

C matrica udaljenosti među vrhovima

d vektor potražnje po klijentima (d_1, ..., d_n)

m broj međusobno identičnih vozila

R vektor ruta po vozilu (R_1, ..., R_m)



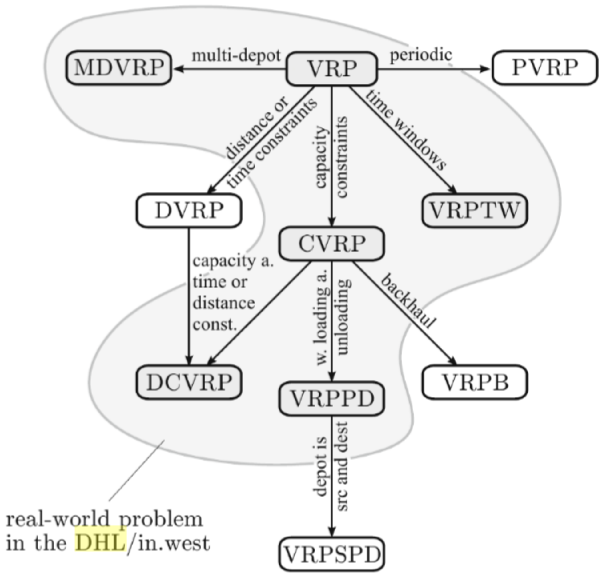
Slika 2.1: Vizualizacija rješenja VRP-a

2.2. Inačice problema

Zadaća VRP-a je minimizacija broja vozila i ukupne cijene puta. S obzirom na mnogobrojne uvjete koji mogu utjecati na određivanje ovih parametara, VRP je rezultirao različitim inačicama koje objedinjuju pojedine uvjete.

Osnovna inačica, koju podrazumijevamo pod nazivom VRP, uzima u obzir samo broj vozila i duljinu puta koju vozila prijeđu. Realne situacije pred nas stavljaju puno više uvjeta. Slika 2.2 prikazuje hijerarhiju VRP inačica te izdvaja one koje su korištene unutar DHL poduzeća za dostavu paketa.

U nastavku ovog poglavlja su objašnjene dvije najutjecajnije i najčešće inačice problema.



Slika 2.2: Inačice VRP-a i njihova uporaba u DHL poduzeću za dostavu

2.2.1. CVRP

Usmjeravanje vozila s ograničenim kapacitetom dobara (engl. *Capacited VRP*) je inačica VRP-a koja podrazumijeva da svako vozilo ima ograničeni kapacitet dobara koje može prenijeti. CVRP postavlja novi uvjet na rutu, svako vozilo može poslužiti samo kupce čija je ukupna potražnja za dobrima manja od maksimalnog kapaciteta vozila.

2.2.2. VRPTW

Usmjeravanje vozila s vremenskim prozorima (engl. *VRP with Time Windows*) je inačica u kojoj se uz svakog kupca $v \in V$ veže vremenski interval unutar kojega kupac mora biti poslužen i vrijeme koje je potrebno da se obavi istovar tereta.

VRPTW uzima u obzir da je potrebno određeno vrijeme za prevaliti put između dva kupca, te da vozila treba rasporediti tako da svaki kupac bude poslužen u terminu u kojem je dostupan. Ova inačica ostavlja mogućnost čekanja na istovar sve do trenutka kada će kupac biti spreman preuzeti robu.

3. Algoritam ušteda

Clarke i Wright su 1964. godine predložili algoritam koji se nadogradio na Danzigovo i Ramserovo algoritamsko rješenje VRP-a iz 1959. Ovaj algoritam pohlepne heuristike je poznat kao *Clarke i Wrightov algoritam ušteda* (engl. *Clarke and Wright savings algorithm*).

Algoritam je zasnovan na *uštedama* (engl. *saving*). Krenimo od pretpostavke da svakoga klijenta $v \in V'$ poslužuje samo jedno vozilo kojemu je početna i završna točka depo $D = v_0$. Ako funkcija l određuje vrijednost između dvije točke, tada će ukupan put vozila iznositi

$$2\sum_{i=1}^{n} l(D, v_i)$$

Ako ovaj scenarij izmijenimo tako da jedno vozilo posluži dva klijenta i i j, cijeli iznos će se umanjiti za vrijednost

$$\begin{aligned} s(v_i, v_j) &= 2l(D, v_i) + 2l(D, v_j) - (l(D, v_i) + l(v_i, v_j) + l(D, v_j)) \\ &= l(D, v_i) + l(D, v_j) - l(v_i, v_j) \end{aligned} \tag{3.1}$$

Vrijednost $s(v_i, v_j)$ se naziva *ušteda* kojom će rezultirati kombiniranje klijenata i i j u istu rutu. Što je ušteda veća to će imati veći prioritet u ovome algoritmu. Algoritam će pokušati grupirati što je više moguće veza između kupaca za koje je izračunata ušteda što veća, a da se pri tome ne narušava niti jedno ograničenje. Zbog toga što algoritam prvo pokušava grupirati veće uštede spada u kategoriju pohlepnih heuristika. Slika 3.1 prikazuje grupiranje više kupaca u jednu rutu čime bi algoritam rezultirao, na predstavljenom problemu, ako je kapacitet vozila dovoljan za posluživanje dva klijenta. Algoritam 1 detaljno opisuje sve korake algoritma.

