SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 492

IZRADA HEURISTIKA ZA PROBLEM USMJERAVANJA VOZILA KORIŠTENJEM GENETIČKOG PROGRAMIRANJA

Marko Opačić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 492

IZRADA HEURISTIKA ZA PROBLEM USMJERAVANJA VOZILA KORIŠTENJEM GENETIČKOG PROGRAMIRANJA

Marko Opačić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 492

Pristupnik:	**Marko Opačić (0036523828)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Izrada heuristika za problem usmjeravanja vozila korištenjem genetičkog programiranja**

Opis zadatka:

Proučiti problem usmjeravanja vozila te različite varijante tog problema. Istražiti i implementirati jednostavne heuristike za rješavanje problema usmjeravanja vozila. Proučiti primjenu genetičkog programiranja za automatsku izrada jednostavnih heuristika za problem usmjeravanja vozila. Definirati karakteristike problema koje će genetičko programiranje koristiti u razvoju heuristika. Implementirati okvir unutar kojeg će biti moguće razvijati nove heuristike za problem usmjeravanja vozila. Usporediti razvijene heuristike s jednostavnim postojećim heuristikama za razmatrani problem. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 10. lipnja 2022.

Želio bih se zahvaliti mentoru Marku Đuraseviću na njegovom trudu i strpljenju. Uvijek je bio spreman i voljan razgovarati, dati savjet, pomoći i izaći u susret te sam mu na tome iznimno zahvalan.

SADRŽAJ

1. Uvod

Pružatelji usluga dostave svakodnevno se susreću s problemom određivanja ruta za svoja vozila. Kako se povećava učestalost dostavljanja robe i broj dostavnih vozila, tako nastaje i potreba za optimizacijom ruta kojima vozila putuju radi smanjenja troškova transporta i brže dostave. U novije vrijeme taj optimizacijski problem naziva se problem usmjeravanja vozila, te je od velike praktične važnosti. Problem je pronaći skup ruta za flotu vozila s minimalnom ukupnom duljinom ruta.

Kako je ovo kombinatorički težak problem za koji iscrpna pretraga za najboljim skupom ruta postaje neizvediva i uz relativno mali broj čvorova koje je potrebno posjetiti, obično se za njegovo rješavanje koriste heuristike. Korištenjem heuristika tako ne dobivamo nužno optimalno rješenje, već se nadamo da ćemo dobiti rješenje koje nam je dovoljno dobro ali koje je moguće izračunati uz smanjene zahtjeve računalnih resursa u usporedbi s metodama koje traže isključivo optimalno rješenje. Međutim, nailazimo na problem izrade heuristika. Ovdje posežemo za genetičkim programiranjem kao alatom koji generira, evaluira i unaprjeđuje heuristike.

Genetičko programiranje je evolucijski pristup rješavanju optimizacijskih problema. Zasniva se na održavanju i evoluiranju populacije programa u nadi da će svaka sljedeća generacija izroditi bolje jedinke. Programi kroz generacije mutiraju i međusobno se križaju ovisno o njihovoj učinkovitosti. Glavni i često jedini mjeritelj učinkovitosti programa je tzv. funkcija dobrote (eng. *fitness function*). Ovisno o tome kako smo postavili problem, funkciju dobrote želimo minimizirati ili maksimizirati. Kao evolucijski pristup, genetičko programiranje obično nema znanje o domeni problema koji se pokušava riješiti, te je stoga funkcija dobrote jedina naznaka sustavu genetičkog programiranja razvija li se neki program u dobrom smjeru.

Konkretno, kod problema usmjeravanja vozila program će ustvari predstavljati samu heuristiku koju razvijamo. Za pojedini problem želimo minimizirati ukupni zbroj duljina ruta svih vozila korištenih u rješenju. Taj zbroj će ovisiti o programu odnosno heuristici koju koristimo pri rješavanju. Vrijednost funkcije dobrote za pojedinu heuristiku će biti ukupni zbroj vrijednosti tih duljina za svaki problem iz zadanog

skupa problema kad na njih primijenimo zadanu heuristiku. Uz ovako zadanu funkciju dobrote, njenu vrijednost želimo minimizirati. Drugim riječima, želimo minimizirati ukupnu duljinu svih ruta za neki skup problema kad na sve probleme iz skupa primijenimo određenu heuristiku.

Za dobre rezultate dobivenih heuristika ključan je odabir karakteristika problema kojima će te heuristike baratati. Jednostavne heuristike poput najbližeg susjeda koriste samo udaljenost do najbližeg susjeda kao karakteristiku.

Cilj ovog rada je izrada heuristika za problem usmjeravanja vozila korištenjem genetičkog programiranja. U daljnjem tekstu najprije slijedi detaljniji uvod u problem usmjeravanja vozila. Zatim ćemo se kroz uvod u genetičko programiranje upoznati s nekim osnovnim pojmovima i konceptima. Nakon toga slijedi pregled primjene genetičkog programiranja na problem usmjeravanja vozila. Konačno, opisani su korišteni postupci, odabrane karakteristike problema usmjeravanja vozila i dobiveni rezultati.

2. Problem usmjeravanja vozila

2.1. Opis problema

Problem su prvi puta opisali Dantzig i Ramser (1959) kao generalizaciju problema trgovačkog putnika. U problemu trgovačkog putnika traži se najkraća ruta koja posjećuje svih n zadanih točaka. Problem usmjeravanja vozila proširuje taj problem uvođenjem dodatnih uvjeta, uz izmjenjenu terminologiju. Umjesto putnika imamo vozilo, no umjesto jednog vozila ima ih više. Sva vozila kreću iz iste početne točke i u njoj završavaju rutu. Cilj je pronaći skup ruta s minimalnom ukupnom duljinom.

2.2. Varijante problema usmjeravanja vozila

Postoji više inačica problema usmjeravanja vozila koje dodaju neke dodatne uvjete osnovnom problemu. Neke od inačica su navedene u nastavku. U ovom radu razmatra se kapacitivni problem usmjeravanja vozila (CVRP) s konačnim brojem dostupnih vozila jednakih kapaciteta. Problem usmjeravanja vozila u ostatku teksta će često biti označen kraticom VRP.

2.2.1. Capacitated VRP

Osnovni problem se zasniva na tome da skup vozila posjećuje skup čvorova. U stvarnom svijetu, vozila obično posjećuju čvorove radi obavljanja dostave. Čvorovi imaju određenu potražnju odnosno količinu čiju dostavu zahtjevaju, dok vozila koja obavljaju dostavu imaju ograničen kapacitet. Pri određivanju rute ta se ograničenja moraju uzeti u obzir. Ova inačica koja uzima u obzir kapacitet vozila i potražnju čvorova naziva se kapacitativni problem usmjeravanja vozila, engl. Capacitated Vehicle Routing Problem (CVRP).

