SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1970

Uporaba simboličke regresije za rješavanje problema usmjeravanja vozila

Luka Kraljević

Zagreb, lipanj 2019.

Zahvala mentoru prof. dr. sc. Domagoju Jakobovi\'cu na nesebičnoj pomoći, stalnoj suradnji i strpljenju pri izradi ovog rada.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2019.

DIPLOMSKI ZADATAK br. 1970

Pristupnik: **Luka Kraljević (0036484370)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Uporaba simboličke regresije za rješavanje problema usmjeravanja vozila**

Opis zadatka:

Proučiti inačice problema usmjeravanja vozila i postojeće načine rješavanja s obzirom na zadana ograničenja. Posebnu pažnju posvetiti rješavanju problema u dinamičkim uvjetima za proizvoljni optimizacijski kriterij. Definirati hiperheuristički model usmjeravanja vozila koji pronalazi heuristiku odabira sljedećeg korisnika temeljem dinamičkog prioriteta. Ostvariti prilagodljivo okruženje za simulaciju dinamičkog rješavanja problema usmjeravanja vozila uz ograničenja vremena i kapaciteta, kao i mogućnosti dostave i preuzimanja tereta. Primijeniti tehnike simboličke regresije u cilju pronalaženja učinkovitih prioritetnih funkcija za raspoređivanje vozila. Usporediti učinkovitost ostvarenih postupaka s postojećim rješenjima. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:
Doc. dr. sc. Marko Čupić

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

Problem usmjeravanja vozila (Vehicle Routing Problem) je važan kombinatorni optimizacijski problem koji ima širok raspon primjene iz stvarnog svijeta u lancu opskrbe i logistici. Zadatak takvog problema je generiranje skupa ruta za svako vozilo kako bi se poslužili zadani zahtjevi klijenata na različitim mjestima, vozila obično kreću iz jednog mjesta, odnosno skladišta, te se po završetku rute moraju vratiti u isto to skladište. Konačan cilj je konstruirati rute na način da ukupna prijeđena udaljenost bude što manja, kao i broj korištenih vozila.

Do sada su se mnoga istraživanja bavila tim problemom, ali i brojnim varijantama tog problema. CVRP je problem u kojem flota vozila jednakog kapaciteta poslužuje zahtjeve klijenata, dakle, kapacitet vozila je ograničenje koje mora biti zadovoljeno u svakoj ruti vozila. VRPB (VRP with Backhauls) je proširenje VRP-a u kojem se roba može i pokupiti, a ne samo dostaviti. No, sve akcije prikupljanja moraju se desiti tek nakon svih dostava predviđenih na određenoj ruti vozila. VRPPD (VRP with Pick-up and Delivering) je sličan VRPB-u, samo što ovdje nema ograničenja za poziciju lokacija dostave i preuzimanja robe, dok god ima dovoljno mjesta u određenom vozilu. Konačno, VRPTW (VRP with Time Windows) definira problem u kojem svaki klijent, pored standardnih ograničenja, ima definiran i vremenski prozor unutar kojeg se narudžba mora izvršiti.

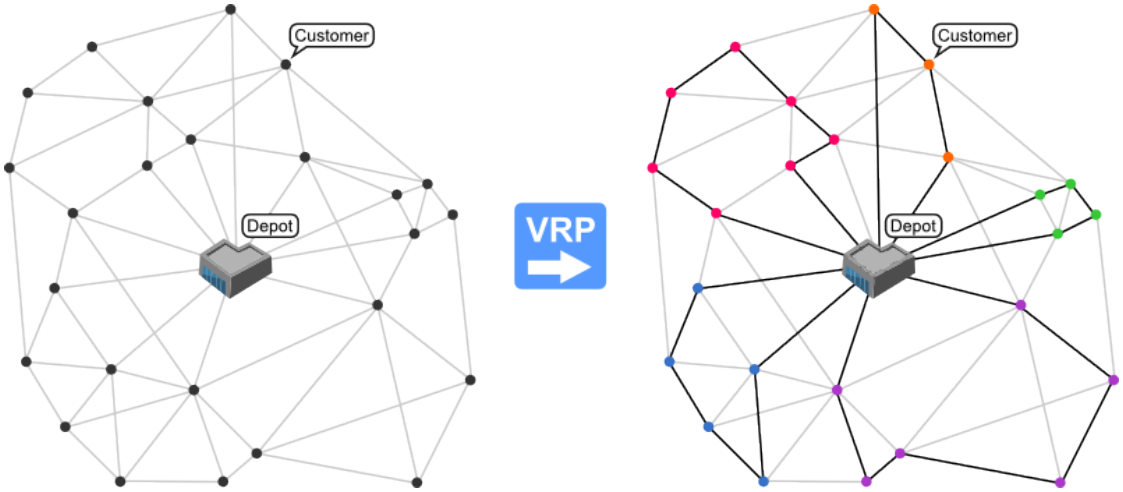
No, u stvarnosti su mnogi problemi dinamičke prirode pa tako i VRP, budući da se mnoge nepredvidive stvari mogu dogoditi prilikom izvođenja početnog plana, primjerice, kvar vozila, gužve na cesti, dolazak novih zahtjeva i sl. Stoga će u ovom radu najveći fokus biti na rješavanju dinamičkog VRP-a (DVRP) koji je danas vrlo aktualna i još relativno neistražena tema. Dolaskom svakog zahtjeva potrebno je odlučiti hoćemo li taj zahtjev prihvatiti ili odbiti s obzirom na trenutno stanje u našoj floti vozila, dakle, dostupnost slobodnih vozila, dovoljan kapacitet, mogućnost dolaska unutar zadanog vremenskog prozora i sl. Moguća rješenja koja će se uzeti u obzir prilikom

izrade projekta su evolucija hiperheuristika te korištenje konstrukcijskih heuristika.

U drugom poglavlju bit će detaljnije opisani statički i dinamički problem usmjeravanja vozila. U trećem poglavlju bit će opisane metode koje se najčešće koriste kao rješenja ovog problema. U četvrtom poglavlju bit će riječi o genetičkom i analitičkom programiranju. Peto poglavlje pojasnit će detalje na implementacijskoj razini o komponenti za rješavanje DVRP-a. Posljednje, šesto poglavlje bit će posvećeno rezultatima dobivenim uz korištene heuristike.

2. Problem usmjeravanja vozila

Problem usmjeravanja vozila pojavio se kao potreba sve jače rastućoj industriji da se suoči s izazovima što učinkovitije dostave dobara svojim klijentima kako bi se ostvarile maksimalne uštede. Primjer jednog rješenja takvog problema nalazi se na slici 2.1. Prvo pojavljivanje rješavanja takvog problema je u radu Georgea Dantziga i Johna Ramsera u 1959. [4] u kojem se prvi put opisivao algoritamski pristup rješavanju problema dostave goriva. Dok se u počecima najviše pažnje posvećivalo statičkoj varijanti sa fiksnim zahtjevima klijenata i njihovim parametrima, danas se sve više istražuje dinamičko okruženje pa će u nastavku biti opisane obje varijante problema.



Slika 2.1: Primjer konstruiranih ruta kod problema usmjeravanja vozila

2.1. Statički problem usmjeravanja vozila

Statički problem usmjeravanja vozila odnosi se na problem u kojem instance imaju unaprijed zadane podatke o klijentima sa svojim zahtjevima kao što su lokacija, količina zahtijevane robe, vremenski prozor unutar kojeg vozilo može stići i sl. Kao što je ranije spomenuto, postoje mnoge varijante VRP-a koje se proučavaju, ali najvažnije

varijante, na kojima će i najviše biti fokus o ovom radu su CVRP (kapacitivni problem usmjeravanja vozila), VRPTW (problem usmjeravanja vozila s vremenskim prozorima) te VRPPD (problem usmjeravanja vozila sa prikupljanjem i dostavom robe).

2.1.1. CVRP (Capacitated Vehicle Routing Problem)

Kapacitivni problem usmjeravanja vozila [13] je najosnovnija varijanta VRP-a u kojoj vozila imaju ograničen kapacitet te je potrebno osigurati da količina robe koju vozilo mora dostaviti na svojoj ruti ne premaši kapacitet vozila, odnosno, vrijedi:

$$\sum_{i=1}^{m} d_i \leq Q \tag{2.1}$$

, gdje je Q kapacitet vozila, a d_i količina robe koja je potrebna pojedinom klijentu.

CVRP također možemo podijeliti na homogeni i heterogeni problem [1]. U homogenom problemu, svako vozilo ima jednaki kapacitet Q, a kako bi se osiguralo da su sva vozila dovoljno velika, potražnja svakog klijenta nikad nije veća od kapaciteta bilo kojeg vozila, odnosno $q_i \leq Q(1 \leq i \leq n)$. Također, ukupne potražnje od svih klijenata ne smiju biti veće od ukupnog kapaciteta svih vozila, tj. $(\sum_{i=1}^{n} q_i) \leq m * Q$.

Drugi tip je heterogeni CVRP u kojem se razmatra flota različitih kapaciteta, a potražnja svakog klijenta nikad nije veća od kapaciteta najvećeg vozila, tj. $q_i \leq Q_{max}(1 \leq i \leq n)$. Za heterogeni CVRP vrijede ista ograničenja kao i za homogeni.

U ovom radu koriste se instance (o kojima će biti više riječi kasnije) u kojima svako vozilo ima isti kapacitet, zadan na početku svake instance; prema tome, fokus će biti isključivo na homogenom CVRP-u.

2.1.2. VRPTW (Vehicle Routing Problem with Time Windows)

VRPTW nasljeđuje sve karakteristike CVRP-a, uz dodatni uvjet da svaki klijent bude poslužen unutar zadanog vremenskog prozora (Cordeau et al., 2001), dakle, svaki klijent definira interval $[s_i, e_i], 1 \leq i \leq n$ gdje s_i označava najraniji, a e_i najkasniji mogući dolazak vozila do klijenta i početak posluživanja. Krajnji cilj problema je minimizirati broj vozila u floti te minimizacija ukupnog prijeđenog puta svih vozila u floti.

Kako bi rješenje VRPTW-a bilo valjano, potrebno je zadovoljiti sljedeće uvjete: vozilo mora doći do svakog klijenta prije gornje granice definiranog intervala za svakog klijenta te ukupni kapacitet robe koja se dostavlja ne smije premašiti kapacitet vozila, kao i u CVRP-u. Vozilo smije doći do klijenta i prije početka vremenskog intervala za posluživanje klijenta, no to će prouzrokovati gubljenje vremena tijekom kojeg se mogao poslužiti neki drugi klijent i to će biti jedna od ključnih problema za prepoznavanje od strane optimizacijskog algoritma.

Potrebno je napomenuti da postoji i varijanta sa mekim vremenskim prozorom [6] kod kojeg se može namjestiti granica do koje će se tolerirati kašnjenje vozila kod nekog klijenta. Rješenje koje će sadržavati kašnjenja vozila do određene gornje granice i dalje će biti valjano, no može se dodatno kažnjavati proporcionalno kašnjenjima prikupljenim kod svih vozila.

2.1.3. VRPPD (VRP with Pickup and Delivery)

VRPPD, odnosno, problem usmjeravanja vozila sa prikupljanjem [15] i dostavom je problem koji, ne samo da sadrži sva ograničenja prethodne dvije varijante, već postoji i mogućnost da se od klijenata roba i prikuplja, osim što se dostavlja. VRPPD dolazi u mnogobrojnim varijantama i svaka od tih varijanti se može riješavati na više načina, ali u radovima se najčešće spominju sljedeće 3 varijante VRPPD-a:

1. VRPB (VRP with Backhauling), gdje se obrađuju zahtjevi na način da se najprije izvršavaju sve dostave, a onda se prelazi na prikupljanja dobara
2. VRPMDP (The VRP with Mixed Deliveries and Pickups) je podvrsta VRPPD-a u kojem nije bitan redoslijed izvršavanja prikupljanja i dostava, dok god se ne krše dana ograničenja
3. VRPSDP (The VRP with Simultaneous Deliveries and Pickups) u kojem klijenti mogu zatražiti i dostavu i prikupljanje robe od strane vozila koje ih poslužuje, dakle, klijenti mogu imati kombinirane zahtjeve, a vozila u jednom zaustavljanju kod klijenta mogu napraviti obje akcije

U svakom slučaju, rješenje generirano u VRPPD-u je valjano ako poštuje sva ograničenja navedena prije (dakle, dovoljno veliki kapacitet i poštivanje vremenskih prozora), kao i ograničenja ovisna o varijanti problema i načinu obrade zahtjeva. U prvoj

varijanti će tako biti potrebno da se sve dostave u jednoj ruti izvrše prije svih prikupljanja, u drugoj će biti potrebno da se klijenti od kojih se skuplja roba nađu u ruti prije klijenata kojima je ta roba namijenjena, kao i u trećoj varijanti, naravno, ako su parovi (prikupljanje, dostava) definirani u pripadnoj instanci, inače je bitno da se samo pazi na kapacitet prilikom prikupljanja i/ili dostave robe kod svakog klijenta.