[1] klijent je vanjski dio rute ako je direktno povezan s depoom

Algorithm 1 Clarke and Wright savings algorithm

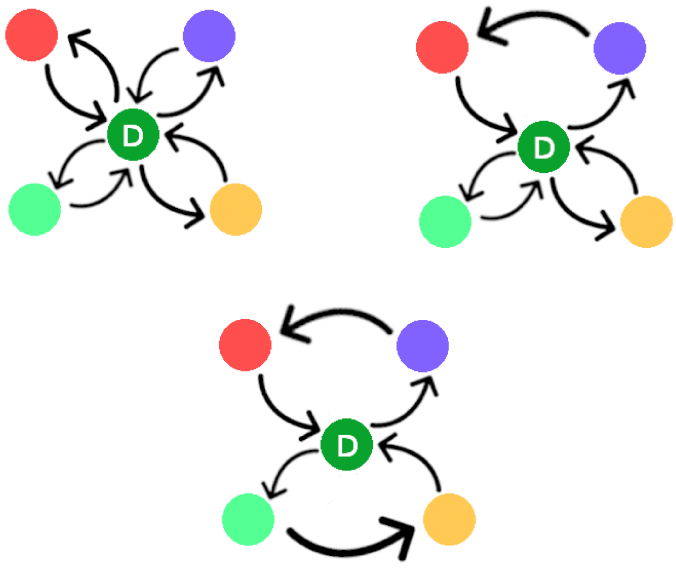
Korak 1: Izračunaj uštede za sve parove (i, j) za koje vrijedi $i, j \in V$ i $i \neq j$

Korak 2: Sortiraj izračunate uštede silazno od najveće prema najmanjoj

Korak 3: Za svaku uštedu $s(i, j)$, iz sortiranog niza ušteda, uključi poveznicu (i, j) u rutu ako niti jedno ograničenje neće biti narušeno i ako vrijedi:

1. Niti i, niti j nisu već dio postojeće rute. Tada stvori novu rutu koja će sadržavati vezu (i, j)
2. Ili, samo je jedan klijent uključen u rutu i nalazi se na *vanjskom*[1] dijelu te rute. Tada dodaj vezu (i, j) na tu rutu.
3. Ili, su oba klijenta, i i j, dio postojećih ruta te su obadva *vanjski* dio ruta. Tada povezujemo te dvije rute vezom (i, j).

Korak 4: Svi kupci koji su preostali, to jest nisu pridodani niti jednoj ruti, pridjeljuju se vozilima koja će poslužiti samo njih i vratiti se u depo, drugim riječima stvaraju zasebnu rutu.



Slika 3.1: Vizualizacija djelovanja algoritma ušteda - Algoritam 1

4. Genetski algoritam

Genetski algoritam je jedan od najpoznatijih evolucijskih algoritama[1]. Genetski algoritam je metaheuristički[2] algoritam koji je inspiriran procesom prirodne selekcije bazirane na Darwinovoj teoriji evolucije[3].

Algoritam u svojoj bazi se koristi evolucijskom strategijom (engl. *Evolution strategy*) (algoritam 2). U prirodnoj selekciji jedinke koje imaju najbolje predispozicije preživljavaju te imaju priliku za stvaranje potomaka koji će imati, u određenoj mjeri, njima specifične karakteristike. Ovaj proces sastoji se od dva dijela, selekcije i križanja. Kako je već spomenuto, genetski algoritam će biti zasnovan na istome principu.

Genetski algoritam najčešće započinje stohastičkim stvaranjem početne populacije. Populaciju čini skupina jedinki, a svaka jedinka predstavljena kromosomom. Nakon inicijalizacije početne populacije potrebno je odrediti koliko je svaka jedinka zadovoljavajuća, to jest koliko je blizu prihvatljivom rješenju problema. Tu vrijednost ćemo definirati uz pomoć *funkcije dobrote* (engl. *fitness function*) koja će će prihvatljivost pokušati preslikati u numeričku vrijednost. Potom, algoritam prelazi na strategiju selekcije, pokušavajući tako odrediti koje jedinke će dobiti priliku za križanje što će u konačnici rezultirati novom generacijom. Križanje je operator genetskog algoritma koji će kombinacijom gena dvaju roditelja generirati potomka. Nove populacije osim potomaka mogu sadržavati i najbolje jedinke koje su prenesene iz prošle populacije zbog najboljih karakteristika, te ovo svojstvo nazivamo *elitizmom*. Ovakav postupak se iterativno ponavlja za svaku sljedeću generaciju.

Problem na koji često nailazimo u genetskom algoritmu je da sve jedinke postanu slične te više ne ostvarujemo napredak populacije evolucijom, a da pri tome rješenje do kojega smo došli nije još nije blizu zadovoljavajućem. Ovaj problem rješavamo tako da stvorimo dovoljno veliku populaciju, tako garantirajući raznolikost jedinki.

[1] evolucijski algoritmi - metaheuristički optimizacijski algoritmi inspirirani biološkom evolucijom.

[2] metaheuristika - metoda koja je generalno primjenjiva na širi skup problema.

[3] Prva moderna teorija evolucije koju je predložio Charles Robert Darwin u djelu *On the Origin of Species*

Uz dobar izbor veličine populacije, uvodimo i novi operator, mutaciju. Mutacija će prema određenom kriteriju unositi stohastičke izmijene gena svake jedinke. Kako je mutacija poprilično destruktivan operator, vjerojatnost njezine pojave je najčešće jako mala, između 2% i 15%.

Algorithm 2 Evolucijska strategija

Ulaz: $populacija$=populacija, $elite$=broj elitistički odabranih jedinki

Izlaz: nova populacija

`//odaberi roditelje`

$parents = select(population, (population.size - elite) * 2)$

`//stvori potomke križanjem roditelja`

$offsprings = crossover(parents)$

`//stvori novu populaciju`

$newPopulation = newPopulation(offsprings)$

`//mutiraj pojedine gene odabranih jedinki`

$mutatePopulation(newPopulation)$

`//dodaj elitistički odabrane jedinke`

$newPopulation.append(elitisticSelect(population, elite))$

return $newPopulation$

Algorithm 3 Genetski algoritam

Ulaz: max=broj evaluacija rješenja
$size$=veličina populacije
$elite$=broj elitistički odabranih jedinki
Izlaz: najbolja jedinka
$population := createInitPopulation(size)$
$best := NULL$
$i := 0$
while ture **do**
  if $i > 0$ **then**
    $population = evolutionStrategy(population, elite)$
  end if
  for $individual \in population$ **do**
    $i := i + 1$
    if $i = max$ **then**
      return $best$
    else
      $individual.fitness = fitnessFunction(individual)$
      if $individual = NULL \vee individual.fitness < best.fitness$ **then**
        $best := individual$
      end if
    end if
  end for
end while

Nakon definiranja genetskog algoritma postavlja se pitanje kako ga upotrijebiti za rješavanje VRP-a? Glavni cilj ostatka rada će biti odgovoriti na ovo pitanje.