Borcinova (2017) daje matematički opis kapacitativnog problema usmjeravanja vozila. Neka je $G = (V, H, c)$ potpuni usmjereni graf sa skupom čvorova $V =$

$0, 1, 2, ..., n$ i skupom bridova $H = (i, j) : i, j \in V, i \neq j$, gdje je 0 oznaka početnog čvora za flotu od p vozila kapaciteta Q. Ostalih n čvorova neka označavaju čvorove klijente. Svaki klijent $i \in V$ ima određenu potražnju $d_i \leq Q$. Nenegativan trošak c_{ij} je pridan svakom bridu $(i, j) \in H$. Matrica troškova je simetrična, tako da je $c_{ij} = c_{ji}$, za svaki $i, j \in V$, te vrijedi nejednakost trokuta $c_{ij} + c_{jk} \geq c_{ik}$, za sve $i, j, k \in V$. Minimalan broj vozila potreban za opskrbu klijenata je $\lceil \frac{\sum_{i=1}^{n} d_i}{Q} \rceil$.

U razmatranju ovog rada, trošak između čvorova jednak je njihovoj udaljenosti.

2.2.2. VRP with Pickup and Delivery

Umjesto polaska iz početnog čvora s punim kapacitetom kao u slučaju CVRP-a, vozila moraju najprije pokupiti dobra u određenim čvorovima i tek nakon toga obaviti dostavu klijentima.

2.2.3. VRP with Time Windows

U dosadašnjim inačicama nije razmatrana vremenska komponenta. U stvarnim slučajevima često je bitno obaviti dostavu unutar nekog zadanog vremenskog intervala. Stoga ova varijanta razmatra slučaj kad je potrebno obaviti dostavu čvorovima između najranijeg i najkasnijeg vremenskog trenutka, često uz određene penale ovisno o tome je li vozilo uranilo, zakasnilo i koliko.

2.2.4. VRP with Profits

Ova inačica uklanja zahtjev za posjećivanjem svih čvorova, te se fokusira na maksimiziranje profita dobivenog od dostava. Postoje razne podinačice ovog problema koje nećemo posebno razmatrati.

2.3. Postojeće heuristike

2.3.1. Najbliži susjed

Čest pristup razvoju heuristika za VRP je korištenje postojećih heuristika za TSP za rutu pojedinog vozila. Na tom se principu bazira i ova heuristika. U svakom trenutku se kao sljedeći čvor u ruti odabire najbliži susjed trenutnog čvora. Ovo je pohlepna heuristika jer ne uzima u obzir ukupnu duljinu puta već samo udaljenost do sljedećeg

najbližeg čvora. Kao takva, njena implementacija je jednostavna ali uglavnom ne daje zadovoljavajuće rezultate.

2.3.2. Nasumičan odabir

U svakom se koraku nasumično odabere jedan od neposjećenih čvorova kao sljedeći čvor. Kad bi posjećivanje odabranog sljedećeg čvora povećalo ukupnu potražnju na ruti toliko da bi ona prekoračila kapacitet vozila, vozilo se vraća u početni čvor. Iako je na prvi pogled ovo vrlo loše rješenje, ova heuristika ponekad daje i bolje rezultate od heuristike najbliži susjed.

2.3.3. Clarke-Wright heuristika

Vjerojatno najpoznatiju heuristiku za problem usmjeravanja vozila razvili su Clarke i Wright (1964). Osnovna ideja je vrlo jednostavna. Razmotrimo problem s početnim čvorom D i n čvorova klijenata. Rješavanje započinjemo s n vozila od kojih svako posjećuje samo jedan od n čvorova te se zatim vraća nazad u početni i time završava rutu. Ukupna duljina svih ruta je tada jednaka $2\sum_{i=1}^{n} d(D,i)$.

Ako sada odaberemo jedno vozilo i njime posjetimo dva čvora, i i j, u jednoj ruti, ukupna duljina svih ruta bit će smanjena za:

$$\begin{aligned} s(i,j) &= 2d(D,i) + 2d(D,j) - (d(D,i) + d(i,j) + d(D,j)) \\ &= d(D,i) + d(i,j) - d(i,j) \end{aligned}$$

Vrijednost $s(i,j)$ nazivamo uštedom dobivenom kombiniranjem čvorova i i j u jednu rutu. Što je veća ušteda, to je povoljnije kombinirati i i j u jednu rutu. Prilikom kombiniranja potrebno je paziti na uvjete problema. Primjerice, kombiniranje nije dozovljeno ako bi se premašio kapacitet vozila na toj ruti.

S godinama su razvijena brojna poboljšanja za Clarke-Wright heuristiku. Pichpibul i Kawtummachai (2012) su tako razvili heuristiku koja u 81% slučajeva daje optimalno rješenje.

3. Genetičko programiranje

3.1. Osnovni pojmovi

Prije nego što razmotrimo primjenu genetičkog programiranja na neki konkretan problem, najprije se moramo upoznati s osnovnim načelima i uvesti neke osnovne pojmove.

3.1.1. Program

Već je utemeljeno da se genetičko programiranje zasniva na evoluiranju populacije programa. Međutim, taj naziv može zavarati, jer to obično nisu programi koji se sastoje od linija koda i koji su izvedivi. Češće se barata sintaksnim stablima koja predstavljaju neki matematički izraz. Pokretanje odnosno evaluacija programa je tada ekvivalentna evaluaciji izraza.

U nastavku teksta pojam program će se koristiti izmjenjivo s pojmovima stablo, izraz i jedinka, ovisno o kontekstu.

3.1.2. Funkcije

Matematički izrazi se sastoje od funkcija, varijabli i konstanti. Stoga ćemo razlikovati dva tipa čvorova sintaksnih stabala, unutarnje čvorove i listove. Unutarnji čvorovi imaju djecu, te će oni predstavljati funkcije. Argumenti funkcije će biti djeca unutarnjeg čvora.

Funkcije su obično matematičke operacije poput zbrajanja, oduzimanja, množenja, dijeljenja, trigonometrijskih funkcija i sl. Da bi sustav genetičkog programiranja ispravno radio, na skup funkcija uvodimo zahtjev zatvorenosti. U osnovi, to radimo zbog načina na koji se primjenjuje operator križanja opisan u poglavlju 3.3.1, a posljedica je da svi argumenti i sve povratne vrijednosti funkcija moraju biti istog tipa. Dodatno, potrebno je pripaziti na rubne slučajeve i domene funkcija. Primjerice, moguće

je da se u sintaksnom stablu pojavi dijeljenje s nulom. Taj slučaj se obično razrješava vraćanjem vrijednosti dijeljenika ili vraćanjem neke zadane vrijednosti poput broja 1.

3.1.3. Terminali

Terminali su zajednički naziv za varijable i konstante. Varijable i konstante za razliku od funkcija ne primaju nikakve argumente, pa zbog toga u sintaksnom stablu neće imati djecu. Shodno tome, terminali su u sintaksnom stablu predstavljeni listovima.