U ovom radu će biti obuhvaćena druga varijanta problema, s obzirom da su instance dostupne za VRPPD konstruirane na način da su definirani parovi klijenata kod kojih prva komponenta označava klijenta kod kojeg se prikuplja, a druga klijenta kod kojeg se dostavlja roba prikupljena kod prvog klijenta i tako za svaki par; prema tome, da bi se postigla maksimalna učinkovitost usluge, dostave i prikupljanja će se moći miješati.

2.2. Dinamički problem usmjeravanja vozila

Dosada smo razmatrali statičke varijante problema usmjeravanja vozila. Sve navedene statičke varijante, koje se mogu i kombinirati, mogu donekle modelirati ponašanje sustava u stvarnom svijetu. No, u stvarnosti, okruženje je obično dinamično i novi zahtjevi mogu stići u stvarnom vremenu tijekom izvršenja ruta. Stoga će glavni fokus rada biti upravo na dinačkom rješavanju problema usmjeravanja vozila.

Tipičan primjer je usluga prikupljanja i dostave u istom danu koje pružaju tvrtke za isporuku. U ovom slučaju, vozila se šalju kako bi poslužila postojeće zahtjeve, međutim, novi zahtjevi mogu stići dok su vozila još uvijek na putu. Tvrtka namjerava prihvatiti što veći broj novih zahtjeva s obzirom na kapacitet vozila, definirane vremenske prozore i druga zadana ograničenja, ovisno o problemu. Kad god se pojavi novi zahtjev, potrebno je odlučiti hoćemo li prihvatiti ili odbiti novi zahtjev kako bi kupac bio obaviješten na vrijeme i upravo to će nam biti glavna mjera kvalitete rješenja prilikom dinamičke simulacije - broj odbijenih zahtjeva.

Formalno, dinamički problem usmjeravanja vozila definirat ćemo na sljedeći način. Klijente kojima se dostavlja roba možemo prikazati kao čvorove potpuno povezanog grafa $G = (V, E)$, gdje se nalazi i početna točka za svako vozilo, odnosno, skladište. Svaki vrh v_i ima svoje koordinate (x_i, y_i) pomoću kojih se izračunaju udaljenosti između svih parova klijenata, dakle, govorimo o potpuno povezanom grafu, a udaljenosti koje izračunamo gledamo kao trošak putovanja $c(v_i, v_j)$.

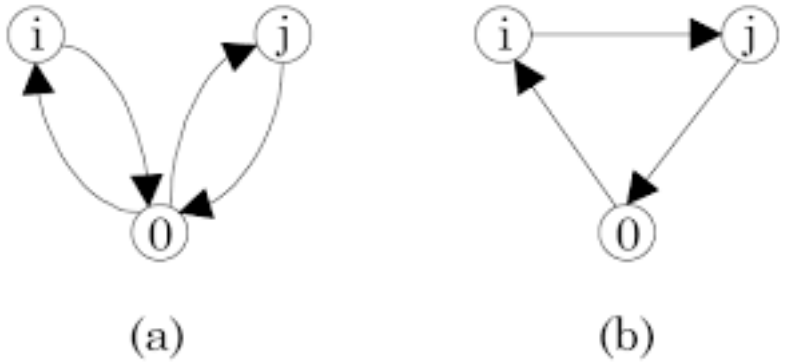
Dinamička simulacija će se odvijati kao diskretan proces dolazaka zahtjeva od trenutka $t = 0$ do $t = T_{max}$, gdje T_{max} predstavlja vremensku oznaku posljednjeg dinamički pristiglog zahtjeva. Rješenje dinamičkog problema je zapravo proces donošenja odluka gdje se donosi odluka pri svakom dolasku novog zahtjeva te ako se uz taj zahtjev može generirati valjano rješenje, on se prihvaća, inače se odbija i simulacija se nastavlja s prethodnim već izgeneriranim rješenjem. Učestalost dolazaka dinamičnih zahtjeva može se opisati mjerom dinamičnosti koja se kreće od 0 do 1 - 0 znači da znamo sve klijente unaprijed, a 1 da će svi klijenti dolaziti kasnije, tijekom simulacije u točno određenom vremenskom trenutku.

Pored dolazaka zahtjeva definiranih u instanci, postoji druga os dinamičkih promjena tijekom simulacije, a to je stohastička promjena svojstava klijenata kroz vrijeme kao što su koordinate položaja te njihova potražnja. Takva komponenta promjena također ima svoju mjeru dinamičnosti, odnosno učestalosti dolazaka do određenih promjena. Ukoliko se desi bilo koja od tih promjena, također se generira novo rješenje uz takve promjene te se takva promjena odbija ako se ne može konstruirati valjano rješenje za takve parametre.

3. Metode rješavanja problema usmjeravanja vozila

Kroz povijest su se javile brojne različite metode rješavanja, kako statičke, tako i dinamičke varijante VRP-a, a u nastavku će ukratko biti objašnjene one najvažnije i one koje su najviše inspirirale ovaj rad.

Nekoliko godina nakon što je VRP prvi put predstavljen, Clarke i Wright (slika 3.1) [3] poboljšali su početni pristup Dantziga i Ramzera iz 1959. predlaganjem heuristike uštede. Heuristika uštede je bila jednostavna za implementaciju i time su proizveli prilično dobra rješenja u to doba istraživanja. Heuristika uštede bazirala se na tome da su se od početnih ruta sastavljenih od jednog klijenta izgrađivale nove rute spajanjem starih ruta na način da se spajanjem dobije najveća ušteda u ukupnoj udaljenosti. Od tada je postala jedna od najpoznatijih heuristika za rješavanje VRP-a. Nakon uspjeha ova dva inovativna rada, predloženi su brojni modeli sa egzaktnim i heurističkim metodama kojima su se riješavali VRP i njegove varijante.



Slika 3.1: Primjer spajanja ruta kod Clarke & Wright heuristike

Zbog uspjeha ranih heurističkih metoda za rješavanje klasičnog VRP-a istraživanja

početkom 90-ih su se razvila do korištenja složenijih heurističkih algoritama i metaheurističkih pristupa. Prva od njih je heuristika umetanja koja je, kao čisti konstruktivni algoritam, poslužila za iterativno izgrađivanje valjanih ruta umetanjem jednog po jednog zahtjeva u rute u koje je taj zahtjev mogao ući poštujući sva ograničenja.

Zatim slijede poboljšavajuće heuristike koje već generirana rješenja daljnjim zamjenama grana i klijenata unutar rute poboljšavaju generirajući susjedstvo rješenja i prelazeći na boljeg susjeda. Primjeri takvih heuristika su k-opt swap (micanje k grana i dodavanje k novih grana unutar rute), Or-opt swap (micanje jednog ili niza klijenata iz rute i premještanje na neki drugi dio rute), mehanizam λ-zamjene (u kojem se zamjeni sadržaj dviju ruta tako da se iz prve i druge rute uzme podskup klijenata koji ne smije biti veći od λ te se ta dva podskupa zamijene) itd. Aproksimativne metode temeljene na gradijentnom spustu, hibridno simulirano kaljenje s tabu pretraživanjem i tabu pretraživanje razvio je Osman 1993. [12].

Kasnije se počelo sve jače širiti istraživanje dinamičkog VRP-a pa je tako AbdAllah 2013. [1] odlučio riješiti taj problem korištenjem genetskog algoritma koji optimizira statičku VRP instancu na svaki dolazak novog zahtjeva. Problem je bio podijeljen na 2 podsistema: upravitelj događaja i optimizacijski podsistem, a kodiranje kromosoma GA proveo je uz pomoć cijelih brojeva koji označavaju korisnike (negativni brojevi označavali su vozilo kako bi ograničio dvije različite rute). Zaključak je bio da je DVRP baziran na GA značajno boljih performansi od npr. PSO, Ant system ili Tabu pretrage budući da je bilo pronađeno 66 % novih boljih rješenja.

U doktorskoj disertaciji Penny Louise Holborn iz 2013. [7] se dotaknula zanimljive teme statičkih i dinamičkih rješenja problema usmjeravanja vozila sa prikupljanjima i dostavama uz vremenske prozore. Tamo su opisane mnoge heuristike vezane za DPDPTW kao što su konstrukcijske heuristike umetanja novih zahtjeva na različite načine te poboljšavajuće heuristike. U dinamičkom problemu najprije se generira statičko rješenje, a nakon toga se primjenjuju heuristike za uključivanje pristiglih zahtjeva u trenutna rješenja, a neke od tih heuristika su bile umetanje zahtjeva s minimalnim ukupnim troškom, umetanje novih zahtjeva i zahtjeva koji još nisu fiksirani (NFR, *eng. Non-Fixed Requests*) te umetanje novih zahtjeva i lokacija fiksiranih zahtjeva na koje se mora izvršiti dostava (NFL, *eng. Non-Fixed Locations*), a svaki je testiran sa 3 načina umetanja: nasumičan (pritom pazeći na izvedivost), pohlepan (minimizirajući povećanje cijene) te tzv. slack (uzimajući u obzir najprije najhitnije zahtjeve). Uz sve

to su se koristile i poboljšavajuće heuristike kao što su Tabu-pretraga, Branch and Bound, a najbolji rezulati dobiveni su sa slack heuristikom poboljšanom s Tabu-pretragom i Brach and Bound heuristikom te podacima sa zahtjevima velike hitnosti.

Rad iz 2017. [8] pozabavio se dinamičkim problemom usmjeravanja vozila sa vremenskim prozorima. Koristi se meta algoritam kojim se u svakoj iteraciji preuzima određen zahtjev i određuje se može li ga se prihvatiti na temelju prethodnih već prihvaćenih zahtjeva, a odabir slijedećeg zahtjeva vrši se temeljem heurističke funkcije koja se može definirati ručno ili se može evoluirati pomoću GP. Rezultati su pokazali da su puno bolji uspjeh dale evoluirane nego ručno kreirane heuristike, da stavljanje probabilističkih završnih funkcija nije uvijek najbolja ideja te da je bolja wait-first nego drive-first strategija. Wait-first i drive-first su strategije razvijene kako bi se riješilo pitanje vremena polaska vozila pa dok wait-first nalaže da vozilo čeka dok god ima dovoljno vremena da se posluže svi preostali klijenti, drive-first strategija označava pravilo polaska vozila neposredno nakon posluživanja klijenta kako bi se što prije stiglo do sljedećeg klijenta.

Konačno, Haitao Xu [16] rješavao je dinamički VRP korištenjem algoritma optimizacije kolonijom mrava (eng. Ant Colony Optimization, ACO) poboljšanog sa križanjima, K-means i 2-Opt algoritmom. K-means grupirao je početni prostor korisnika u manje grupe, a svaku takvu regiju je onda rješavao ACO. U ACO se potom ugrađivao operator križanja koji je odabirao koji korisnici u kojim rutama će se križati te se koristila 2-Opt heuristika lokalne pretrage kako bi se zamjenom parova susjednih lokacija unutar istog puta eventualno dobilo bolje rješenje. Ovako poboljšani ACO parira svim novije objavljenim sustavima za rješavanje DVRP-a, pa čak je i nešto bolji.

4. Metode simboličke regresije

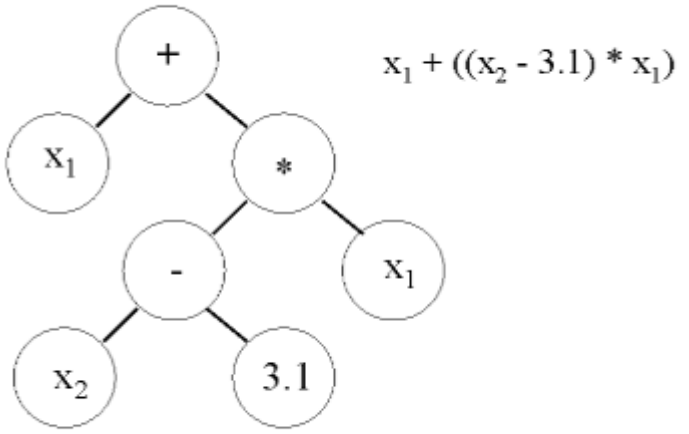
Simbolička regresija pokazala se u mnogim problemima kao iznimno moćna metoda za dobivanje optimalnih rješenja, a u nastavku će biti opisane dvije metode simboličke regresije korištene u ovom radu - genetičko i analitičko programiranje.