4.1. Implementacija

4.1.1. Prikazi rješenja

Populacija genetskog algoritma se sastoji od određenog broja jedinki, to jest kromosoma. Kako je to uobičajeno, kromosom će predstavljati niz brojeva. U ovom poglavlju bit će definirani načini na koje ćemo dekodirati kromosome tako da ih možemo interpretirati kao moguće scenarije VRP-a, a tako i kao moguće rješenje.

Kodiranje slučajnim ključem

Kodiranje slučajnim ključem (engl. *Random key encoding*), to jest RKE, koristi jednu listu realnih brojeva za prikazivanje rasporeda vozila. Svi brojevi unutar liste moraju biti u rasponu $[0, m\rangle$, a veličina liste će odgovarati broju klijenata, to jest lista će biti dugačka n. Svaki indeks polja će predstavljati jedan posao, odnosno klijenta kojega treba poslužiti.

2.53	0.31	0.12	1.97	0.81	1.43	2.28

Slika 4.1: Raspored kodiran slučajnim ključem

Prema rasporedu na slici 4.1 klijentu v_0 će odgovarati broj 2.53. Brojevi zapisani u listi će biti interpretirani tako da ćemo cijeli dio realnoga broja interpretirati kao redni broj vozila kojemu je pridružen posao odgovarajućeg indeksa liste, dok će decimalni dio određivati poredak posla. Poslovi čija vrijednost ima manji decimalni broj će biti prioritetni. Objasnimo ovo pobliže na primjeru slike 4.1. Klijenti v_0 ($\mathbf{2}.53$) i klijent v_6 ($\mathbf{2}.28$) će biti posluženi istim vozilom, u ovome slučaju, to će biti vozilo $m_{\mathbf{2}}$. Također, vozilo m_2 će prema rasporedu prvo posjetiti klijenta v_6 ($2.\mathbf{28}$), a zatim klijenta v_0 ($2.\mathbf{53}$).

Kodiranje pomičnim zarezom

Kodiranje pomičnim zarezom (engl. *Floating point encoding*), FPE, raspoređivanje obavlja na temelju jednog niza realnih brojeva. Svi brojevi u nizu moraju biti iz intervala $[0, 1]$, a broj elemenata niza će odgovarati broju klijenata, to jest broju n.

0.85	0.21	0.13	0.57	0.29	0.44	0.71

Slika 4.2: Raspored kodiran pomičnim zarezom

Kako bismo odredili kojem vozilu pripada pojedini posao, interval $[0, 1]$ ćemo podijeliti na m jednakih dijelova. Svi poslovi koji se nalaze unutar intervala j se pripisuju vozilu m_j, a posao s najmanjom vrijednošću će se obaviti prvi. Objasnimo ovo na rasporedu sa slike 4.2, uz pretpostavku da na raspolaganju imamo 3 vozila. Vozilu

m_0 će pripasti svi poslovi iz intervala $[0, 0.33\rangle$, u ovome slučaju to su poslovi v_1 (0.21), v_2 (0.13) i v_4 (0.29), a raspored njihova izvršavanja će biti v_2, v_1 i v_4.

4.1.2. Operatori

U ovoj sekciji će biti objašnjeni svi implementirani operatori genetskog algoritma. U sklopu konkretne implementacije rada svi operatori su izvedeni kao strategije koje se predaju genetskome algoritmu na daljnju uporabu u svrhu evolucije.

Selekcije

K-turnirska selekcija (engl. *K-tournament selection*) je selekcija koja nasumično, prema uniformnoj raspodjeli, odabire k jedinki iz populacije te među njima izabire onu koja ima najbolju vrijednost funkcije dobrote.

Križanja

Operatori križanja (engl. *crossover*), pomoću kojih se dobivaju nove generacije genetskoga algoritma su se pokazali kao najvažnija komponenta evolucije. U ovome odlomku su objašnjena sva implementirana križanja, a za njihov opis će biti korištena sljedeća notacija:

β^{f_1} prvi roditelj s genima $\beta_1^{f1}, ..., \beta_n^{f1}$

β^{f_2} drugi roditelj s genima $\beta_1^{f2}, ..., \beta_n^{f2}$

β^s potomak $\beta_1^s, ..., \beta_n^s$

Discrete Crossover je vrsta križanja koja β_i^s izabire nasumično između β_i^{f1} i β_i^{f2} za svaki i za koji vrijedi $1 \leq i \leq n$.

Simple Arithmetic Crossover ovo križanje prvo izabire vrijednost k, zatim se prvih k vrijednosti potomka prekopiraju od jednog nasumično odabranog roditelja po tom se ostali geni računaju kao aritmetička sredina između gena roditelja (formula 4.1).

Whole Arithmetic Crossover je vjerojatno najčešće korišteni operator, a potomak se izračunava kao težinska suma dva roditelja s istim parametrom α.

$$\beta^s = \alpha \cdot \beta^{f_1} + (1 - \alpha) \cdot \beta^{f_2} \tag{4.1}$$

Local Crossover ovo križanje je slično *Whole arithmetic crossoveru* uz to da je novi α izabran nasumično za svaki pojedini gen.

Flat Crossover je križanje gdje je potomak β^s generiran tako da se vrijednosti gena nasumično izabiru iz intervala $\left[\beta^{f_1}, \beta^{f_2}\right]$.

SBX Crossover Simulirano binarno križanje (engl. *Simulated Binary Crossover*) je križanje koje pokušava simulirati binarno križanje u jednoj točki. Dva potomka, β^{s_1} i β^{s_2}, su generirana na sljedeći način

$$\beta^{s_{1,2}} = \frac{1}{2}\left[(1 \pm B_k)\beta^{f_1} + (1 \mp B_k)\beta^{f_2}\right]$$

gdje je B_k koeficijent ≥ 0, a dobiven je prema sljedećoj formuli za slučajno odabrani parametar u iz intervala $\langle 0, 1\rangle$

$$B(u) = \begin{cases} (2 \cdot u)^{\frac{1}{\eta+1}} & \text{ako je } u \leq \frac{1}{2} \\ (\frac{1}{2 \cdot (1-u)})^{\frac{1}{\eta+1}} & \text{ako je } u > \frac{1}{2} \end{cases}$$

U konkretnoj implementaciji potomak kojim će rezultirati križanje će biti izabran nasumično β^{s_1} ili β^{s_2}.