Iako se skup terminala može sastojati samo od varijabli, primjenom funkcija mogu se generirati i određene konstante. Na primjer, dijeljenje neke varijable same sa sobom kao rezultat će dati broj 1, dok će oduzimanje varijable od same sebe dati broj 0. Tako se ranije navedeni rubni slučaj dijeljenja s nulom može pojaviti čak i ako nemamo konstante u skupu terminala.

3.1.4. Skup primitiva

Unijom skupa funkcija i skupa terminala dobivamo tzv. skup primitiva. Svi čvorovi sintaksnog stabla su primitivi.

3.1.5. Populacija programa

Genetičko programiranje evoluira populaciju programa, i to generaciju po generaciju stohastički transformirajući jednu populaciju programa u novu populaciju u nadi da će nova generacija sadržavati bolje programe.

3.1.6. Funkcija dobrote

Kako je cilj evolucije programa kreiranje boljih jedinki prenošenjem dijelova programa koji su dobro prilagođeni u sljedeću generaciju, potrebna nam je mjera prilagođenosti programa. Tu mjeru dobivamo računanjem vrijednosti funkcije dobrote, te ju zovemo *fitness*. Funkcija dobrote jedini je mehanizam kojime sustavu genetičkog programiranja zadajemo cilj, odnosno određujemo što programom želimo postići. Funkcija dobrote može se mjeriti na više načina. Primjerice, funkcija dobrote može računati grešku između rezultata izvođenja programa i tražene vrijednosti. Na sličan način može se računati vrijeme, preciznost ili profit. Ako želimo minimizirati vrijednost funkcije dobrote, za jedinku s nižom vrijednošću funkcije dobrote reći ćemo da ima

bolji *fitness*. Jednako tako ćemo za jedinku s višom vrijednosti funkcije dobrote reći da ima loš *fitness*, odnosno da je lošije prilagođena.

3.1.7. Genetske operacije

Već je utemeljeno da se genetsko programiranje, kao i drugi evolucijski algoritmi, zasniva na načelu evoluiranja populacije. Evoluiranje populacije zapravo znači generiranje novih jedinki iz postojećih. Nove jedinke dobivamo primjenom određenih genetskih operatora nad postojećim jedinkama.

Dvije su glavne genetske operacije:

- **Križanje** - kreiranje nove jedinke kombinacijom nasumično odabranih dijelova iz dvije roditeljske jedinke.
- **Mutacija** - kreiranje nove jedinke nasumičnom promjenom dijela jedne roditeljske jedinke.

Obje operacije su detaljnije opisane u poglavlju 3.3

3.2. Inicijalizacija populacije

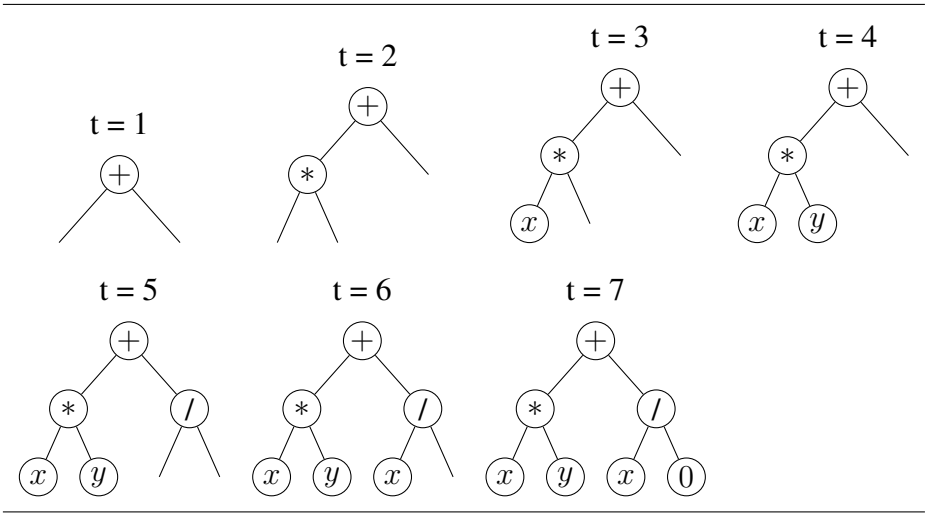
Da bismo mogli evoluirati neku populaciju, najprije ju moramo nasumično generirati. Dvije najčešće korištene metode generiranja nasumičnih stabala su metode `full` i `grow`. Obje su opisane u nastavku.

3.2.1. Full metoda

Stabla se generiraju tako da ne premašuju zadanu maksimalnu dubinu d. Čvorovi se nasumično odabiru iz skupa funkcija dokle god je trenutna dubina manja od maksimalne. Kad se dosegne maksimalna dubina čvorovi se nasumično odabiru iz skupa terminala. Generiranje stabla korištenjem metode `full` prikazano je na slici 3.1 Iako se na prvu može činiti da će sva stabla uz zadanu maksimalnu dubinu biti istog oblika, to je istinito samo u slučaju kad sve funkcije imaju jednak broj argumenata.

Za primjere su korišteni sljedeći skupovi:

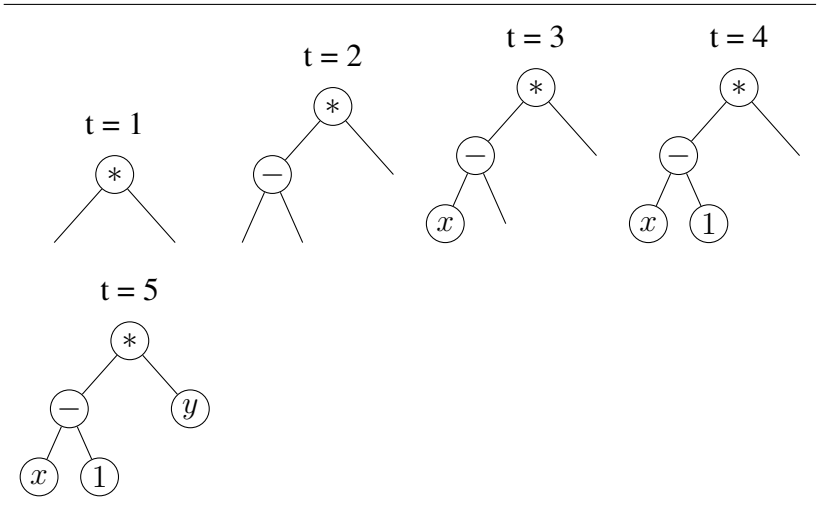
- **funkcije**: $\{+, -, *, /\}$
- **terminali**: $\{x, y, 0, 1\}$



Slika 3.1: generiranje stabla metodom full uz $d = 2$

3.2.2. Grow metoda

Kao i kod `full` metode, stabla su ograničena maksimalnom dubinom. No, za razliku od metode `full`, u svakom trenutku biramo čvorove iz cijelog skupa primitiva, dakle uključujući i funkcije i terminale, dokle god je trenutna dubina manja od maksimalne. Na maksimalnoj dubini se za čvor opet mogu birati samo terminali. Jasno je da metoda `grow` opisanim postupkom rezultira generiranjem stabala raznovrsnijih oblika od metode `full`. Na slici 3.2 prikazan je primjer generiranja stabla korištenjem metode `grow` uz maksimalnu dubinu $d = 3$. Valja primijetiti da dubina generiranog stabla može biti manja od maksimalne dubine, što nikad nije slučaj kod metode `full`.