4.1. Genetičko programiranje

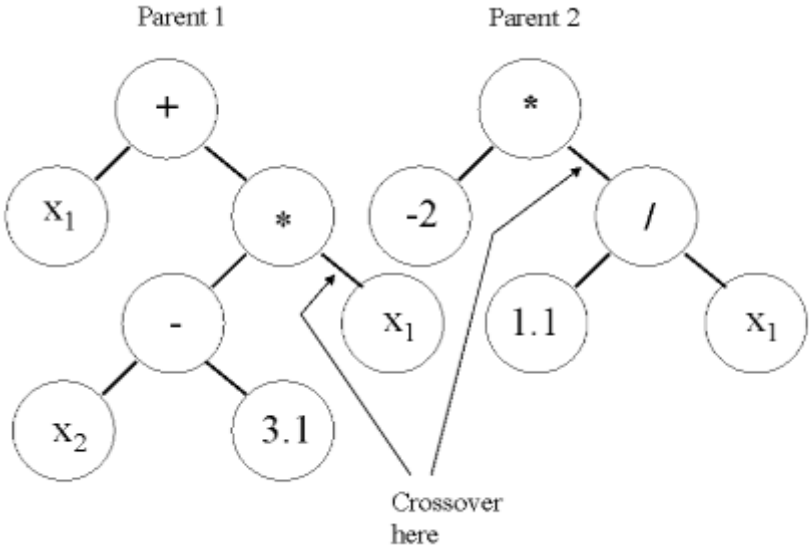
Genetičko programiranje je ideja Johna Koze [9], koji proširuje rad koji je pokrenuo John Holland. Do danas se GP koristio u brojnim optimizacijskim problemima jer se pokazao prilično uspješnim u njihovom rješavanju, ali i zato što ima nešto drugačiji pristup od ostalih metoda evolucijskog računarstva. Naime, Koza je gledao računalne programe kao organizme koji bi se mogli razvijati kako bi bolje obavili svoje dužnosti. Njegovi su sustavi koristili princip prirodne selekcije za odabir računalnih programa koji bi proizvodili potomstvo - nove programe koji bi bolje obavljali svoje zadatke.

Na početku se GP koristio kako bi se pronašla formula koja bi se što bolje prilagodila zadanim ulaznim podacima, odnosno, pronalazio bi se meta model u strojnom učenju koji bi bio bolji od svih fiksnih modela kojima bi se tražili samo parametri, a kasnije se počeo koristiti i na sve složenijim optimizacijskim problemima, ali i na svim drugim koji zahtijevaju generiranje određenog algoritma, odnosno programa. Dakle, genetičko programiranje evoluira skup programa koji služe za generiranje pravih rješenja na instancama problema, a svaki od generiranih programa se evaluira kako bi se odabrao što bolja jedinka.

Najprije se nasumično izgenerira unaprijed zadan broj jedinki koje su u obliku stabla, prikazanog na slici 4.1, a svaki čvor stabla se sastoji od funkcija koje mogu primati argumente te terminala (završnih znakova) koji su specifični za svaku domenu. Naravno, čvorovi s funkcijama uvijek imaju djecu, a čvorovi s terminalima su završni čvorovi, odnosno, listovi stabla.

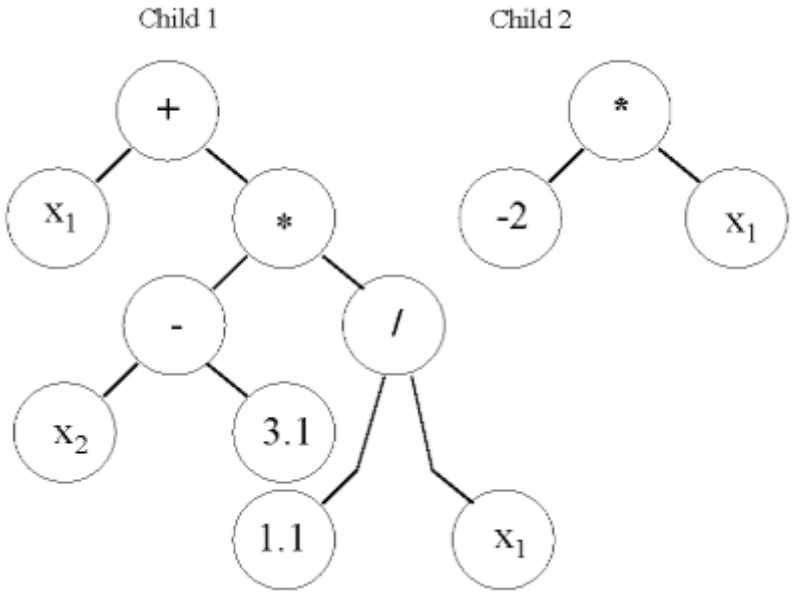


Slika 4.1: Primjer stabla generiranog tijekom GP-a



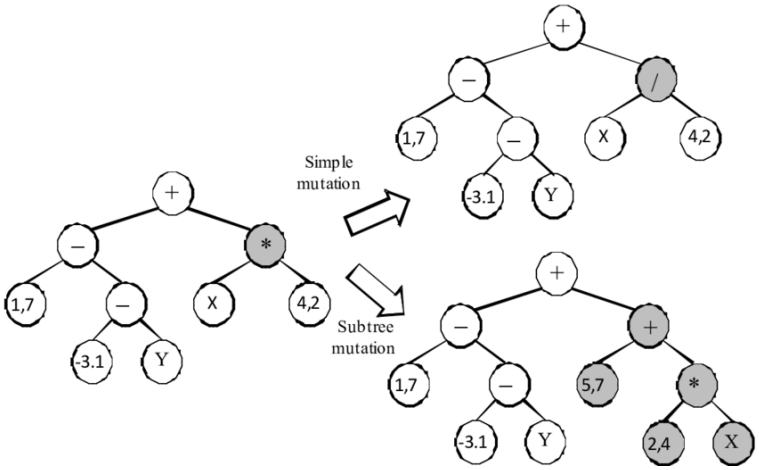
Slika 4.2: Izgled stabala prije križanja

Zatim se svaka jedinka evaluira na skupu za učenje vezanog za određeni problem te slijedi selekcija jedinki. Nakon selekcije para jedinki, vrši se križanje koje kod stabala izgleda tako da se odaberu podstabla kod prve i kod druge jedinke te se jednostavno zamijene, što je prikazano na slikama 4.2 i 4.3 . Nakon križanja slijedi mutacija koja će s vrlo malom vjerojatnošću promijeniti stablo na dva moguća načina: jednostavnom mutacijom ili mutacijom podstabla. U jednostavnoj mutaciji samo će se promijeniti nasumično odabrani čvor, uz pažnju da se promijeni u operator s jednakim brojem argumenata kao i operator koji je bio u čvoru prije zamjene, dok će se u mutaciji podsta-



Slika 4.3: Izgled stabala nakon križanja

bla promijeniti taj čvor i cijelo njegovo podstablo, što zapravo izgleda kao generiranje potpuno novog stabla unutar jedinke. Konačan rezultat obaju načina mutacije nalazi se na slici 4.4. Ovakav postupak evolucije stabala ponavlja se zadan broj generacija ili dok se ne postigne neki drugi uvjet zaustavljanja, npr. dosegnuta željena dobrota.



Slika 4.4: Izgled stabala nakon mutacije

Pored navedenih operatora križanja i mutacije, postoje još i jednostavno križanje, uniformno križanje, križanje s čuvanjem konteksta te križanje u jednoj točki. Dodatni operatori mutacije su mutacija permutacijom, mutacija koristeći Gaussovu razdiobu, zamjena stabla njegovim podstablom, mutacija komplementom čvora te mutacija sma-

njivanjem nasumičnog podstabla.

4.2. Analitičko programiranje

Analitičko programiranje inspirirano je numeričkim metodama u Hilbertovim funkcionalnim prostorima i genetičkim programiranjem. Naime, od genetičkog programiranja proizlazi ideja evolucijskog stvaranja simboličkog rješenja, dok se ideje funkcionalnih prostora i izgradnje rezultirajuće funkcije uz pomoć procesa pretraživanja (obično Ritz ili Galerkin metoda) prihvaćaju se iz Hilbertovih prostora. Poput GP ili GE (gramatička evolucija), AP se temelji na skupu funkcija, operatora i terminala (završnih znakova), koji su obično konstante ili nezavisne varijable [17].

Svi ti objekti stvaraju skup iz kojeg se pokušava sintetizirati odgovarajuća formula. Glavno načelo analitičkog programiranja temelji se na diskretnom upravljanju skupovima (Discrete set handling, DSH). DSH se može shvatiti kao sučelje između korištenog evolucijskog algoritma i problema koji mora biti riješen simboličkom metodom. Zbog te razdvojenosti analitičko programiranje se može poslužiti gotovo svim evolucijskim algoritmima.

Kao što je ranije spomenuto, koristimo skup matematičkih objekata kao što su funkcije, operatori i terminali te su pomiješani u jedan zajednički skup. Zbog takve varijabilnosti unutar jednog skupa, taj skup se naziva opći funkcionalni skup (eng. General functional set, GFS). Međutim, taj skup ima ugniježđenu strukturu, odnosno, sastoji se od podskupova na način da svaki podskup sadrži samo one funkcije koje imaju isti broj argumenata. Tako, primjerice, imamo opći skup, koji se još naziva GFS_{all}, skup funkcije od 3 varijable GFS_{3arg} te, primjerice, skup terminala koji se nalaze u GFS_{0arg}.

Jedinke u analitičkom programiranju se sastoje od nenumeričkih izraza (kao što su operatori, funkcije i sl.) koji su u evolucijskom procesu izraženi njihovim cjelobrojnim indeksima ili poljima realnih brojeva koji se potom diskretizacijom mogu pretvoriti u cjelobrojne indekse. Ti indeksi služe kao pokazivači u skupu izraza te se koristi za sintezu rezultirajućih programa na kojima se onda može raditi evaluacija dobrote.

Postoje tri verzije analitičkog programiranja. Sve tri verzije koriste se za simboličku regresiju. Prva verzija, imena AP_{basic} (osnovni algoritam) koristi nasumično

generirane konstante prilikom sinteze pojedine formule. Druga verzija (AP_{meta}) je modificirana u vidu procjene konstanti. Naime, ovdje se prilikom izgradnje formule za konstantu koristi samo oznaka K, a nakon sinteze formule su svi K-ovi indeksirani rednim brojem te se procjenjuju pomoću drugog evolucijskog algoritma. Budući da sada imamo dva evolucijska algoritma (jedan za sintezu formule, a drugi za procjenu parametara), ova verzija se naziva analitičko programiranje s metaevolucijom. Budući da druga inačica zahtijeva prilično vremena, AP_{meta} je izmijenjen na treću verziju, koja se razlikuje od druge u procjeni konstanti, što se radi korištenjem prikladne metode za nelinearno podešavanje (nonlinear fitting) (AP_{nf}). Ova je metoda pokazala najbolje performanse prilikom procjene nepoznatih konstanti.

Prisutnost strukture podskupa u GFS-u je vrlo važna za analitičko programiranje jer se koristi za izbjegavanje sinteze nevaljanih programa, tj. programa koji sadrže funkcije bez argumenata ili premalo argumenata ili s desne strane operatora nema ničega itd. Također je važan dio analitičkog programiranja i slijed matematičkih operacija koje se koriste za sintezu programa. Te se operacije koriste za transformaciju jedinke populacije evolucijskog algoritma u odgovarajući program, tj. mapiranje iz domene jedinki u domenu programa (formula).

Mapiranje se sastoji od dva glavna dijela. Prvi dio je DSH, a drugi su sigurnosni postupci koji ne dopuštaju sintetizirati nevaljane programe. DSH se koristi za obradu proizvoljnih objekata poput jezičnih pojmova, logičkih izraza ili korisnički definiranih funkcija. U AP-u, DSH se koristi za mapiranje jedinke u GFS i zajedno sa sigurnosnim postupcima stvara mapiranje koja pretvara proizvoljnu jedinku u program. Jedinke u populaciji čine cjelobrojni parametri koji pokazuju na elemente skupa GFS.

$$individual = \{\, 1, 6, 7, 8, 9, 11 \,\}$$

$$GFS_{all} = \{\, +, -,\ /,\ a^b,\ d/dt, Sin, Cos, Tan, t, C1, Mod, \ldots \,\}$$

$$GFS_{0arg} = \{\, t, x, y, z, C1, C2, Kinchin, t, C4, C5, \omega, \ldots \,\}$$

$$X + Y \rightarrow Sin(X) + Cos(Y) \rightarrow Sin\big(Tan(X)\big) + Cos(t) \rightarrow Sin\big(Tan(\omega)\big) + Cos(t)$$

Slika 4.5: Primjer preslikavanja genotipa AP-a u fenotip

Analitičko programiranje je zapravo niz mapiranja jedinke evolucijskog algoritma u formulu. Na slici 4.5 je pokazan primjer kako se stvara konačan program od jedne jedinke. Indeks 1 u položaju prvog parametra znači da se koristi operator + iz GFS_{all}. Budući da operator zbrajanja mora imati najmanje dva argumenta čitaju se sljedeća dva slobodna indeksa u jedinci (u ovom slučaju 6 i 7, tj. sinus i kosinus). Obje su funkcije, sinus i kosinus, funkcije koje traže jedan argument, prema tome, obje funkcije čitaju dalje po jedan slobodan indeks te ono što se nalazi unutar GFS_{all} uzimaju kao svoje argumente (tangens i varijabla t).