BLX-alpha Crossover (engl. *Blend alpha crossover*) generira potomke tako da uzima uzorak iz raspona $[min_i - I \cdot a, max_i + I \cdot a]$ za svaki gen i. Vrijednosti max_i i min_i odgovaraju i-tom roditeljskom genu veće, to jest manje vrijednosti, a I apsolutnoj vrijednosti njihove razlike. Konačno, potomak se generira na sljedeći način

$$\beta^s = (min - I \cdot a) + \alpha \cdot |(max + I \cdot a) - (min - I \cdot a)|$$

uz korisnički definiran parametar a, te vrijednost α slučajno izabranu iz intervala $[0, 1]$ za svaki pojedini gen.

Combined Crossover je križanje koje prima listu objekata koji predstavljaju konkretne strategije križanja te pri svakom pozivu nasumično izabire jednu strategiju iz liste koja će biti upotrijebljena.

Mutacije

U sklopu ovoga rada implementirane su tri različite mutacije koje mogu utjecati na promjenu gena.

1. Zamjena gena s drugim genom kromosoma koji se nasumično izabire prema uniformnoj raspodjeli (*SwapGenesMutation*).

2. Promjena gena u nasumično odabranu vrijednost iz određenog intervala (*RandomMutation*).

3. Dodavanje nasumično odabrane vrijednosti iz intervala odabranom genu (*AddRandomMutation*).

4.1.3. Funkcija dobrote

Pri simuliranju svake strategije obilaska klijenata potrebno je odrediti kvalitetu rute, to jest koliko je određeni raspored vozila prihvatljiv. To ćemo učiniti jednostavnom funkcijom koja će numerički odrediti ukupnu cijenu nekakvog rasporeda vozila. Takvu funkciju nazivamo funkcija dobrote (engl. *fitness function*) (algoritam 4), a ona će biti izračunata za svaku pojedinu rutu koju jedinka definira. Uvijek ćemo težiti opciji za koju će suma vrijednosti funkcije imati manju vrijednost (kako je već definirano algoritmom 3).

Algorithm 4 Funkcija dobrote

Ulaz: $capacity$=ukupan kapacitet, $overcapacity$=prekoračenje kapaciteta vozila,
$distance$=prijeđena udaljenost,
$overtime$=vrijeme provedeno čekajući, $late$= broj kašnjenja an destinaciju
Izlaz: vrijednost funkcije dobrote
$fitness = distance + 1000 \cdot overcapacity$
if $timeWidow$ **then**
  $fitness = fitness + overtime$
  if $late > 0$ **then**
    $fitness = fitness \cdot late \cdot numerOfVehicles$
  end if
end if
return $fitness$

Kako je prikazano algoritmom 4, prekoračenje kapaciteta vozila ćemo skupo kažnjavati množenjem s 1000 dok će najskuplja opcija biti kašnjenje na odredište, to jest neuspješno posluživanje klijenta.

4.2. Instance problema

Za prikaz jedne instance problema korištene su *Hombergerove instance*. Slika 4.3 prikazuje jedan primjer takve instance.

Hombergerove instance se mogu podijeliti u tri kategorije ovisno o raspodijeli klijenata.

1. **C** – grupirani kupci (engl. *Clustered customers*)
2. **R** – uniformno raspodijeljeni kupci(engl. *Uniformly distributed customers*)
3. **RC** – kombinacija C i R tipa

C tip grupira određeni podskup kupaca na manjem teritoriju, a vremenski prozori se određuju prema već poznatim rješenjima. R tip lokaciju svakoga kupca određuje nasumično na cijeloj raspoloživoj površini.

C101

VEHICLE

NUMBER	CAPACITY
25	200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE	TIME
0	40	50	0	0	1236	0	
1	45	68	10	912	967	90	
2	45	70	30	825	870	90	
...							

Slika 4.3: Instanca problema

4.3. Analiza rezultata

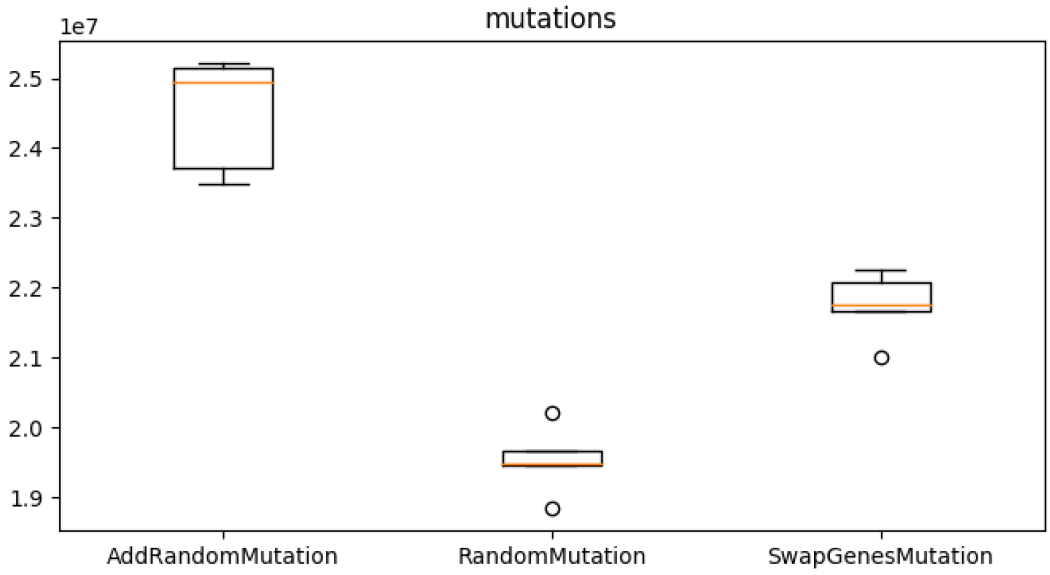
Rezultati koji će biti prikazani u nastavku su produkt genetskog algoritma čija je populacija postavljena na 50 jedinki, a maksimalan broj evaluacija rješenja na 58800, odnosno 1200 evolucija uz to da će najbolja jedinka uvijek biti prenesena u sljedeću generaciju. Broj evaluacija je određen tako da se izračunom rješenja s većim brojem evaluacija ustanovilo kako nema značajnog napretka rezultata.