Slika 3.2: generiranje stabla metodom grow uz $d = 3$

3.2.3. Ramped half-and-half

Pregledom gore navedenih metoda i danih primjeraka može se primijetiti da ni jedna ni druga same po sebi nisu posve prikladne za generiranje raznovrsne populacije. Metoda `full` će za skup funkcija koje imaju ujednačen broj argumenata dati stabla sličnih oblika. Dapače, ako sve funkcije imaju jednak broj argumenata, sva će generirana stabla uz danu dubinu biti istog oblika. S druge strane, `grow` metoda će rezultirati stablima različitih oblika, ali ne pruža nikakvu kontrolu nad očekivanom dubinom stabla.

Stoga je Koza (1992) predložio da se inicijalizacija populacije odrađuje tako da se 50% populacije generira metodom `full`, a 50% metodom `grow`. Ova metoda se naziva *ramped half-and-half*.

Vrijedi napomenuti da početna populacija ne mora biti potpuno slučajna. Ako znamo nešto o vjerojatnim svojstvima željenog rješenja, stabla s tim svojstvima se mogu iskoristiti za generiranje početne populacije.

3.3. Operatori križanja i mutacije

3.3.1. Operator križanja

Operator križanja je u genetičkom programiranju najčešće implementiran kao križanje podstabala. Uz dva dana roditelja, u svakom od roditelja odaberemo točku križanja, odnosno određeni čvor. Podstabla su tada stabla kojima su korijeni odabrane točke križanja. Križanje tada obavljamo zamjenom dvaju podstabala.

3.3.2. Operator mutacije

Najčešći oblik mutacije korišten u genetičkom programiranju je mutacija podstabla. Kao i kod križanja, nasumično se odabire jedan čvor stabla kao točka mutacije. Podstablo određeno tim čvorom se zatim mijenja nasumičnim podstablom. Ovakva mutacija može se implementirati kao križanje s nasumično generiranim stablom.

Još jedan čest oblik mutacije je mutacija točke. Umjesto zamjene čitavog podstabla mijenjamo samo jedan čvor stabla. Jedino je potrebno voditi računa o tome koji tip čvora mijenjamo, tako da zamjenski čvor bude istog tipa. Primjerice, kod mutacije se ne smije dogoditi da funkciju zamijenimo terminalom.

3.3.3. Odabir operatora i reprodukcijski operator

Operatori su u genetskom programiranju obično međusobno isključivi, što znači da nije moguća kompozicija operatora. Vjerojatnosti primjene operatora se poprilično razlikuju. Prema O'Neill (2009), obično se operator križanja najčešće primjenjuje, s vjerojatnošću od 90% i više. S druge strane, vjerojatnost primjene operatora mutacije je znatno manja, oko 1%. Kada je zbroj vjerojatnosti primjene operatora križanja i mutacije manji od 100%, dodatno se koristi tzv. operator reprodukcije. Ovaj operator jednostavno kopira program u sljedeću generaciju, odnosno kreira potomka koji je jednak roditelju. Ako je zbroj vjerojatnosti križanja i mutacije jednak p, tada je vjerojatnost reprodukcije jednaka $1 - p$.

3.4. Selekcija

Genetički operatori primjenjuju se na jedinke ovisno o njihovom *fitnessu*. Bolje jedinke će imati veću vjerojatnost stvaranja potomaka od lošije prilagođenih jedinki. Potrebno je odrediti postupak kojim će se odabirati programi iz kojih ćemo generirati potomke primjenom genetičkih operatora. Taj postupak zvat ćemo metoda selekcije. Postoje razne metode selekcije, ali u genetičkom programiranju najčešće se koristi turnirska selekcija.

3.4.1. Turnirska selekcija

Kao što je ranije navedeno, glavni kriterij za selekciju će biti *fitness* programa. U turnirskoj selekciji najprije odaberemo nasumičan skup programa iz populacije. Odabrane programe zatim međusobno uspoređujemo prema fitnesu te najbolji program biramo za roditelja. Broj potrebnih turnira ovisi o operatoru koji primjenjujemo. Za operator križanja potrebna su dva roditelja, pa će se odraditi dva turnira. Za operator mutacije potreban je jedan turnir.

3.4.2. Troturnirska selekcija

Postoji varijanta turnirske selekcije u kojoj se uvijek odabiru tri jedinke. Dvije jedinke s najvećim *fitnessom* odabiru se za križanje i njihov potomak odnosno rezultat križanja dodaje se u populaciju. Treća jedinka iz odabira, s najgorim *fitnessom* od odabranih, uklanja se iz populacije. S ovakvim načinom selekcije više ne razlikujemo generacije, već baratamo iteracijama. U svakoj iteraciji događa se jedna troturnirska selekcija i

posljedično križanje. Ako je veličina populacije n, tada populaciju nakon svakih n iteracija možemo smatrati novom generacijom.

Dodatno, ako je u sustavu potrebno primijeniti mutacije, one se uz određenu vjerojatnost primjenjuju na jedinku dobivenu križanjem.

3.5. Parametri

Za samo pokretanje sustava genetičkog programiranje bitno je odrediti određene kontrolne parametre. Najvažniji od njih je veličina populacije. Općenito pravilo koje daju O'Neill (2009) je da veličina populacije bude što veća, a tipično 500. Ostali parametri uključuju vjerojatnosti primjene genetskih operacija i maksimalnu veličinu programa odnosno dubinu stabala.

Optimalne vrijednosti parametara značajno ovise od jedne primjene do druge. Ipak, iako su ovi parametri vrlo bitni, genetičko programiranje je u praksi robusno i uglavnom će dobro raditi uz mnoštvo različitih vrijednosti parametara.

3.6. Kriterij zaustavljanja

Iako bismo teoretski mogli pustiti sustav da generira nove generacije unedogled, voljeli bismo dobiti neke rezultate. Stoga nam je potreban nekakav kriterij po kojem ćemo odlučiti kada zaustaviti GP sustav. U trenutku zaustavljanja kao rezultat se vraća najbolja dosadašnja jedinka.