Kako bismo izbjegli sintezu nevaljanih funkcija postoji nekoliko sigurnosnih trikova koji se koriste u AP-u. Jedan od njih je razlog zbog kojeg se GFS sastoji od podskupova koji sadrže funkcije s istim brojem argumenata. Naime, postojanje takve ugrađene strukture omogućava korištenje posebnog sigurnosnog potprograma koji mjeri koliko je još parametara ostalo do kraja jedinke, odnosno, ako se traži u nekoj funkciji više argumenata nego što ih jedinka može dati, ta funkcija će se zamijeniti drugim funkcijama s istim indeksnim pokazivačem iz podskupa s nižim brojem argumenata. To se upravo događa na primjeru na slici, naime, ako zadnji argument za funkciju jednog argumenta nije terminal, što u ovom slučaju nije, spuštamo se po podskupovima sve dok ne dođemo do podskupa koji sadrži zadovoljavajući broj argumenata, što je u ovom slučaju varijabla.

Duljina genotipa kod analitičkog programiranja može biti zadana parametrom, no bitno je napomenuti da to ne znači nužno da će duljina generirane formule biti jednaka toj duljini. Naime, kako prolazimo po genotipu i kreiramo formulu, pratimo istovremeno koliki je broj argumenta funkcije na koju smo naišli u GFS-u kao i koliko je još gena ostalo do kraja jedinke te ako smo blizu kraja, jednostavno prelazimo na sljedeći GFS s kojim nećemo prekoračiti jedinku. Time se može desiti da je duljina fenotipa manja od duljine genotipa, ponekad i drastično manja, ovisno o odabiru fukcija i sadržaju genotipa.

U ovom radu koristit će se algoritam diferencijalna evolucija pa ćemo se u nastavku samo fokusirati na njega. Diferencijalna evolucija je vrsta evolucijskog algoritma koja se zasniva na generiranju novih jedinki korištenjem razlika trenutnih jedinki [14]. Na početku, kao i obično, se inicijalizira populacija jedinki sa slučajnim realnim brojevima. Nadalje se u svakoj generaciji ponavlja isti postupak: za svaki primjer se generira tzv. mutirani vektor koji se može izračunavati na različite načine, primjerice, tako

da se svaki put uzmu slučajne jedinke x_{r1}, x_{r2}, x_{r3} te računamo vektor po formuli:

$$v_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{4.1}$$

Parametar F nam služi za upravljanje veličinom utjecaja operatora diferencijalne mutacije i poprima vrijednosti između 0 i 1. Tada se radi križanje svake jedinke i pripadnog vektora po komponentama prema formuli:

$$u_{j,i} = \begin{cases} v_{j,i}, \ ako\ je\ rand(0,1) \ \leq \ CR\ ili\ je\ j = j_{rand} \\ x_{j,i}, \ za\ sve\ ostalo \end{cases} \tag{4.2}$$

CR je parametar križanja, a jrand je slučajna vrijednost između 1 i dimnezije vektora. Nakon križanja se provodi selekcija na način da se između svakog para jedinke i pripadnog vektora u , odabire onaj koji ima veću dobrotu te se takav postupak ponavlja do kraja. Pseudokod koji je upravo opisan nalazi se u algoritmu 1.

Algorithm 1 Pseudokod diferencijalne evolucije

Ulaz: evolutionParameters
Izlaz: best
population := generatePopulation()
repeat
  for $(i := 0; i < size(population); inc(i))$ **do**
    x_{r1}, x_{r2}, x_{r3} := takeRandomIndividuals(population)
    j := randomInteger(1, n)
    u_i := performCrossover(x_{r1}, x_{r2}, x_{r3}, x_i)
    if u_i.fitness() > x_j.fitness() **then**
      x_i = u_i
    end if
  end for
until stopCriterion
return population.best()

5. Implementacija sustava i primjena metoda rješavanja

Cijeli algoritam učenja i testiranja formula generiranih od strane genetičkog i analitičkog programiranja implementiran kao dio sustava evolucijskog računarstva (ECF). U nastavku će biti potanko objašnjeni svi implementacijski detalji, kao i upute za korištenje sustava za rješavanje dinamičkog problema usmjeravanja vozila.

5.1. Korištene instance

5.1.1. Gehring & Homberger instance

Za CVRP i VRPTW korištene su Gehring & Homberger [5] instance, čiji je primjer dan na slici 5.1.

Na početku svake instance definirano je njeno ime te maksimalan broj raspoloživih vozila zu tu instancu i kapacitet svakog vozila. Zatim se generira niz zapisa o klijentima u toj instanci, a za svakog klijenta postoji informacija o njegovom indeksu, koordinatama, količini robe koju traži, donja i gornja granica vremenskog prozora unutar kojeg vozilo mora stići do klijenta te vrijeme potrebno da se klijent posluži.

Instance su podijeljene u 6 klasa problema: C1, C2, R1, R2, RC1, RC2. Skup instanci koje počinju s C imaju grupirane klijente čiji su vremenski prozori kreirani na temelju poznatog rješenja. Skup instanci sa oznakom R raspršeni su uniformno nasumično po kvadratnoj površini. Skup RC ima kombinaciju oba prethodna načina generiranja klijenata. Problemi sa oznakom 1 sadržavaju klijente sa vrlo malim vremenskim prozorima te je zadan mali kapacitet vozila, dok je potpuno obrnut način kreiranja klijenata u problemima s oznakom 2. Kada sve to iskombiniramo, dobijemo 3 * 2 = 6 klasa problema na kojima se može vrlo vjerno modelirati stvarna situacija te se može time naučiti što bolja moguća heuristika.

r1_2_1

VEHICLE
NUMBER CAPACITY
50 200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	70	70	0	0	634	0
1	107	77	34	37	47	10
2	109	139	8	123	133	10
3	120	22	39	124	134	10
4	48	47	19	31	41	10
5	116	22	32	148	158	10
6	12	138	14	285	295	10
7	86	40	22	34	44	10
8	121	124	21	160	170	10
9	61	57	35	385	395	10
10	40	113	23	487	497	10
11	129	24	18	345	355	10
12	12	84	16	209	219	10
13	44	116	27	63	73	10
14	102	52	15	169	179	10
15	41	36	13	61	71	10
16	132	133	24	150	160	10
17	104	139	29	346	356	10
18	104	54	13	63	73	10
19	22	104	11	241	251	10
20	46	133	7	86	96	10

Slika 5.1: Primjer Gehring & Homberger instance

5.1.2. Li & Lim instance

Instance koje su se koristile za VRPPD su Lim & Lim [10] instance čiji je primjer dan na slici 5.2.

U prvom retku su definirani opći podaci koje mora poštivati rješenje problema, a to su redom maksimalni broj vozila, kapacitet svakog vozila i brzina koja se ne koristi. Nadalje su definirani klijenti, slično kao kod Gehring & Homberger instanci, a za svakog klijenta generirani su podaci o njegovoj oznaci, položaju te potražnji koja je pozitivna ako se radi o preuzimanju, a negativna ako se radi o dostavi robe. U nastavku su na raspolaganju i podaci o vremenskom prozoru kao u VRPTW-u te vrijeme posluživanja klijenta, a na kraju su navedeni i indeksi pripadnog para klijenta za prikupljanje, odnosno dostavu. Naime, u Li & Lim instancama vrijedi pravilo da se prikupljanje robe vrši kod jednog klijenta, a ta roba je namijenjena njegovom klijentu paru i samo se njemu ta roba može dostaviti. Pogledajmo zadnja 2 stupca u primjeru instance, predzadnji stupac predstavlja indeks prikupljanja, a posljednji indeks dostavljanja. Ako se radi o klijentu od kojeg se prikuplja roba, njegov indeks za prikupljanje će biti 0, a indeks za dostavu će biti indeks onog klijenta kojemu ta roba mora biti dostavljena, a i vidimo da je ovdje potražnja pozitivna. Slično, kod klijenata kojima se dostavlja roba, indeks za prikupljanje će biti indeks para od kojeg mora stići roba, a indeks dostave će

25	1000	1						
0	40	50	0	0	960	0	0	0
1	25	85	-20	0	911	10	5	0
2	22	75	-10	0	919	10	8	0
3	22	85	10	0	910	10	0	44
4	20	80	40	644	764	10	0	70
5	20	85	20	0	909	10	0	1
6	18	75	20	388	508	10	0	43
7	15	75	-10	0	914	10	13	0
8	15	80	10	367	487	10	0	2
9	10	35	-28	371	491	10	86	0
10	10	40	30	519	639	10	0	97
11	8	40	-10	195	315	10	47	0
12	8	45	20	0	917	10	0	48
13	5	35	10	653	773	10	0	7
14	5	45	10	35	155	10	0	60
15	2	40	20	174	294	10	0	100
16	0	40	-5	255	375	10	73	0
17	0	45	-13	703	823	10	87	0
18	44	5	20	335	455	10	0	51
19	42	10	40	254	374	10	0	59
20	42	15	-8	537	657	10	72	0
21	40	5	-10	0	905	10	32	0
22	40	15	-16	375	495	10	65	0
23	38	5	-27	201	321	10	69	0
24	38	15	-20	681	801	10	41	0
25	35	5	-9	784	904	10	67	0
26	95	30	-10	0	891	10	28	0
27	95	35	20	146	266	10	0	91
28	92	30	10	149	269	10	0	26

Slika 5.2: Primjer Li & Lim instance

biti postavljen na 0, a potražnja kod takvih klijenata je uvijek negativna.

5.1.3. Dinamičke varijante instanci

Potrebno je još kratko spomenuti i dinamičke varijante ovih dviju instanci. One se ne razlikuju značajno od njihovih statičkih varijanti, osim po novom stupcu podataka AVAILABLE TIME koja govori kad će se točno pojaviti određeni klijent. Takve instance dolaze i u različitim varijantama ovisno o stupnju dinamičnosti, odnosno, po broju klijenata koji će se pojavljivati tijekom simulacije, a neće biti poznati unaprijed. Tako, primjerice, možemo imati varijante sa stupnjem dinamičnosti 0, 0.1, 0.5 i 1.0. Na slikama 5.3 i 5.4 vidimo primjere dinamičkih instanci Gehring & Homberger [11] te Li & Lim [7] sa stupnjem dinamičnosti 0.5. Postoji i varijanta dinamičkih instanci za VRPTW generirana u radu [2] u kojoj su tri instance bile korištene za generiranje 6 klasa dinamičkih problema, ovisno o stupnju dinamičnosti dolazaka klijenata, no u ovom radu nisu korištene.

Primijetimo da u Li & Lim instancama sudionici para prikupljanja i dostave imaju identično vrijeme pojavljivanja budući da oni zajedno predstavljaju jedan zahtjev koji

c101

VEHICLE
NUMBER CAPACITY
25 200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME	AVAIL. TIME
0	40	50	0	0	1236		
1	45	68	10	912	967		
2	45	70	30	825	870		
3	42	66	10	65	146		
4	42	68	10	727	782		
5	42	65	10	15	67		
6	40	69	20	621	702		
7	40	66	20	170	225		
8	38	68	20	255	324		
9	38	70	10	534	605		
10	35	66	10	357	410		
11	35	69	10	448	505		
12	25	85	20	652	721		
13	22	75	30	30	92		
14	22	85	10	567	620		
15	20	80	40	384	429		
16	20	85	40	475	528		
17	18	75	20	99	148		
18	15	75	20	179	254		
19	15	80	10	278	345		
20	30	50	10	10	73		
21	30	52	20	914	965		
22	28	52	20	812	883		
23	28	55	10	732	777		
24	25	50	10	65	144		
25	25	52	40	169	224		

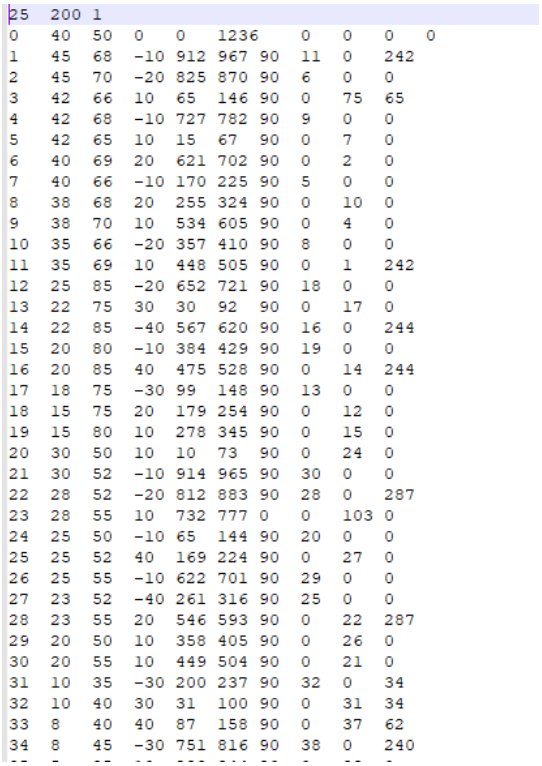
Slika 5.3: Primjer dinamičke Gehring & Homberger instance

će se kasnije, ovisno o mogućnosti konstrukcije rješenja s tim klijentima, prihvatiti ili odbiti.