Pri određivanju najboljih operatora, križanja i mutacije, korištena je instanca problema tipa *C*, konkretno problem C_1_4_1, i prikaz rezultata kodiranjem slučajnim ključem.

Kako je za stvaranje realne slike o rezultatima genetskoga algoritma potrebno napraviti višestruke izračune uz iste postavke, većina rješenja će biti prikazana pomoću *boxplot* dijagrama u kojima je jedan zapis nastao kao produkt 5 istovrsnih rezultata.

4.3.1. Usporedba mutacija

Grafikon na slici 4.4 prikazuje različit utjecaj mutacija uz istovrsno križanje *SBX Crossover* pri vjerojatnosti mutacije od 1%. Iz dijagrama je vidljivo da mutacija izmjene gena slučajnom vrijednošću (*RandomMutation*) rezultira najuspješnijim riješenima.



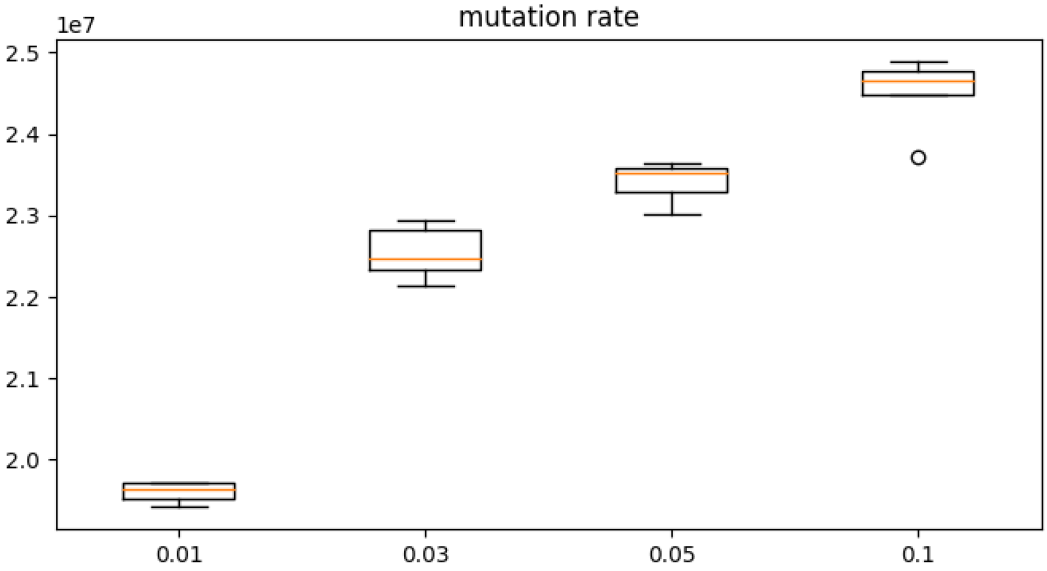
Slika 4.4: Utjecaj različitih mutacija

U nastavku bismo voljeli znati kakav utjecaj na našu populaciju ima vrijednost vjerojatnosti mutacije gena (engl. *mutation rate*). Upravo tu ovisnost prikazuje dijagram na slici 4.5, iz njega možemo zaključiti da je mutacija od 0.01 najprihvatljivija, a povećavanjem te vrijednosti ne pridonosimo uspješnosti algoritma.

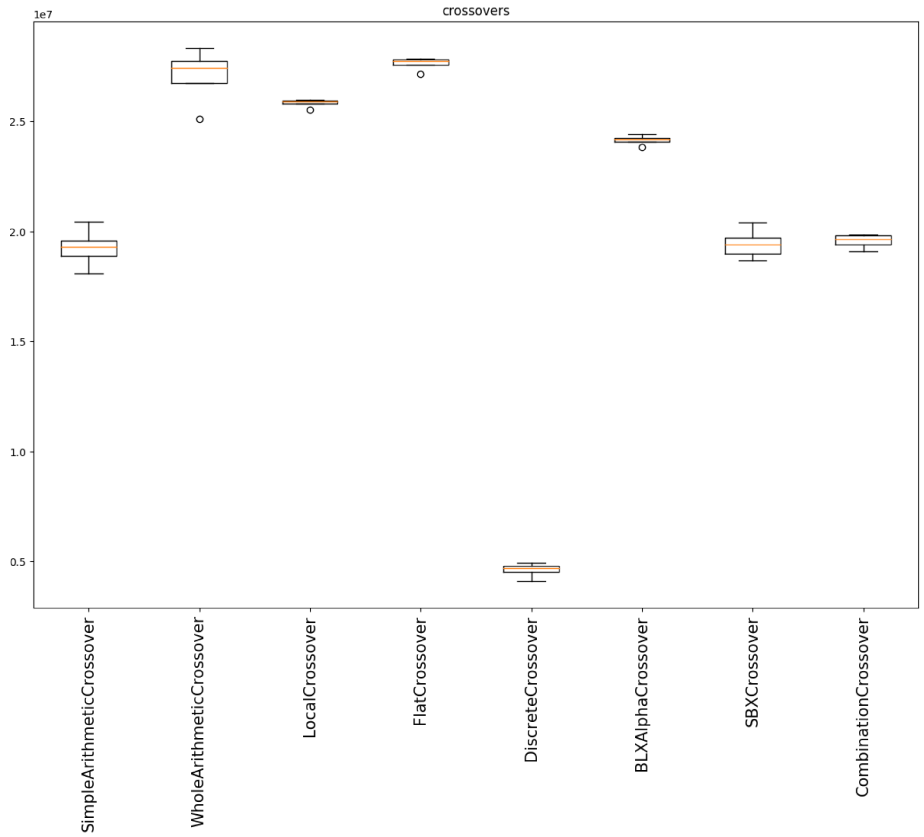
4.3.2. Usporedba križanja

Križanje se pokazalo kao najutjecajniji operator genetskog algoritma te će odabir ispravnog križanja biti ključan za uspješnu optimizaciju.