Obično se kao kriterij zaustavljanja koristi broj generacija. Tipično se preporuča da maksimalan broj generacija bude u rasponu od 10 do 50 generacija. Prema O'Neill (2009), ako se dovoljno dobra jedinka ne pronađe u tim ranim generacijama šanse da će se kasnije pojaviti bolje su poprilično male. Ovaj savjet dakako ovisi i o drugim faktorima poput veličine populacije i maksimalne dubine stabala.

Ako sustav nema striktno odvojene generacije, odnosno evolucija se odvija u iteracijama, tada se kao kriterij zaustavljanja može koristiti i broj iteracija. Ako je n_p veličina populacije, a g broj generacija, broj iteracija $i = n_p * g$ kao kriterij zaustavljanja usporediv je s kriterijem zaustavljanja od g generacija.

4. Primjena genetičkog programiranja na problem usmjeravanja vozila

4.1. Reprezentacija rješenja

Rješenje jednog pokretanja sustava genetičkog programiranja bit će najbolja jedinka iz svih populacija generiranih do trenutka zaustavljanja. To rješenje je izraz prikazan kao sintaksno stablo, uz koje dodatno vraćamo vrijednost funkcije dobrote.

4.2. Karakteristike problema

Bilo koja metoda rješavanja problema usmjeravanja vozila, pa tako i heuristike, mora baratati nekim informacijama o problemu. Primjerice, heuristika najbliži susjed mora imati informacije o udaljenostima između čvorova. Osim udaljenosti, postoji mnoštvo informacija koje bi nas mogle zanimati pri rješavanju konkretnog problema. Ove informacije nazivat ćemo karakteristikama. U sintaksnom stablu, karakteristike će ustvari biti varijable.

Kako je problem usmjeravanja vozila generalizacija problema trgovačkog putnika (TSP), karakteristike osmišljene za TSP su relevantne i za VRP. Rasku et al. (2016) dijele karakteristike problema usmjeravanja vozila na sedam kategorija:

1. Karakteristike distribucije čvorova (ND)
2. Karakteristike minimalnog razapinjućeg stabla (MST)
3. Local Search Probing karakteristike (LSP)
4. Branch-and-cut probing features (BCP)
5. Geometrijske karakteristike (G)
6. Karakteristike najbližeg susjedstva (NN)

7. Karakteristike specifične za VRP (DC)

U ovom radu detaljnije ćemo razmotriti neke od karakteristika iz kategorija distribucije čvorova, najbližeg susjedstva te karakteristika specifičnih za VRP.

4.2.1. Karakteristike distribucije čvorova

Centroid čvorova

Centroid je točka koja opisuje aritmetičku sredinu svih točaka iz nekog skupa točaka. koordinate x_c i y_c centroida skupa točaka $T_i(x_i, y_i)$ dane su izrazima:

$$x_c = \sum_{i=1}^{n} x_i$$
$$y_c = \sum_{i=1}^{n} y_i$$

Centroid tada označavamo kao $C(x_c, y_c)$.

Udaljenost od centroida

Za potencijalni sljedeći čvor zanima nas koja je njegova udaljenost od centroida. Ako je ta udaljenost velika, to znači da je čvor udaljen od ostalih čvorova u grafu. U tom slučaju, možda bi bilo povoljnije odabrati čvor koji je bliži centroidu. Ipak, ovo su samo nagađanja te ćemo zaključke o tome kakve su udaljenosti od centroida podobne prepuštamo GP sustavu.

4.2.2. Karakteristike najbližeg susjedstva

Udaljenost do prvog najbližeg susjeda

Ovo je jedina karakteristika koju koristi heuristika najbliži susjed, uz potražnju klijenata u CVRP-u da se ne bi prekoračio kapacitet vozila.

4.2.3. Karakteristike specifične za VRP

Udaljenost do početnog čvora

Označava koliko je čvor udaljen od početnog.

Preostali kapacitet vozila na ruti

Ukupna potražnja čvorova na ruti ne smije premašiti kapacitet. Kod malog preostalog kapaciteta teoretski je poželjno ne udaljavati vozilo previše od početnog čvora, pošto će se u njega uskoro trebati vratiti.

4.3. Odabir skupova funkcija i terminala

Prvi koraci u definiranju sustava genetičkog programiranja za rješavanje nekog problema su definiranje skupova funkcija i skupova terminala.

4.3.1. Skup funkcija

Sve funkcije iz odabranog skupa funkcija imaju dva argumenta. Odabrane funkcije su navedene u sljedećoj tablici:

Naziv funkcije	Argumenti	Povratna vrijednost
ADD	a, b	$a + b$
SUB	a, b	$a - b$
MUL	a, b	$a * b$
DIV	a, b	a/b
MAX	a, b	a ako je $a > b$, inače b
MIN	a, b	a ako je $a < b$, inače b

Tablica 4.1: skup funkcija

4.3.2. Skup terminala

Odabrani skup terminala ne sadrži konstante, već samo varijable, odnosno karakteristike.

Kratki naziv karakteristike	Naziv karakteristike
dist_to_node	Udaljenost od sljedećeg čvora
dist_to_centroid	Udaljenost od centroida
dist_to_depot	Udaljenost od početnog čvora
remaining_capacity	Preostali kapacitet vozila na ruti

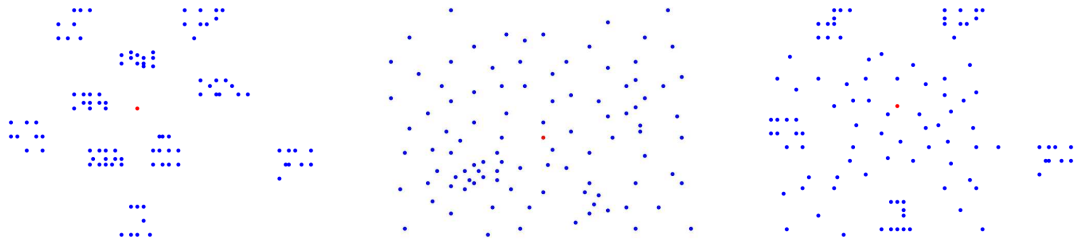
Tablica 4.2: skup terminala

5. Rezultati

5.1. Skup podataka

U treniranju i testiranju sustava korišteni su problemski skupovi Solomon. Solomon instance su podijeljene u 3 skupine:

- *Clustered*, oznaka C
- *Random*, oznaka R
- *Random/Clustered*, oznaka RC

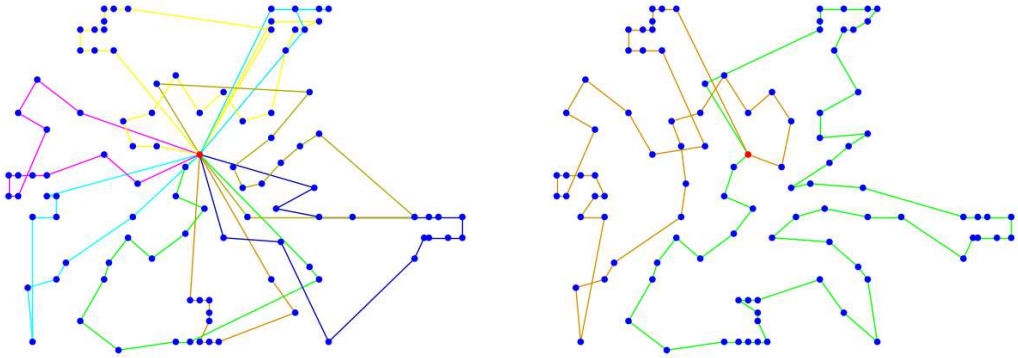


Slika 5.1: Usporedba instanci različitih skupina. Slijeva nadesno, primjeri C, R i RC skupine

Dodatno, svaka od skupina je podijeljena u dvije podskupine, 1 i 2. C skupina podijeljena je u C1 i C2, R skupina u R1 i R2, te RC skupina u RC1 i RC2. Podskupine 1 općenito imaju manje kapacitete vozila, pa je za njihovo rješavanje potrebno iskoristiti više vozila nego za podskupine 2, što se može vidjeti na slici 5.2.

5.2. Korišteni parametri

Nakon nekoliko pokretanja i testiranja, u svim relevantnim pokretanjima korišteni su parametri navedeni u tablici 5.2. Korijenski čvor je na dubini 1, dok se mutacija obavlja s danom vjerojatnošću u svakoj iteraciji nad jedinkom dobivenom križanjem. Nakon zadanog maksimalnog broja iteracija izvršavanje se zaustavlja.



Slika 5.2: Usporedba rješenja na instancama podskupine RC1 i RC2.

Parametar	Vrijednost
Veličina populacije	200
Maksimalna dubina stabla	6
Maksimalni broj iteracija	5000
Vjerojatnost mutacije	2%

Tablica 5.1: korišteni parametri

5.3. Referentne vrijednosti

Postoje razne metode i rješavači problema usmjeravanja vozila, te brojni numerički rezultati konkretno za Solomon instance. Na te vrijednosti možemo jedino gledati kao na donju granicu onog što je moguće, pošto usporedba s heuristikama i korištenom metodologijom nije poštena ni relevantna. Za ovaj rad je najrelevantija heuristika najbliži susjed jer koristi istu metodologiju praćenja karakteristika, iako ona prati samo jednu. U razmatranju dobivenih rezultata najviše ćemo se fokusirati na usporedbu upravo s tom heuristikom.

5.4. Dobivena rješenja

U početnim pokretanjima uz samo tri karakteristike, udaljenost do sljedećeg čvora, udaljenost sljedećeg čvora do centroida i udaljenost sljedećeg čvora od početnog čvora, dizajnirani sustav nije dao zadovoljavajuće rezultate. Generalno nije pronađena heuristika bolja od heuristike najbliži susjed, prema kojoj su u konačnici konvergirali svi programi iz populacije. Isprva se činilo da sustavu informacije o samo tim trima uda-

ljenostima nisu bile dovoljne da razvije značajnije heuristike. Međutim, promjenom parametara a ponajprije povećanjem populacije, te manjim izmjenama u sustavu, uspješno su dobivene jedinke koje daju kvalitetna rješenja. Uz dodavanje karakteristike preostalog kapaciteta vozila na ruti kvaliteta rješenja je dodatno poboljšana.

Označimo kapacitet vozila sa c, ukupni zbroj potražnji svih klijenata na ruti s d_{total}, a vrijednost karakteristike preostalog kapaciteta vozila na ruti s $c_{remaining}$. Tada vrijedi: $c_{remaining} = c - d_{total}$

5.4.1. Ponašanje rješenja na skupu za validaciju

U nastavku je prikazano ponašanje najboljih jedinki iz populacije na skupu za treniranje i skupu za validaciju. Skupovi za treniranje, validaciju i testiranje su u relevantnim pokretanjima sadržavali samo instance iz R i RC skupina. Tri su osnovne podjele instanci među korištenim skupovima za treniranje i validaciju:

1. Skup za treniranje sastoji se od instanci iz skupina R1 i RC1, skupovi za validaciju i treniranje od R2 i RC2, slika 5.3

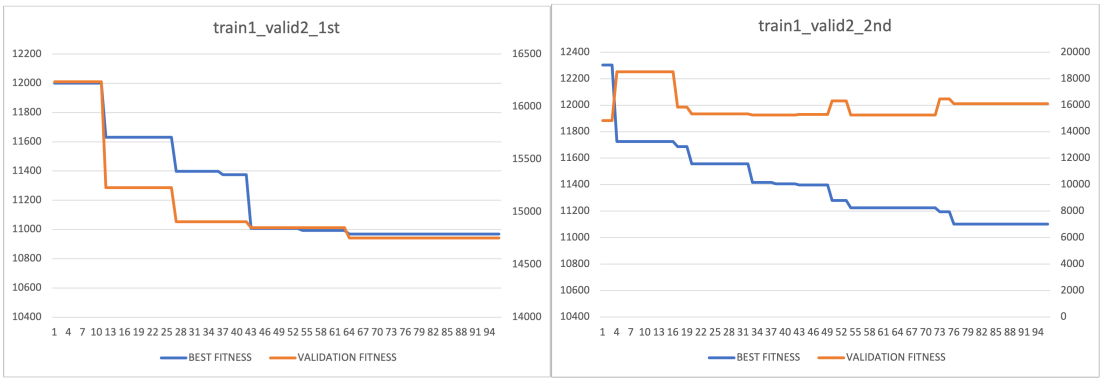
2. Skup za treniranje sastoji se od instanci iz skupina R2 i RC2, skupovi za validaciju i treniranje od R1 i RC1, slika 5.4

3. Skup za treniranje sastoji se od instanci iz skupina R1 i RC2, skupovi za validaciju i treniranje od R2 i RC1, slika 5.5