5.2. Algoritam rješavanja problema usmjeravanja vozila

Problem usmjeravanja vozila može se riješiti na različite načine, kao što je opisanu u poglavlju 3., no u ovom radu taj problem je riješen koristeći metode simboličke regresije, odnosno, genetičko programiranje i analitičko programiranje. Njihova primjena je važna jer ćemo pomoću hiperheuristike koju oni formiraju moći konstrukcijskim metodama generirati rješenje problema za sve tri varijante. Konkretno, prilikom izgrađivanja rješenja potrebno je svaki put odlučiti kojeg klijenta staviti u određenu rutu, a u toj odluci nam upravo pomaže heuristika koju nam generira GP ili AP.

Kako bi simboličkom regresijom dobili što bolje programe, odnosno, heuristike, od velike je važnosti odrediti funkcije i terminale koji će imati priliku pojaviti se u



```
25  200 1
0   40  50  0   0   1236    0   0   0   0
1   45  68  -10 912 967 90  11  0   242
2   45  70  -20 825 870 90  6   0   0
3   42  66  10  65  146 90  0   75  65
4   42  68  -10 727 782 90  9   0   0
5   42  65  10  15  67  90  0   7   0
6   40  69  20  621 702 90  0   2   0
7   40  66  -10 170 225 90  5   0   0
8   38  68  20  255 324 90  0   10  0
9   38  70  10  534 605 90  0   4   0
10  35  66  -20 357 410 90  8   0   0
11  35  69  10  448 505 90  0   1   242
12  25  85  -20 652 721 90  18  0   0
13  22  75  30  30  92  90  0   17  0
14  22  85  -40 567 620 90  16  0   244
15  20  80  -10 384 429 90  19  0   0
16  20  85  40  475 528 90  0   14  244
17  18  75  -30 99  148 90  13  0   0
18  15  75  20  179 254 90  0   12  0
19  15  80  10  278 345 90  0   15  0
20  30  50  10  10  73  90  0   24  0
21  30  52  -10 914 965 90  30  0   0
22  28  52  -20 812 883 90  28  0   287
23  28  55  10  732 777 0   0   103 0
24  25  50  -10 65  144 90  20  0   0
25  25  52  40  169 224 90  0   27  0
26  25  55  -10 622 701 90  29  0   0
27  23  52  -40 261 316 90  25  0   0
28  23  55  20  546 593 90  0   22  287
29  20  50  10  358 405 90  0   26  0
30  20  55  10  449 504 90  0   21  0
31  10  35  -30 200 237 90  32  0   34
32  10  40  30  31  100 90  0   31  34
33  8   40  40  87  158 90  0   37  62
34  8   45  -30 751 816 90  38  0   240
```

Slika 5.4: Primjer dinamičke Li & Lim instance

programu. U tablicama 5.1 i 5.2 opisane su sve korištene funkcije i terminali prilikom pokretanja GP-a i AP-a.

Tablica 5.1: Funkcije

Funkcija	Opis
+, -, *, /	Osnovne operacije s 2 argumenta
pos	Ako je broj pozitivan, vraća se taj broj, inače 0, 1 argument
ifgt	Grananje, ako je prvi argument veći od drugog, vraća se treći argument, inače četvrti

Većina navedenih terminala je intuitivno jasna, odnosno, izvučena su direktno iz podataka navedenih u zadanim instancama, no, neke je potrebno detaljnije definirati. Terminal VTD, odnosno gustoća klijenata u prostoru izračunat je kao:

$$VTD(k) = \sum_{v \in V} e^{-\frac{1}{2}(\frac{100 * c(v,v_k)}{maxd(v_i,v_j)})^2} \tag{5.1}$$

Terminal VHD je izražen formulom:

Tablica 5.2: Terminali

Terminal	Opis
dist	Udaljenost do sljedećeg klijenta
d	Količina tražene robe klijenta
rc	Preostali kapacitet vozila
drc	Omjer potražnje klijenta i preostalog kapaciteta
ncc	Udaljenost do najbližeg neposluženog klijenta
st	Vrijeme potrebno za posluživanje klijenta
rt	Trenutak u kojem klijent postaje spreman za posluživanje
dd	Trenutak nakon kojeg vozilo više ne može doći do klijenta
t	Trenutno vrijeme
ttrt	Vrijeme preostalo do trenutka kad klijent postaje spreman za posluživanje
ttdd	Vrijeme preostalo do trenutka kad klijent prestaje biti spreman za posluživanje
wt	Vrijeme čekanja da klijent postane spreman ukoliko taj klijent postane odabran
tard	Zakašnjelost usluge ako je odabran ovaj klijent
vtd	Gustoća klijenata u prostoru
vhd	Gustoća vozila u prostoru
ttnv	Vrijeme putovanja do najbližeg vozila

$$VHD(k) = \sum_{i=1}^{k} e^{-\frac{1}{2}*(\frac{8*c(loc_i,v_k)}{maxd(v_i,v_j)})^2} \tag{5.2}$$

Algoritam rješavanja problema usmjeravanja vozila koji je prikazan u algoritmu 2 sastoji se od dvije neovisne komponente koje se mogu pokretati zasebno. Najprije se simboličkom regresijom, ako se odlučimo nju koristiti, nauči program na temelju statičkih instanci koje ponudimo kao jedan od ulaznih podataka. Učenje funkcionira na način da se jedinke, odnosno, programi koji se generiraju tijekom evolucije evaluiraju na skupu tih instanci, a sama evaluacija funkcionira na način da jedinku koju evaluiramo koristimo kao heuristiku za izgrađivanje rješenja, odnosno, skupa ruta, te na temelju kvalitete tog rješenja određujemo i dobrotu dane jedinke.

Generiranje rješenja možemo napraviti na dva načina: slijedno i paralelno (algoritmi 3 i 4). Serijska konstrukcija podrazumijeva generiranje rješenja vozilo po vozilo, dakle, dok god možemo staviti klijenta u rutu pojedinog vozila, izgrađujemo tu rutu sve dok više ne možemo dodati nijednog klijenta, a nakon toga završavamo rutu i pre-

Algorithm 2 Učenje heuristike na statičkim primjerima

Ulaz: instance, numOfGen
Izlaz: heuristics
population := generatePopulation()
for $(i := 0; i < size(population); inc(i))$ **do**
  unit := population(i)
  solution := generateSolution(unit, instance)
  unit.fitness = evaluateSolution(solution)
end for
for $(i := 0; i < numOfGen; inc(i))$ **do**
  tournament := selectTournament(population)
  startTournament()
  newUnit := crossover(tournament)
  newUnit := mutation(newUnit)
  solution := generateSolution(newUnit, instance, generateType)
  newUnit.fitness = evaluateSolution(solution)
  population.add(newUnit)
end for
return population.best()

lazimo na sljedeće vozilo. Paralelna konstrukcija, s druge strane, prati simultano sva vozila te se u svakom trenutku odabire vozilo koje je najranije dostupno, a nakon toga se odabire klijent koji se može dodati pomoću heuristike i on se dodaje na rutu tog vozila. Ako se nijedan klijent ne može dodati u njegovu rutu, slično kao kod serijske konstrukcije, završavamo tu rutu i proglašavamo to vozilo nedostupnim za tu iteraciju konstrukcije rješenja.

Algorithm 3 Generate Solution Serial

```
Ulaz: unit, instance
Izlaz: solution
routes := initializeRoutes()
visitedCustomers := 0
j := 0
repeat
  repeat
    minValue := MAXVALUE
    customerFound := false
    for (i := 0; i < n; inc(i)) do
      value = unit.getValue(i)
      if value < minValue then
        minValue = value
        customerFound = true
        visitedCustomers++
      end if
    end for
  until ¬customerFound
  finishRoute(route[j++])
until visitedCustomers < instance.numCustomers
return routes
```

Nakon procesa učenja, slijedi ispitivanje na skupu za ispitivanje prikazana u 5. Ispitivanje se, kao što je ranije spomenuto, može pokretati zasebno i ne treba se prije nje nužno pokretati simbolička regresija, bilo to genetičko ili analitičko programiranje. Pored ovih heuristika koriste se i dodatne jednostavne heuristike navedene u radu [8] kako bi se usporedio učinak heuristika dobivenih simboličkom regresijom, a one su redom:

Algorithm 4 Generate Solution Parallel

```
Ulaz: unit, instance
Izlaz: solution
routes := initializeRoutes()
visitedCustomers := 0
repeat
    vehicle := findAvailableVehicle()
    minValue := MAXVALUE
    customerFound := false
    for (i := 0; i < n; inc(i)) do
        value = unit.getValue(i)
        if value < minValue then
            minValue = value
            customerFound = true
            visitedCustomers++
        end if
    end for
    if ¬customerFound then
        finishRoute(route[vehicle])
    end if
until visitedCustomers < instance.numCustomers
return routes
```

1. EF (earliest-first) - najprije se poslužuje zahtjev koji se može najranije započeti, $EF(i) = max\{t + c(loc, v(i)), l(t)\}$, gdje su loc trenutni položaj, $v(i)$ položaj klijenta i, a $l(t)$ donja granica vremenskog prozora

2. UF (urgent-first) - prvo se poslužuje najhitniji zahtjev, tj. onaj kojem je kraj vremenskog prozora za posluživanje najbliže, $UF(i) = u(i)$, gdje je $u(i)$ gornja granica vremenskog prozora

3. LC (linear-combination) - računa se kao linearna kombinacija različitih atributa, $LC(i) = A1 + A2 + A3 + 0.1A4 + 0.1A5 - A6$, a oni su definirani na slijedeći način:
   - $A1 = EF(i)$
   - $A2 = UF(i)$
   - $A3 = c(loc, v(i))$
   - $A4 = (1 - \frac{\overline{Q}}{Q})c(v(i), v_0)$, gdje je $\frac{\overline{Q}}{Q}$ normaliziran trenutni kapacitet vozila
   - $A5 = d(i)$, gdje je $d(i)$ potražnja definirana u instanci
   - $A6 = min_{i=1}^{k} c(loc_i, v(i))$

4. NN (nearest neighbour) - metoda najbližeg susjeda, $NN(i) = c(loc, v(i))$

5.3. Dinamička simulacija

Dinamička simulacija, prikazana na algoritmu 5, koristi se prilikom ispitivanja rješenja dobivenih simboličkom regresijom, a može se koristiti i u metodama učenja. Simulacija kreće generiranjem početnog rješenja za klijente koji su zadani odmah na početku simulacije. Broj vozila koji se koristi u simulaciji određen je tako da se riješi taj isti statički problem na početku pomoću serijske konstrukcije, a tom broju se samo pridodaju još dva vozila, kako je navedeno u radu [2]. Nakon toga, kreće simulacija u stvarnom vremenu gdje se prilikom otkucavanja svake jedinice vremena stalno ažurira trenutna situacija u floti vozila, npr. ako je neki klijent u stvarnom vremenu upravo posjećen, taj klijent postaje zaključan i on više ne može mijenjati svoju poziciju.

Nakon što se napravi prolaz po svim rutama, provjerava se je li stigao novi zahtjev i/ili se generira određena promjena unutar instance. Ako je stigao zahtjev ili se desila promjena u podacima instance, generira se novo rješenje koje uključuje tog novog klijenta ili novu promjenu, i ako je novo generirano rješenje valjano, novi zahtjev

Algorithm 5 Dinamička simulacija

```
Ulaz: instance, heuristic, changeProbability
Izlaz: dynamicRoutes
dynamicRoutes := initializeRoutes()
rejectedRequests := 0
initialSolution = generateSolutionParallel(heuristic, instance)
for (t := 0; t < T; inc(t)) do
    updateRealTimeRoutes(dynamicRoutes)
    dynamicCustomer := extractCustomer(instance)
    instance.randomlyChangeData(changeProbability)
    newSolution := generateSolutionParallel(heuristic, instance)
    if newSolution.feasible() then
        dynamicRoutes := newSolution
    else
        rejectedRequests++
    end if
end for
return dynamicRoutes
```

(promjena) se prihvaća, inače se odbija i nastavlja se simulacija sa starim rješenjem. Simulacija se zaustavlja nakon zadnjeg pristiglog zahtjeva.