Dijagramu slike 4.6 prikazuje usporedbu više križanja na istoj instanci problema, uočljivo je da *Simple Arithmetic Crossover*, *SBX Crossover* i njihova kombinacija



Slika 4.5: Utjecaj različite vjerojatnosti mutacije za *RandomMutation*



Slika 4.6: Usporedba različitih križanja

Combination Crossover daju bolje rezultate u usporedbi s ostalim križanjima. *Discrete Crossover* se izdvaja dajući znatno bolje rezultate nad svim ostalim križanjima.

Konačno, uz odabrani operator križanja završili smo s odabirom operatora genetskog algoritma. Tako da ćemo na ostalim testnim primjerima za različite instance problema koristiti ***Descrete Crossover*** križanje i ***Random Mutation*** mutaciju uz vjerojatnost mutacije od 0.1%.

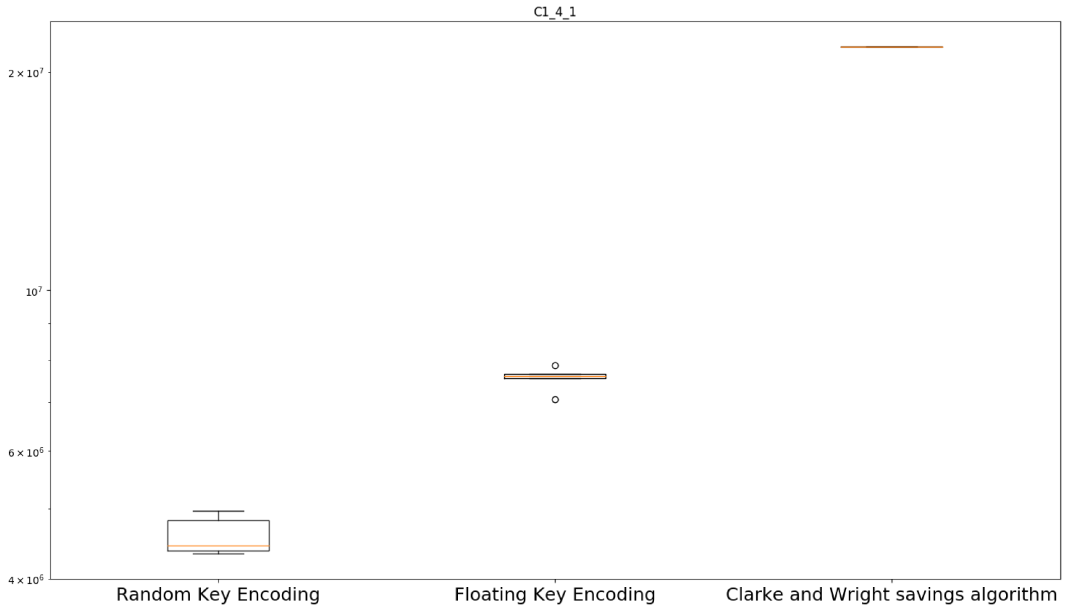
4.3.3. Usporedba rezultata na različitim instancama problema

Nakon što smo definirali sve potrebne operatore i parametre genetskoga algoritma zanimat će nas kako će algoritam reagirati na ostale klase problema. Također, želimo usporediti kodiranje slučajnim ključem i kodiranje pomičnim zarezom. Dijagrami na slikama 4.7, 4.8 i 4.9, uspoređuju rezultate dobivene genetskim algoritmom uz različite prikaze te algoritmom ušteda za tri odabrane instance iz svake kategorije problema (*C_1_4_1, R_1_4_1 i RC_1_4_1*).

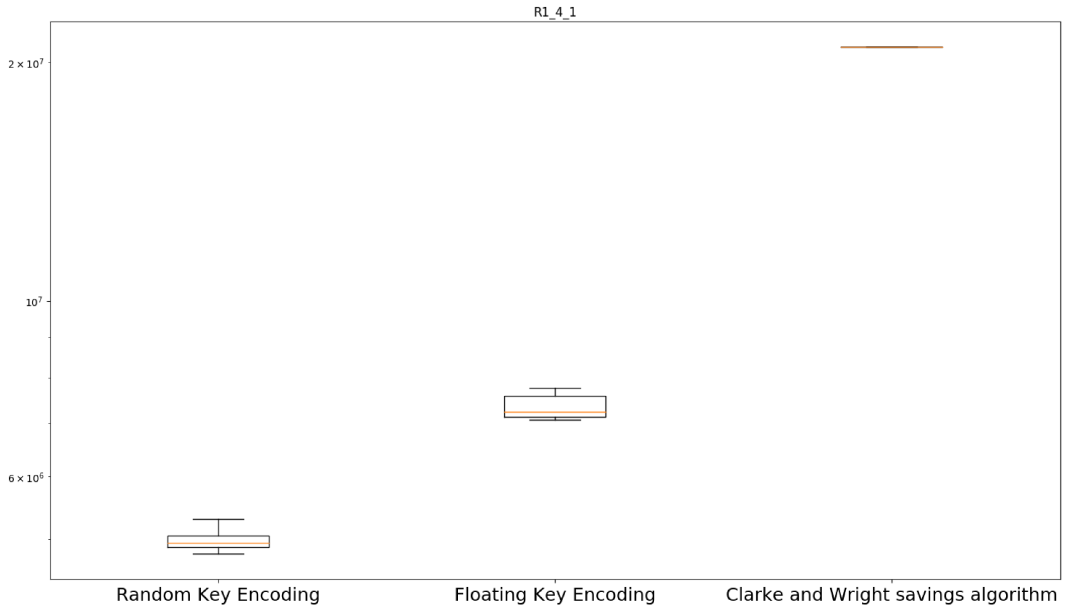
Tablica 4.1 prikazuje sve rezultate koji su prikazani na dijagramima (slike 4.7, 4.8, 4.9). Uvidom u tablicu vidljivo je da genetski algoritam rezultira boljim rješenjima u usporedbi sa Clark i Wrighteovom heuristikom, dok je prikaz rješenja kodiranih slučajnim ključem bolji od prikaza kodiranih pomičnim zarezom.