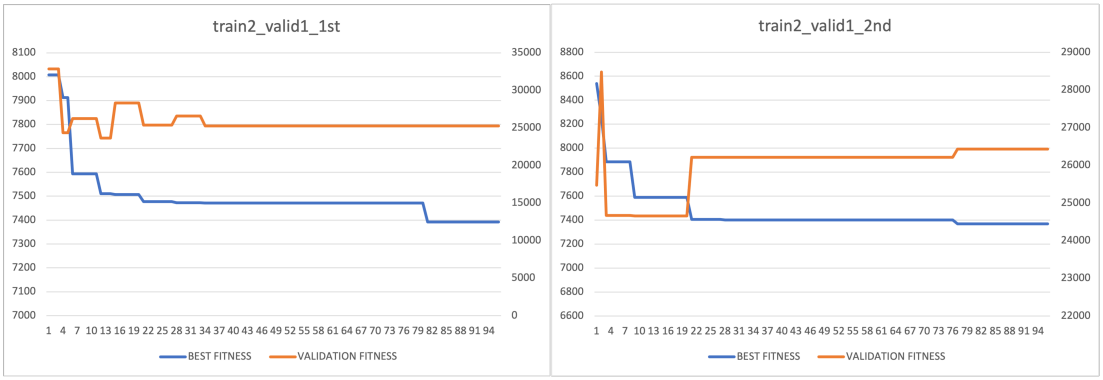
Za svaku od podjela dana su dva primjera izvršavanja. Iz grafova je vidljivo je da se vrlo često javlja problem prenaučenosti. Naime, vrijednost funkcije dobrote na skupu za validaciju raste prije izvršavanja svih iteracija. Ovaj problem može se kratkoročno zaobići tako da se uzme rješenje s najboljom vrijednošću funkcije dobrote na skupu za validaciju tijekom cijelog jednog izvršavanja, umjesto najboljeg rješenja na skupu za treniranje. Postoje razne metode za sprečavanje prenaučenosti, od kojih su neke navedenu u poglavlju 5.5.2.

Za 1. podjelu smo u jednom primjeru dobili poprilično zadovoljavajuću krivulju, no već za drugi primjer s istom podjelom najbolje rješenje na skupu za validaciju je ono najbolje iz početne populacije. Stoga ne možemo donijeti neke značajne zaključke o primjerenosti i kvaliteti raspodjele skupova. Da bismo mogli donijeti takve zaključke, potrebno je izvršavanje pokrenuti mnogo više od dva puta, pošto će zbog stohastične prirode genetičkog programiranja svako pokretanje dati drugačije rezultate.

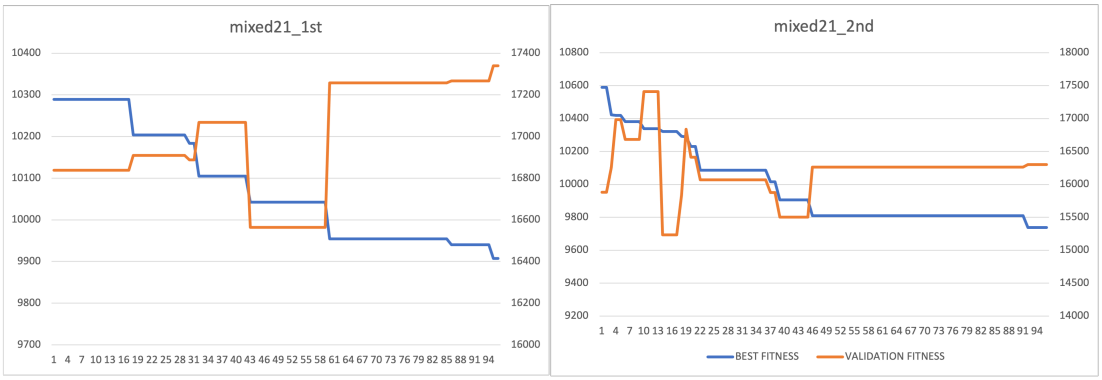
Čak i uz problem prenaučenosti, neke od dobivenih heuristika su davale kvalitetna rješenja, što je opisano kasnije u poglavlju 5.4.2.



Slika 5.3: ponašanje najbolje jedinke na skupu za validaciju za 1. podjelu



Slika 5.4: ponašanje najbolje jedinke na skupu za validaciju za 2. podjelu



Slika 5.5: ponašanje najbolje jedinke na skupu za validaciju za 2. podjelu

5.4.2. Usporedba dobivenih rješenja s heuristikom najbliži susjed

Zbog pojave prenaučenosti, u sljedećim podacima uzete su jedinke s najboljim validacijskim *fitnessom* za određenu podjelu. Uspoređene su vrijednosti funkcije dobrote na testnom skupu najboljih jedinki dobivenih za određenu podjelu s vrijednostima funkcije dobrote za heuristiku najbliži susjed na istom testnom skupu. Poboljšanja su generalno u rasponu 8-16%. Najbolji rezultati postignuti su za 2. podjelu, gdje je poboljšanje u odnosu na heuristiku najbliži susjed 15.52%.

Podjela	**dobiveno rješenje**	**najbliži susjed**	**poboljšanje**
1.	2617.689	2984.446	12.29%
2.	6581.458	7790.891	15.52%
3.	6134.065	6675.992	8.12%

Tablica 5.2: korišteni parametri

5.4.3. Istaknuto rješenje za C skupinu

U ranim fazama testiranja sustava nisu dobiveni zadovoljavajući rezultati na C skupini problema, te se od eksperimentiranja na toj skupini brzo odustalo. U kasnijoj fazi je pri treniranju sustava na RC skupini problema dobiveno jedno zanimljivo rješenje, za koje se pokazalo da daje relativno kvalitetna rješenja za C skupinu. Pošto je izraz vrlo jednostavan, smatram da ga ovdje vrijedi istaknuti i razmotriti. Rješenje je nazvano `simple_c_rc`, te je tako navedeno i u tablici 5.3. Izraz je sljedeći, nakon sređivanja:

$$d_{node} + d_{centroid} - d_{depot}$$

Udaljenost do sljedećeg čvora označena je s d_{node}, udaljenost do centroida s $d_{centroid}$, te udaljenost do početnog čvora s d_{depot}. Vidljivo je da će ovaj izraz biti što manji za što manje vrijednosti d_{node} i $d_{centroid}$, a što veći d_{depot}. Drugim rječima, bit će minimalan ako je sljedeći čvor blizak trenutnom čvoru i centroidu, a daleko od početnog čvora. Upravo takav slučaj je uobičajen kod klastera, s napomenom da pozicija centroida ima korisniju informaciju kada je velik udio preostalih čvorova u jednom klasteru. Ovo nas navodi na pomisao da bi se kao moguće dodatno poboljšanje za C skupinu moglo uvesti karakteristiku centroida obližnjih m čvorova, no to nije ostvareno u ovom radu.

U usporedbi s heuristikom najbliži susjed, ova heuristika u nekim slučajevima daje čak i do 17% bolje rješenje. Ipak, za nasumičnu prostornu raspodjelu čvorova odnosno R skupinu, ova heuristika daje loše rezultate, čak ponekad lošije od heuristike najbliži susjed. Ovi podaci vidljivi su u tablici 5.3.