5.4. Korištenje sustava

Kao što je ranije spomenuto, sustav se sastoji od dvije glavne komponente: komponente za učenje heuristike i komponenta za ispitivanje. Obje komponente se postavljaju u datoteci s parametrima. Prvi korak prije pokretanja aplikacije je kreirati tekstualnu datoteku koja će sadržavati sve potrebne parametre za normalan rad sustava.

```
<ECF>
        <Algorithm>
            <SteadyStateTournament>
                <Entry desc="" key="tsize">7</Entry>
                </SteadyStateTournament>
        </Algorithm>
        <Genotype>
                <Tree>
                        <Entry key="functionset">+ - / * pos ifgt</Entry>
                        <Entry key="maxdepth">3</Entry>
                        <Entry key="terminalset">dist ncc d [-1 1]</Entry>
                </Tree>
        </Genotype>
```

Najprije imamo dio postavki vezan za ECF koji u pozadini vrši evolucijske funckionalnosti i ovdje se definiraju svi potrebni parametri vezani za odabrani evolucijski algoritam koji smo odabrali, konkretno, na primjeru gore, za genetičko programiranje. U čvoru $Algorithm$ definiramo evolucijski algoritam koji želimo primijeniti nad genotipom. U ovom radu za GP korištena je isključivo turnirska selekcija i, kao što vidimo, potrebno je definirati samo jedan parametar, a to je veličina turnira. Ispod čvora $Algorithm$ nalazi se čvor $Genotype$ u kojem navodimo genotip koji želimo koristiti u okviru evolucije. Za GP se navodi čvor $Tree$, a njegovi parametri su skup funkcija, skup terminala i maksimalna dubina stabla koji će se generirati prilikom kreiranja populacije stabala.

```
<ECF>
        <Algorithm>
        <DifferentialEvolution>
            <Entry desc="" key="F">0.9</Entry>
```

```
                    <Entry desc="" key="CR">0.8</Entry>
    </DifferentialEvolution>
</Algorithm>
<Genotype>
    <APGenotype>
                    <Entry desc="" key="lbound">0</Entry>
                    <Entry desc="" key="ubound">30</Entry>
                    <Entry desc="" key="dimension">30</Entry>
        <Entry desc="" key="functionset">+ - / * pos ifgt</Entry>
        <Entry desc="" key="terminalset">dist d rc drc ncc [-1 1]</Entry>
    </APGenotype>
</Genotype>
```

Ukoliko želimo koristiti analitičko programiranje te algoritam diferencijalnu evoluciju, postavke moraju izgledati kao na primjeru gore. Slično kao kod GP-a, definiramo u čvoru $Algorithm$ koji ćemo algoritam koristiti, a u slučaju AP-a, stavljamo diferencijalnu evoluciju. Unutar tog čvora moramo definirati realne parametre F i CR (koji, sjetimo se, mogu biti između 0 i 1) i time definiramo algoritam za AP. Genotip koji koristimo za AP je $APGenotype$ i njega stavljamo unutar čvora $Genotype$. Za APGenotype postavljamo donju granicu i gornju granicu mogućih brojeva koji će se generirati, dimenziju polja realnih brojeva te, kao i kod GP-a, skup funkcija i terminala.

Na donjem primjeru nalazi se drugi dio parametara koji se moraju postaviti. Čvor $Registry$ sadrži sve paramete specifične za konkretan problem koji se riješava pomoću ECF-a, što je u ovom slučaju VRP.

```
    <Registry>
            <Entry key="vrp_instances">instancesSolomon.txt</Entry>
            <Entry key="dynamic_instances">DPDVTW instances</Entry>
            <Entry key="dynamic_simulate">0</Entry>
            <Entry key="heuristic_type">0</Entry>
            <Entry key="run_symbreg">1</Entry>
            <Entry key="symbreg_type">GP</Entry>
            <Entry key="individual_file">individual.txt</Entry>
            <Entry key="data_type">solomon</Entry>
            <Entry key="abs_heuristic">0</Entry>
    <Entry key="vrp_type">vrptw</Entry>
            <Entry key="fitness_type">0</Entry>
            <Entry key="soft_windows">0</Entry>
            <Entry key="sgs_type">serial</Entry>
```

```
                <Entry key="vehicle_number">0</Entry>
                <Entry key="vehicle_cost">10000</Entry>
                <Entry key="rejected_request_cost">10000</Entry>
                <Entry key="first_only">0</Entry>
                <Entry key="change_probability">0.9</Entry>

        <Entry key="batch.repeats">0</Entry>
        <Entry key="log.filename">log.txt</Entry>
        <Entry key="log.level">3</Entry>
        <Entry key="mutation.indprob">0.1</Entry>
        <Entry key="population.size">50</Entry>
        <Entry key="term.maxgen">500</Entry>
                <Entry key="term.fitnessval">0</Entry>
    </Registry>
</ECF>
```

Parametri u zajedničkom djelu označavaju redom:

- vrp_instances definira datoteku unutar koje su popisane sve instance potrebne kako bi se provelo učenje nad njima
- dynamic_instances može primiti putanju do jedne datoteke ili do direktorija u kojem se nalaze dinamičke instance nad kojima će se testirati odabrana heuristika
- zastavica dynamic_simulate postavlja se na 1 ili 0 ovisno o tome želi li se pokrenuti dinamička simulacija na testnim instancama
- heuristic_type postavlja indeks korištene heuristike, a u tablici 5.3 dan je ključ putem kojeg se zadaje ispravan tip heuristike
- zastavica run_symbreg se postavlja na 1 ako se želi pokrenuti simbolička regresija, inače treba biti 0
- u symbreg_type postavlja se željena metoda simboličke regresije, dakle, GP ili AP
- ukoliko je odabrana heuristika pod indeksom 0 (dakle, simbolička regresija) za testiranje, pod individual_file potrebno je staviti putanju na datoteku koja sadrži ispravnu individuu koja se želi testirati
- pod data_type daje se uputa sustavu kojim tip instanci treba čitati (solomon ili lilim)
- zastavicom abs_heuristic određuje se prisutnost dodatnog djelovanja na vrijednost heuristike apsolutnom vrijednošću

– vrp_type postavlja se na cvrp, vrptw ili vrppd

– parametar fitness_type određuje kako će se računati funkcija dobrote tijekom učenja, a uglavnom je postavljena na 0

– zastavicom soft_windows određuje se hoće li se koristiti meki vremenski prozori u VRPTW varijanti

– sgs_type određuje hoće li se tijekom učenja rješenja graditi paralelno ili serijski (parallel, serial)

– vehicle_number postavlja potreban broj vozila za paralelnu konstrukciju rješenja

– vehicle_cost je parametar koji odlučuje koliko će se kažnjavati broj vozila u rješenju

– rejected_request_cost je parametar koji odlučuje koliko će se kažnjavati broj odbijenih zahtjeva

– first_only je zastavica koja, ako je postavljena, uzrokuje izgradnju rješenja samo na temelju prve instance, dok se za sve ostale koristi jedno te ista ruta

– change_probability se postavlja za vjerojatnost dolazaka stohastičkih promjena podataka unutar instance, kao što su položaj klijenta i njegova potražnja

Tablica 5.3: Heuristike

Indeks heuristike	Naziv heuristike
0	Simbolička regresija (GP/AP)
1	EF (earliest-first)
2	UF (urgent-first)
3	LC (linear-combination)
4	NN (nearest-neighbour)

6. Rezultati

U ovom poglavlju bit će opisani svi rezultati ovog rada. Najprije će biti riječi o parametrima koje je trebalo postaviti kako bi se omogućila maksimalna kvaliteta eksperimenta. Potom će biti prikazani rezultati prve komponente, odnosno, učenja heuristika na statičkim primjerima pomoću simboličke regresije. Na kraju će biti rezultati testiranja heuristika evoluiranih pomoću GP-a i AP-a, kao i jednostavnih heuristika na dinamičkim primjerima.

6.1. Definirani parametri

Pokretanje ekspermenta podrazumijeva i početne postavke koje je potrebno definirati prije samog pokretanja. Pogledajmo najprije kako izgleda funkcija cilja i dobrote koja se koristi tijekom učenja na statičkim i dinamičkim primjerima:

$$f(ind) = \sum_{instance} \big(n * vc + td\big) + \sum_{dynamicInstance} \big(rrc * rr\big) \tag{6.1}$$

Evaluacija svake jedinke funkcionira na način da se prolaskom kroz sve zadane instance zbroje doprinosi dobroti svake instance, a svaki taj doprinos računa se kao u formuli 6.1. n predstavlja broj ruta, vc predstavlja konstantu *vehicle cost* koja se zadaje u parametrima prije pokretanja, a td označava ukupnu prijeđenu udaljenost prijeđenu prema rješenju konstruiranom za određenu instancu. Doprinos kod dinamičkih instanci sastoji se od konstante rrc koja označava *rejected request cost*, odnosno trošak koji predstavlja jedan odbijeni zahtjev, a rr predstavlja broj odbijenih zahtjeva.

Početni skupovi terminala i početni skupovi parametara navedeni su u tablicama 6.1 i 6.2. Ovdje su statičke instance odabrane za učenje, a dinamičke za testiranje, kako bi se heuristika naučena na statičkim problemima kasnije mogla provjeriti na realnijim, dinamičkim problemima. Svi parametri i selekcija značajki gledaju se isključivo na primjerima za učenje, a problem na kojem su se sva izvođenja bazirala je VRPTW.

Popis instanci za učenje i testiranje za svaki problem nalazi se u tablici 6.3.

Tablica 6.1: Skup početnih terminala

Tip problema	Skup početnih terminala
CVRP	dist d rc drc ncc
VRPTW/VRPPD	dist d rc drc ncc t rt dd ttrt ttdd wt st tard vtd vhd ttnv

Tablica 6.2: Početni parametri za izvođenje

Parametar	Vrijednost parametra
veličina populacije	20
maksimalna dubina stabla	3
veličina turnira	7
F	0.9
CR	0.8
dimenzija polja realnih brojeva	30
vjerojatnost mutacije	0.5

Kako bismo znali koliki je potreban broj evaluacija potreban kako bismo na vrijeme zaustavili algoritam bez nepotrebnog trošenja resursa, potrebno je napraviti analizu konvergencije algoritma. Analiza konvergencije napravljena je samo za GP, no očekuje se da će konvergencija i za AP biti približno takva. Program je pokrenut samo jednom na početnom skupu instanci, a rezultati analize mogu se vidjeti na slici 6.1. Na rezultatima također vidimo i analizu na skupu za ispitivanje. Iz rezultata vidimo kako je prilikom izvođenja 20000 evaluacija najveća promjena dobrote zabilježena na početku izvođenja, odnosno u prvih 1000 evaluacija. Veće poboljšanje nakon toga se dešava nakon polovice izvođenja, no, s obzirom da to poboljšanje nije toliko veliko kao poboljšanje na početku, a na ispitnom skupu su najbolji rezultati zabilježeni upravo za malo broj evaluacija, sustav će se izvoditi na maksimalno 1500 evaluacija.

Prije optimizacije parametara vezanih za samu evoluciju napravljena je selekcija značajki (eng. feature selection). Selekcija značajki koristan je alat kojim se mogu ukloniti terminali za koje se ispostavi da ne doprinose poboljšanju rezultata evolucije. Postoji mnogo varijanti selekcije značajki, no ovdje će biti fokus na dvjema, a to su

Tablica 6.3: Korištene instance prilikom učenja i testiranja

Varijanta	Skup za učenje	Skup za testiranje
CVRP	C2_2_1	c101-0.0, c101-0.1, c101-0.5, c101-1.0
	R2_2_1	rc101-0.0, rc101-0.1, rc101-0.5, rc101-1.0
	RC1_2_1	rc102-0.0, rc102-0.1, rc102-0.5, rc102-1.0
VRPTW	C2_2_1	c101-0.0, c101-0.1, c101-0.5, c101-1.0
	R2_2_1	rc101-0.0, rc101-0.1, rc101-0.5, rc101-1.0
	RC1_2_1	rc102-0.0, rc102-0.1, rc102-0.5, rc102-1.0
VRPPD	lc101	lc101-0.0, lc101-0.1, lc101-0.5, lc101-1.0
	lr101	lr201-0.0, lr201-0.1, lr201-0.5, lr201-1.0
	lrc101	lrc201-0.0, lrc201-0.1, lrc201-0.5, lrc201-1.0
	lrc202	