Tablica 4.1: Rezultati različitih algoritama po instancama problema

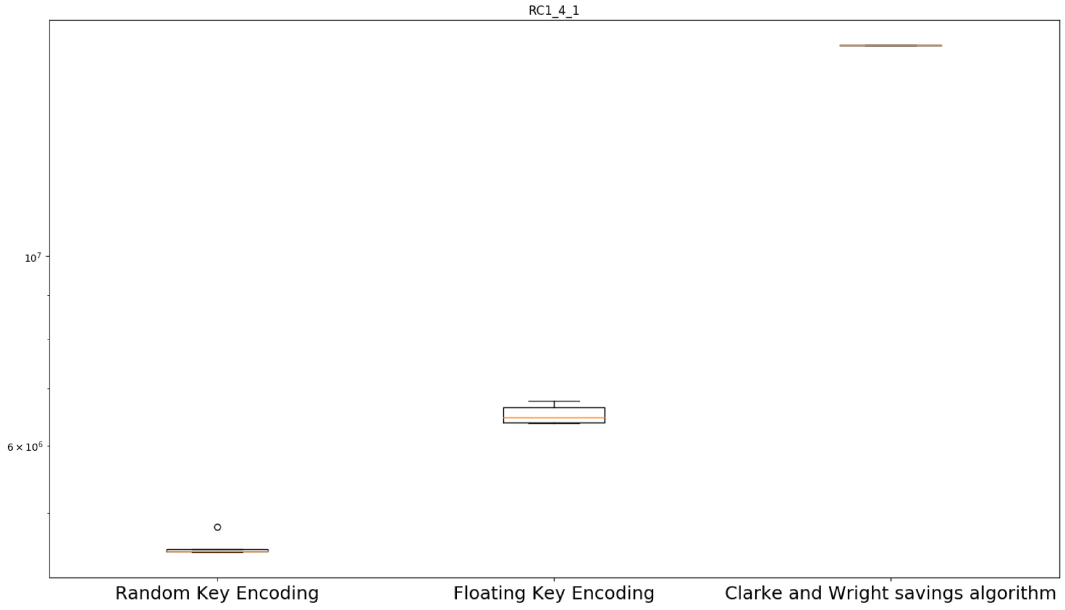
algoritam	vrijednost	C	R	RC
RKE	minimum	4 329 882	4 792 491	4 493 631
	prosjek	4 587 390	4 996 640	4 570 151
	maksimum	4 967 740	5 306 096	4 819 033
FPE	minimum	7 072 007	7 081 652	6 364 000
	prosjek	7 557 567	7 360 116	6 525 733
	maksimum	7 875 331	7 764 609	6 761 640
C-W	rezultat	21 652 588	20 895 398	17 678 605



Slika 4.7: Rezultati rješenja za instancu problema iz klase C



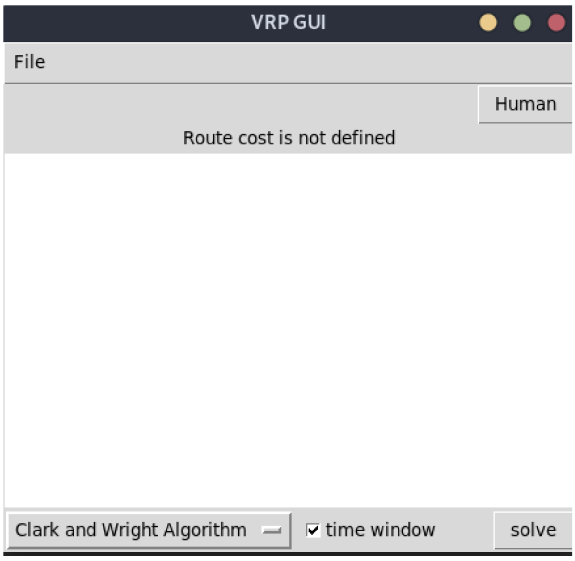
Slika 4.8: Rezultati rješenja za instancu problema iz klase R



Slika 4.9: Rezultati rješenja za instancu problema iz klase RC

5. Grafičko korisničko sučelje

Kao dio implementacije izrađeno je grafičko korisničko sučelje u svrhu prikaza problema usmjeravanja vozila i manipulacije implementiranim algoritmima (slika 5.1).