Skupina		**`simple_c_rc`**	**najbliži susjed**
C	C1	9837.236	11803.499
	C2	6071.248	6203.895
R	R1	14596.147	14092.334
	R2	8781.436	8830.009
RC	RC1	10076.068	11380.821
	RC2	6232.705	7240.338

Tablica 5.3: korišteni parametri

5.5. Moguća poboljšanja

5.5.1. Ubrzanje izvršavanja

Glavna zapreka unaprjeđenju sustava bilo je trajanje jednog pokretanja sustava, koje je negativno utjecalo na mogućnost eksperimentiranja, kako s drugačijim parametrima tako i s uvođenjem dodatnih karakteristika. Ovaj problem bi se mogao riješiti na dva načina. Prvi je optimizacija implementacije. Optimizaciji je pridana velika pozornost, te ne vidim moguća očita ubrzanja, osim mogućnosti implementacije u nekom bržem programskom jeziku. Drugi način ubrzanja provedbe jednog pokretanja sustava je jednostavno korištenje boljeg i bržeg hardvera, što bi ubrzalo rad sustava neovisno o algoritmu.

5.5.2. Izbjegavanje prenaučenosti

Drugi problem koji se konstantno pojavljivao je prenaučenost. Ovisno o veličini skupa za treniranje i veličini populacije, prije završetka jednog treniranja dogodilo bi se da rješenja sve lošije i lošije generaliziraju na neviđene podatke. U nekim slučajevima bi po završetku treniranja rješenja bila gora od heuristike najbliži susjed. Postoje razne metode sprječavanja prenaučenosti, ali preferirana metoda u genetičkom programiranje je primjena određenih penala za veliku kompleksnost jedinki. Ova metoda naziva

se parsimonijski pritisak (*eng. parsimony pressure*), te se pokazala uspješnom u brojnim radovima, poput Soule i Foster (1998) i Cavaretta i Chellapilla (1999).

5.5.3. Dodatne karakteristike

Sustav u trenutnom stanju barata sa samo četiri karakteristike problema. Kako i sa te četiri vidimo napredak u odnosu na samo jednu, odnosno na heuristiku najbliži susjed, možemo očekivati da će dodavanje nekih dodatnih karakteristika jednako tako pozitivno utjecati na kvalitetu rješenja. Sporo izvršavanje je nažalost spriječilo eksperimentiranje s nekim drugim karakteristikama.

5.5.4. Ostale inačice problema

Trenutno sustav može izrađivati heuristike samo za CVRP inačicu, no uz izmjene to se može proširiti i na primjerice VRPTW. Iste Solomon instance se mogu primjenjivati i na VRPTW.

6. Zaključak

Problem usmjeravanja vozila jedan je od najproučavanijih problema u računarskoj znanosti. Postoje razne inačice ovog problema, no u ovom se radu razmatrala samo inačica kapacitativnog problema usmjeravanja vozila (*eng. CVRP*). Razmotrene su neke od heuristika za rješavanje problema usmjeravanja vozila, od kojih je jedna i heuristika najbliži susjed. Ona pri rješavanju prati samo jednu karakteristiku problema, a to je udaljenost do sljedećeg čvora. Ideja za unaprjeđenje te heuristike bila je izrada heuristika koje će pratiti i neke druge karakteristike problema, poput centroida čvorova i preostalog kapaciteta.

Proces izrade heuristika željelo se automatizirati, te je u tu svrhu implementiran sustav genetičkog programiranja koji će automatski generirati, evaluirati i unaprjeđivati nove heuristike. Pri izradi heuristika implementiranom sustavu genetičkog programiranja na raspolaganju su bile četiri karakteristike, udaljenost do sljedećeg čvora, udaljenost do centroida svih čvorova, udaljenost do početnog čvora te preostali kapacitet vozila na ruti. Navedenom metodom uspješno su razvijene bolje heuristike od heuristike najbliži susjed. Poboljšanja su bila u rasponu 8-17%, ovisno o korištenim skupovima problema. Ipak, dobivene heuristike kvalitetom rješenja ne mogu se mjeriti sa naprednijim metodama i algoritmima. Pošto je treniranje sustava genetičkog programiranja stohastički proces, postoji mogućnost da se i bez promjene parametara i sustava u naknadnim pokretanjima dobiju bolje heuristike.

Jedan od problema koji su se pojavljivali pri pokretanju sustava genetičkog programiranja je sporo izvršavanje. Velika pažnja je predana implementaciji raznih optimizacija, no i s njima je izvršavanje i dalje bilo poprilično sporo. Glavna zapreka ubrzanju bila je evaluacija funkcije dobrote, što je čest slučaj u genetičkom programiranju. To je donekle poremetilo proces eksperimentiranja s raznim karakteristikama i parametrima sustava. Drugi problem je bio problem prenaučenosti, koji se javljao u većini pokretanja.

Moguća poboljšanja sustava su ubrzavanje izvršavanja, korištenje skupa za validaciju da bi se odredili optimalni parametri sustava, primjena metoda za sprečavanje

prenaučenosti, te dodavanje novih karakteristika. Za sprječavanje prenaučenosti i ograničavanje kompleksnosti jedinki moguća je primjena metode parsimonijskog pritiska, koja se pokazala vrlo uspješnom u brojnim radovima.

LITERATURA

Zuzana Borcinova. Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review*, stranice 463–469, 2017.

Michael J Cavaretta i Kumar Chellapilla. Data mining using genetic programming: The implications of parsimony on generalization error. U *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, svezak 2, stranice 1330–1337. IEEE, 1999.

Geoff Clarke i John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

George B Dantzig i John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

JRGP Koza. On the programming of computers by means of natural selection. *Genetic programming*, 1992.

Michael O'Neill. Riccardo poli, william b. langdon, nicholas f. mcphee: a field guide to genetic programming, 2009.

Tantikorn Pichpibul i Ruengsak Kawtummachai. An improved clarke and wright savings algorithm for the capacitated vehicle routing problem. *ScienceAsia*, 38(3): 307–318, 2012.

Jussi Rasku, Tommi Kärkkäinen, i Nysret Musliu. Feature extractors for describing vehicle routing problem instances. *OASICS; 50*, 2016.

Terence Soule i James A Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary computation*, 6(4):293–309, 1998.

Izrada heuristika za problem usmjeravanja vozila korištenjem genetičkog programiranja

Sažetak

Opisan je problem usmjeravanja vozila i njegove inačice. Definiran je skup karakteristika problema koje genetičko programiranje koristi pri razvoju novih heuristika. Implementiran je sustav genetičkog programiranja koji omogućava definiranje novih karakteristika, podešavanje parametara sustava te razvoj novih heuristika. Rezultatima rada pokušalo se pronaći heuristike koje su kvalitetom rješenja usporedive s postojećim heuristikama.

Ključne riječi: Problem usmjeravanja vozila, genetičko programiranje

Design of heuristics using genetic programming for the vehicle routing problem

Abstract

The vehicle routing problem and its variants were described and presented. A set of features was defined to be used by genetic programming in the development of new heuristics. A genetic programming system was implemented with which it is possible to define new features, modify system parameters and develop new heuristics. Through the results of this work, an attempt was made to find heuristics whose solutions are comparable in quality to those of existing heuristics.

Keywords: Vehicle routing problem, genetic programming