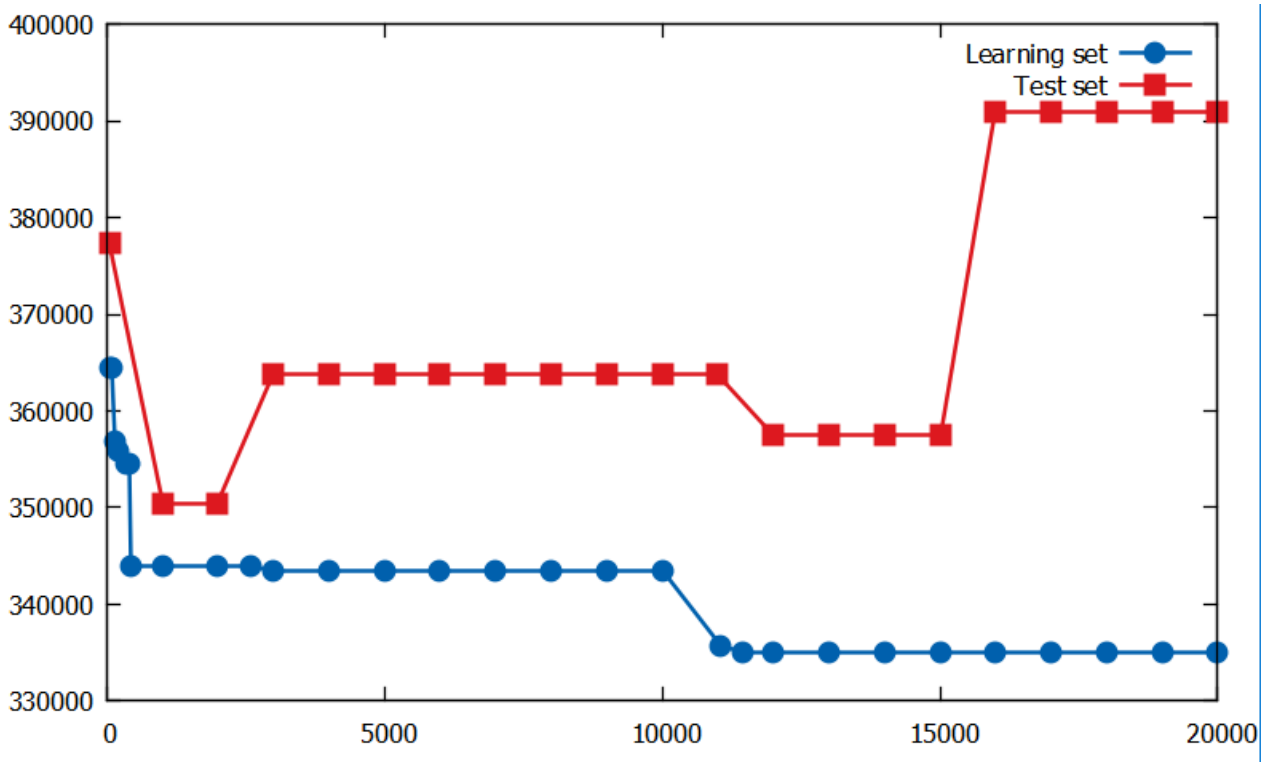
konstruktivna i destruktivna. Konstruktivna selekcija kreće od praznog skupa korištenih terminala i u svakoj iteraciji se uzimaju redom terminali iz skupa neiskorištenih terminala te se pokreće evolucija. Nakon što su svi terminali prošli evoluciju, odabire se najbolji, no on se priključuje skupu korištenih terminala samo ako poboljšava trenutni najbolji rezultat. Ako to nije slučaj, algoritam staje. Postoji i pohlepna varijanta algoritma koja odmah uzima prvi poboljšavajući terminal bez provjervanja ostalih terminala.

Destruktivna selekcija značajki kreće od punog skupa korištenih terminala i svaku iteraciju izbacuje onaj terminal čijim se izbacivanjem dobiva najbolji rezultat na skupu korištenih terminala. Za destruktivnu selekciju također postoji pohlepna varijanta. Ovdje je korištena pohlepna varijanta konstruktivne selekcije značajki, a terminali za svaku varijantu VRP-a navedeni su u tablici 6.4.

Tablica 6.4: Rezultati selekcije značajki

Tip problema	Skup korištenih terminala
CVRP	dist d rc drc
VRPTW/VRPPD	dist ncc d rc drc rt ttrt

Za CVRP i VRPTW nakon selekcije je pokrenuta analiza prije i poslije selekcije

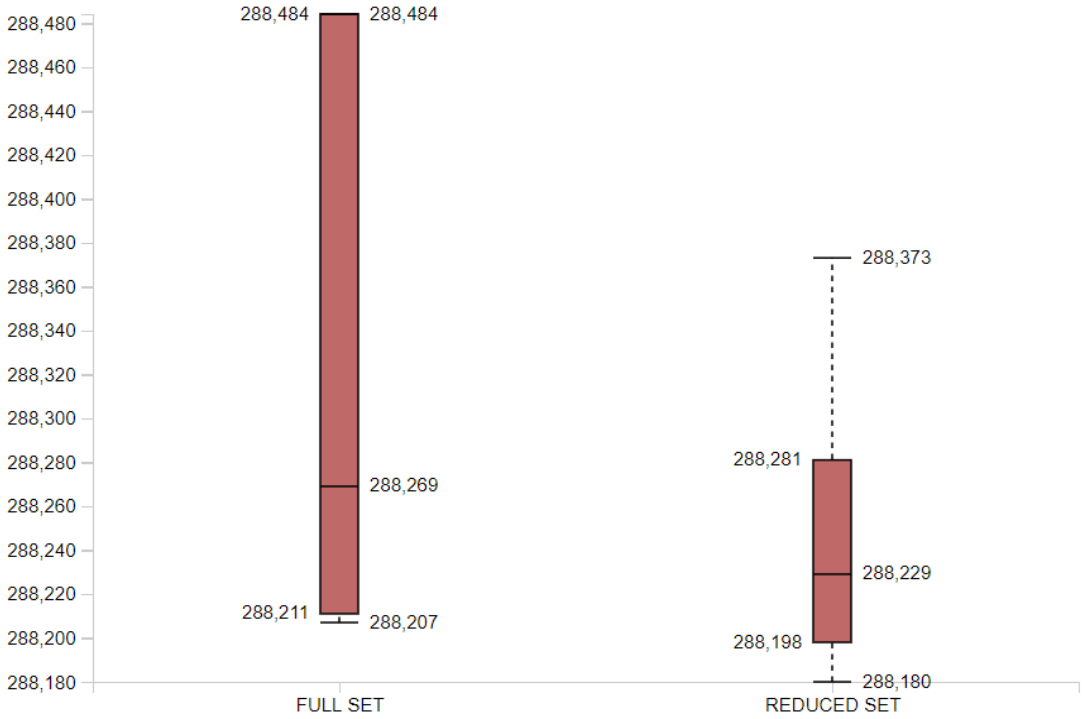


Slika 6.1: Analiza konvergencije za GP

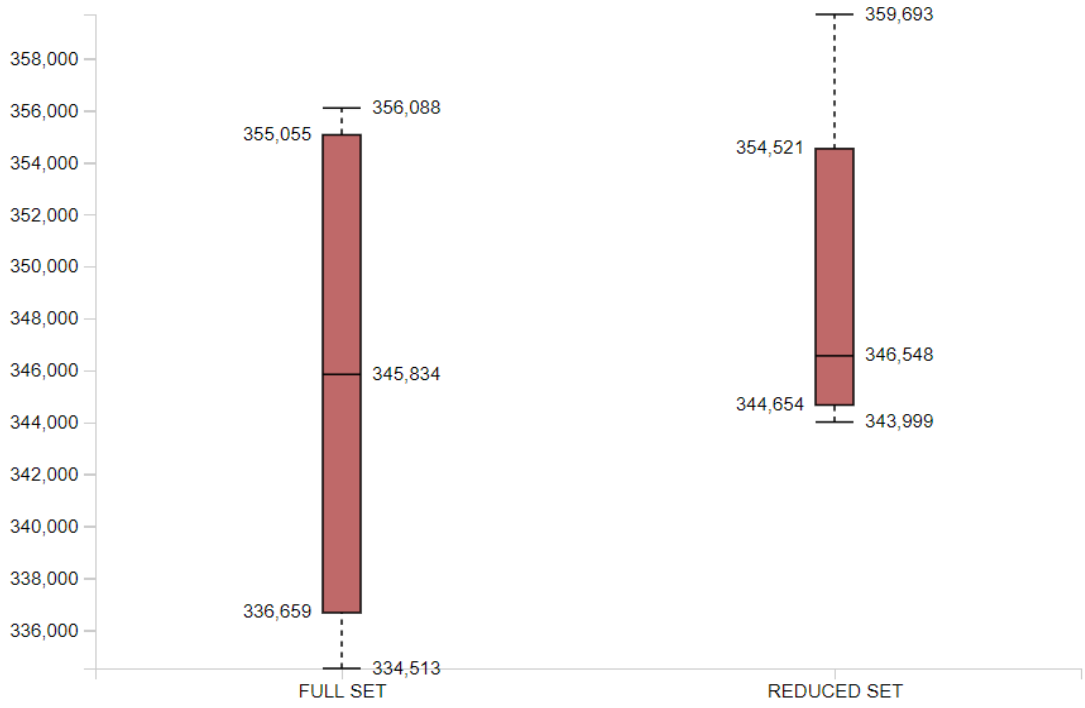
značajki 5 puta za svaku varijantu i za svaki tip skupa terminala. Ako pogledamo malo bolje analizu rezultata selekcije značajki za CVRP i VRPTW (slike 6.2 . 6.3), vidimo da je kod CVRP-a medijan vrijednosti dobrote nešto niži kod reduciranog skupa nego kod originalnog skupa, stoga vidimo da je ovdje dobro napravljena selekcija značajki. No, kod VRPTW-a, medijani kod oba skupa terminala su relativno isti, a za početni skup terminala je čak i nešto niži, što možemo reći i za minimalne vrijednosti kod početnog skupa. To možemo objasniti malim brojem pokretanja postavljen radi uštede resursa ili pak češćim nailaskom na terminale koji su se našli u odabranom skupu prilikom pokretanja na cijelom skupu terminala.

Nadalje, provedena je i optimizacija parametara specifičnih za evoluciju, odnosno, veličina populacije i vjerojatnost mutacije. Optimizacija parametara je provedena na način da se jedan parametar mijenja kroz nekoliko diskretnih unaprijed zadanih vrijednosti dok su ostali parametri fiksirani. Jednom kad se nađe najbolja vrijednost za određeni parametar, ta vrijednost se uzima i fiksira prije nastavka optimizacije drugih parametara. Isti postupak se ponavlja dok se ne prođu svi željeni parametri.

U ovom radu je za svaku vrijednost parametra pokrenut algoritam 5 puta. Najprije,

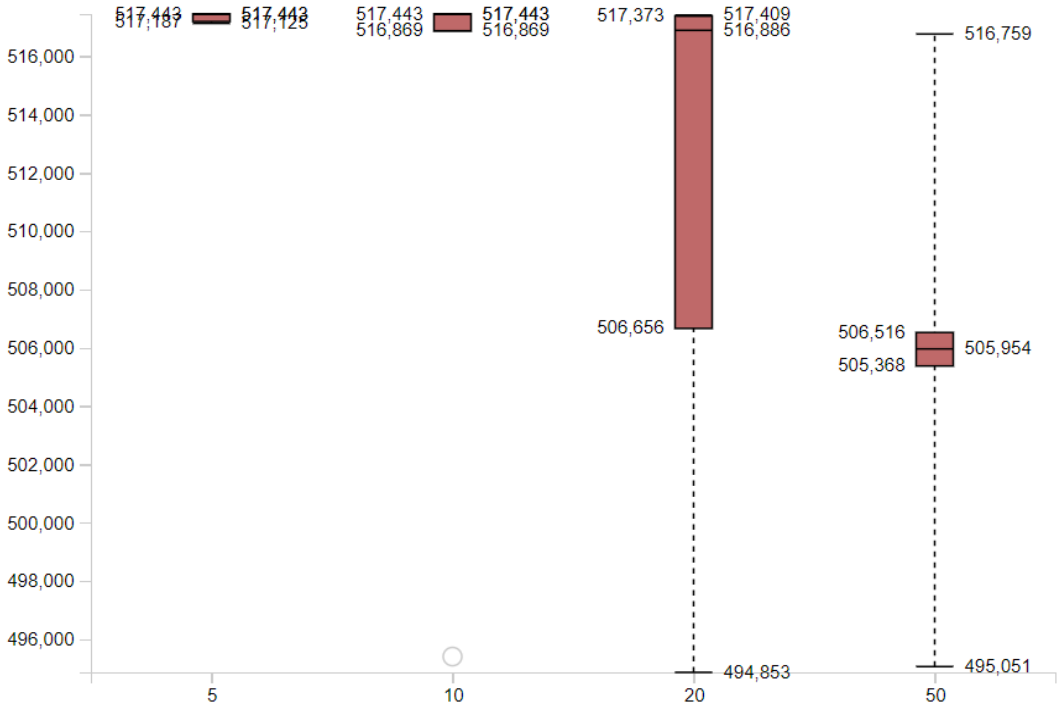


Slika 6.2: Rezultati selekcije značajki za CVRP



Slika 6.3: Rezultati selekcije značajki za VRPTW

imamo parametar veličine populacije. Za veličinu populacije prikazani su rezultati na slici 6.4. Sa grafa vidimo da je najočitiji izbor za veličinu populacije najveći ponuđeni broj - 50, budući da je njegov medijan daleko ispod svih ostalih veličina populacija, a i disperzija rezultata oko medijana je vrlo mala.