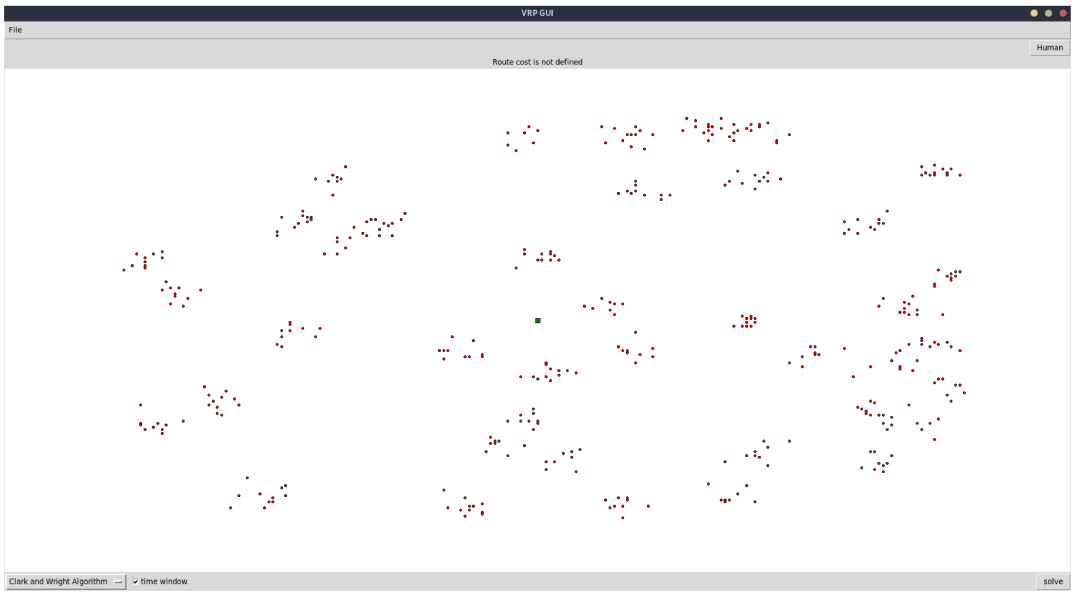


Slika 5.1: Korisničko grafičko sučelje

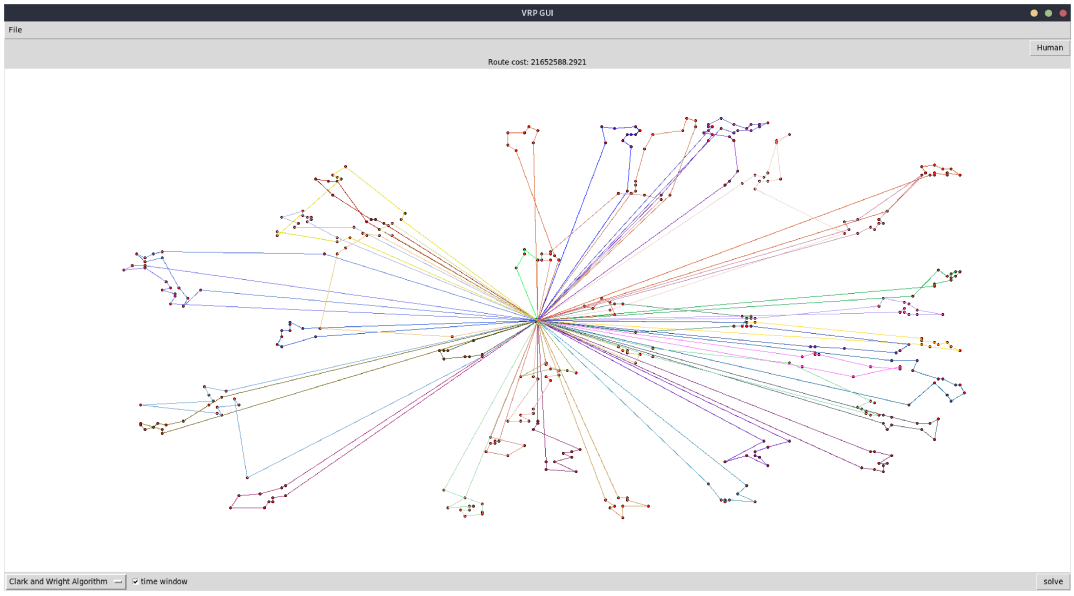
Sučelje omogućuje učitavanje i spremanje instance problema, isto kao učitavanje i spremanje rješenja problema (izbornik *Files*). Ručni unos instance problema je moguće nakon pritiska na dugme *Human*, daljnjim pritiscima na središnji dio sučelja unose se prvo depo, a zatim proizvoljan broj kupaca.

Rješavanje problema algoritmima je moguće pomoću pritiska na dugme *solve* pri čemu će program započeti rješavanje algoritmom koji je odabran u padajućem izborniku. Po završetku rješenje problema će se iscrtati na ekranu.

Kada je rješenje problema ispisano na ekranu, u gornjoj traci će se ispisati vrijednost funkcije dobrote za učitano rješenje.



Slika 5.2: Prikaz učitane instance problema



Slika 5.3: Prikaz vizualizacije rješenja VRP-a pomoću algoritma ušteda

6. Zaključak

Ovaj rad se bavi opisom problema usmjeravanja vozila, isto kao i opisima dviju metoda kojima se pokušava dobiti njegovo rješenje.

Prva opisana metoda je Clarkeov i Wrighteov algoritam ušteda koji uz pomoć pohlepne heuristike pokušava ponuditi riješenje VRP-a.

Kao drugi pristup rješavanju problema je objašnjen genetski algoritam. Nakon opisa načela i motivacije za uporabu genetskog algoritma detaljno su opisani svi operatori koji su ostvareni u ovome radu. Rezultati dobiveni uporabom genetskog algoritma su opisani, prikazani grafički te su stavljeni u odnos sa algoritmom ušteda. Kao zaključak je izvedeno da genetski algoritam rezultira boljim rezultatima, te kako kodiranje nasumičnim ključem bolje opisuje rješenje za razliku od kodiranja pomičnim zarezom, a rezultati su najuspješniji za *Descrete Crossover* i *RandomMutation* operatore.

Na kraju rada je prikazano ostvareno grafičko korisničko sučelje te su ukratko objašnjene njegove funkcionalnosti.

LITERATURA

[1] Domagoj Jakobović Ivan Vlašić, Marko Đurasević. *A comparative study of solution representations for the unrelated machines environment*. Faculty of Electrical Engineering and Computing Zagreb, Croatia, 2020.

[2] Network i Emerging Optimization. *The Vehicle Routing Problem*. URL `http://neo.lcc.uma.es/vrp/`.

[3] Massachusetts Institute of Technology Education. *Single-Depot VRP*. URL `http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html`.

[4] Sandeep Dhakal Raymond Chiong. *Natural Intelligence for Scheduling, Planning and Packing Problems*.

[5] Marin Golub Stjepan Picek, Domagoj Jakobovic. *On the Recombination Operator in the Real-Coded Genetic Algorithms*. Faculty of Electrical Engineering and Computing Zagreb, Croatia.

Optimizacija problema usmjeravanja vozila

Sažetak

Ovaj rad se bavi problemom usmjeravanja vozila i njegovim varijantama. Opis metoda i principa rješavanja problema usmjeravanja vozila. Heuristički algoritam ušteda, genetski algoritam i genetski operatori su ostvareni, isto kao i dva prikaza rješenja. Ocjena učinkovitosti ostvarenih metoda nad tri tipa instanci problema. Rezultati su prikazani grafički te su analizirani i međusobno uspoređeni.

Ključne riječi: problem usmjeravanja vozila, VRP, VRPTW, genetski algoritam, genetski operatori, algoritam ušteda

Optimisation of the vehicle routing problem

Abstract

This paper deals with vehicle routing problem and its flavors. Detailed descriptions of vehicle routing problem solving methods and principles were given. Heuristic savings algorithm, genetic algorithm and genetic operators were implemented, as well as two forms of results. Efficiency evaluation of selected methods and their performance on three types of vehicle routing problem. Graphic representations of results were created and results were analysed and mutually compared.

Keywords: vehicle routing problem, VRP, VRPTW, genetic algorithm, genetic operators, savings algorithm