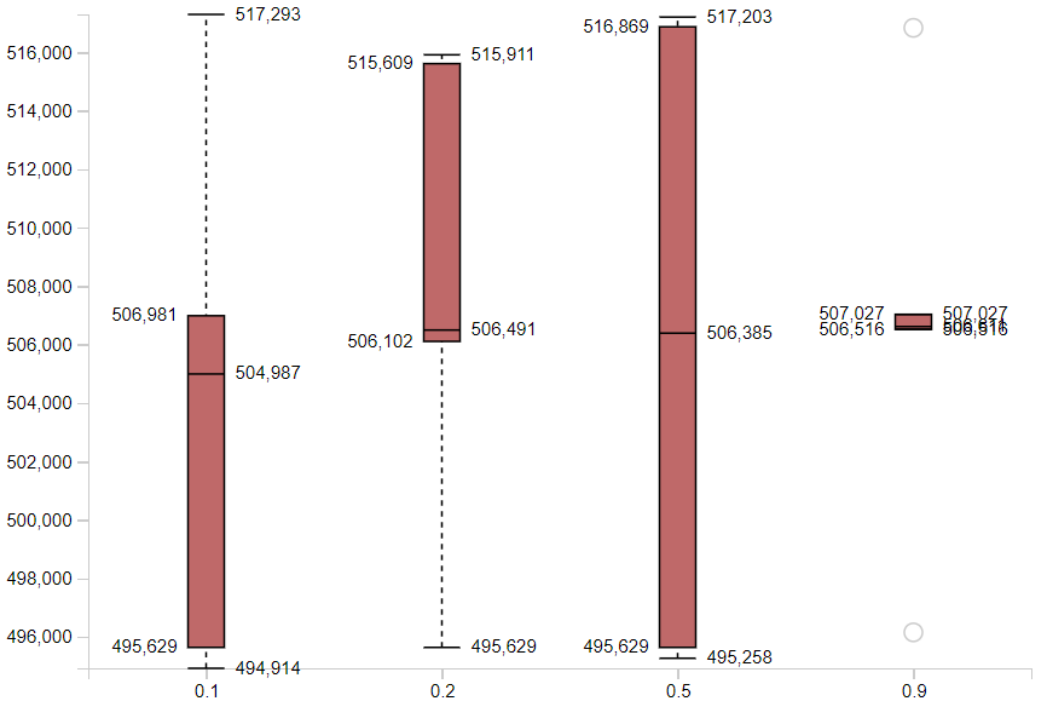


Slika 6.4: Rezultati algoritma za veličinu populacije

Konačno, imamo analizu rezultata za vjerojatnost mutacije prikazanu na slici 6.5 na kojoj vidimo vrlo šaroliko dobivene raspodjele rezultata za različite vjerojatnosti mutacije. Medijani su u svim raspodjelama na gotovo istoj razini, no ovdje se odlučujemo, ipak, za vjerojatnost mutacije 0.1 budući da je medijan ipak nešto niži od ostalih, a i velika većina rezultata je upravo ispod njegovog medijana.

Dakle, sada smo odabrali značajke koje ćemo koristiti u daljnjoj evoluciji, kao i optimalne parametre za evoluciju, a to su:

- broj generacija $\rightarrow 30$ (izračunat iz broja evaluacija)
- veličina populacije $\rightarrow 50$
- vjerojatnost mutacije $\rightarrow 0.1$



Slika 6.5: Rezultati algoritma za vjerojatnost mutacije

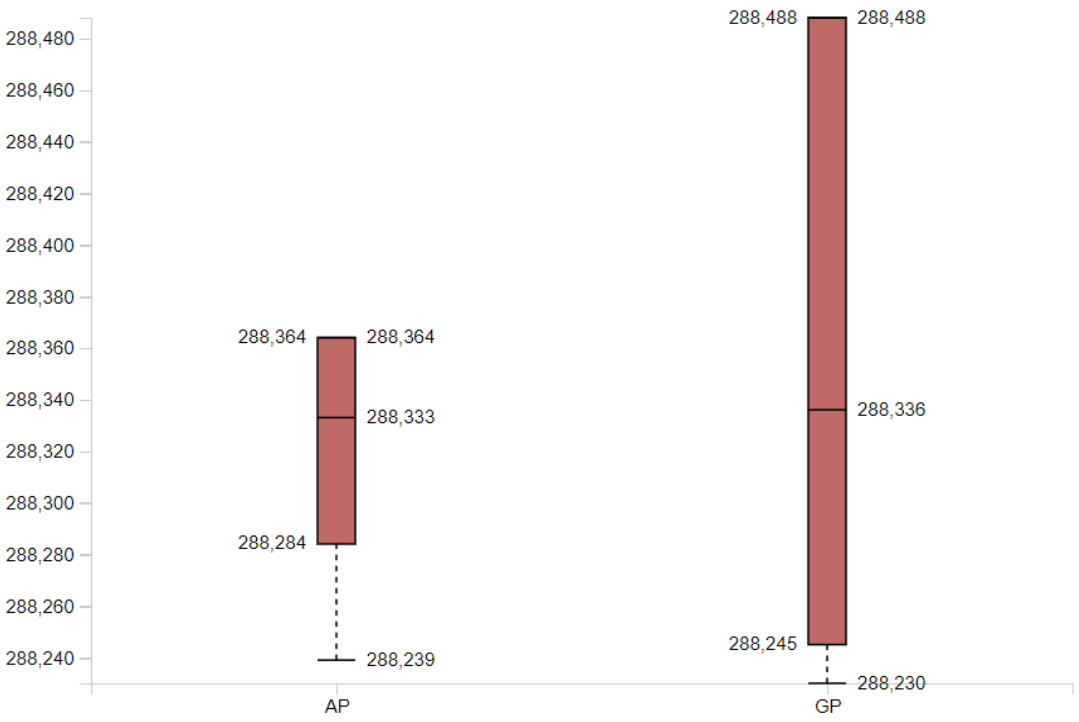
6.2. Učenje na statičkim primjerima

U prethodnom poglavlju izdvojili smo optimalne parametre i sada je potrebno iskoristiti te parametre kako bismo dobili što bolja rješenja za statičke varijante VRP-a koristeći genetičko te analitičko programiranje.

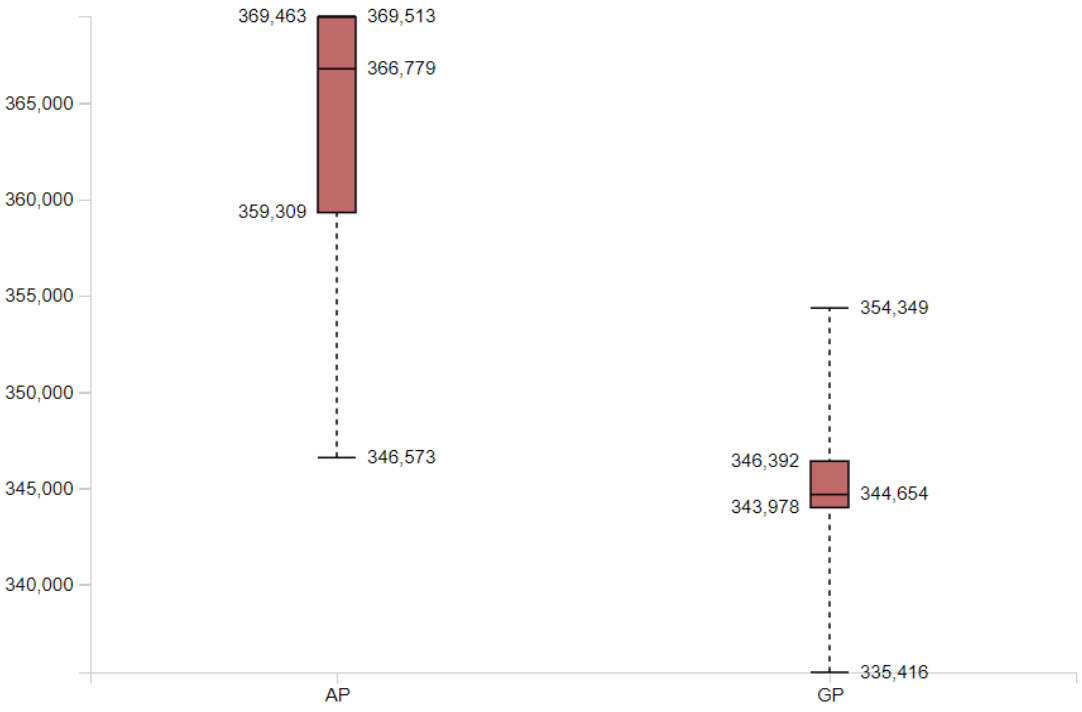
Pogledajmo najprije rezultate za CVRP. Na slici 6.6 možemo vidjeti da su medijani za oba algortima prilično izjednačeni, no kod GP-a su vrijednosti funkcije dobrote ipak malo raspršeniji, prema tome, ovdje je bolje prošao AP.

Na slici 6.7 vidimo rezultate izvođenja za VRPTW. Ovdje je bez sumnje jasno da je GP evoluirao puno bolje funkcije, s obzirom na to da su i medijan i maksimalna i minimalna vrijednost bolje od vrijednosti izgeneriranih od strane AP-a.

Zanimljivi su rezultati za VRPPD na slici 6.8. Naime, ovdje su i AP i GP za većinu pokretanja dobili približno jednake vrijednosti. Ipak, maksimum AP-a je unutar raspona oko medijana, dok je maksimum GP-a daleko izvan tog raspona, te, budući da su minimumi kod oba algoritma također vrlo slični, zaključujemo da je u ovom slučaju



Slika 6.6: Rezultati simboličke regresije za CVRP



Slika 6.7: Rezultati simboličke regresije za VRPTW

AP opet nešto bolji od GP-a.

Slika 6.8: Rezultati simboličke regresije za VRPPD

U tablici 6.5 možemo vidjeti rezultate jednostavnih heuristika za sve statičke instance u svim varijantama VRP-a. Brojke u tablici označavaju ukupan prijeđeni put koji su prošla sva vozila kako bi poslužila klijente. Za CVRP pokrenuta je samo heuristika NN, dok su za VRPTW i VRPPD pokrenute sve 4 heuristike. Za instance i heuristiku u kojoj nisu posluženi svi klijenti dodana je oznaka (N).

6.3. Ispitivanje na dinamičkim primjerima

Sada, kada imamo rezultate i programe simboličke regresije naučene na statičkim primjerima, prostalo je samo testirati sve dobivene jedinke na dinamičkim instancama. Pored rezultata metoda simboličke regresije, bit će navedeni i rezultati jednostavnih heuristika da usporedimo mjeru poboljšanja u odnosu na ručno izgrađene heuristike.

Za svaku instancu najprije je određen početni broj vozila tako da je riješen problem serijski na zahtjevima koji su na početku poznati, a zatim su tom broju vozila pridodana

Tablica 6.5: Rezultati jednostavnih heuristika na statičkim primjerima

Instanca	EF	UF	LC	NN
C2_2_1	2287.15	2745.81	2404.13	2091.27 (CVRP) 6544.65 (VRPTW)
R2_2_1	6434.78	9496.84	7576.82	2269.51 (CVRP) 7151.76 (VRPTW)
RC1_2_1	6333.32	6360.53	5874.35	4126.93 (CVRP) 6607.64 (VRPTW)
lc101	828.937	2047.56	828.937	1222.86
lr101	1900.8	1935.83	2155.49	1640.81
lrc101	2403.78	1471.12 (N)	2222.9	2237.2
lrc202	2348.82	4099	3271.67	2802

još dva vozila kao priprema za nadolazeća vozila. Sjetimo se, tijekom simulacije će pristizati zahtjevi, a svaki zahtjev će biti prihvaćen samo ako se uz taj novi zahtjev može generirati valjano rješenje, inače se zahtjev odbija i simulacija ide dalje bez tog zahtjeva.

Pogledajmo rezultate za sve dinamičke varijante VRP-a. Za CVRP od jednostavnih heuristika korištena je samo heuristika NN (*nearest-neighbour*), budući da su sve ostale heuristike vremenski ovisne, a u CVRP-u vremensku komponentu ignoriramo. Za sve ostale varijante koristimo sve heuristike.

Brojke koje će biti navedene u nastavku prikazivat će broj odbijenih zahtjeva za neku instancu i korištenu heuristiku. Za svaku vrstu problema navedeni su rezultati za odabrane instance za različite stupnjeve dinamičnosti u kontekstu promjena podatak unutar instance. Stoga će za svaku varijantu VRP-a biti 4 tablice koje će predstavljati rezultate uz postavljene parametre promjena unutar instance na 0, 0.2, 0.5 i 0.9. Rezultati za dinamički CVRP prikazani su u tablicama 6.6, 6.7, 6.8 te 6.9, rezultati za dinamički VRPTW u tablicama 6.10, 6.11, 6.12 te 6.13, a rezultati za dinamički VRPPD u tablicama 6.14, 6.15, 6.16 te 6.17. U tablicama 6.18, 6.19 i 6.20 navedeni su rezultati sa parametrom dinamičnosti 0 za sve varijante koji prikazuju ukupan prijeđeni put.

Tablica 6.6: Rezultati dinamičke simulacije za CVRP sa parametrom dinamičnosti promjene instanci 0

Instanca	NN	GP	AP
c101-0.0	0	0	0
c101-0.1	0	0	0
c101-0.5	23	22.8	24.2
c101-1.0	63	63	63.2
rc101-0.0	0	0	0
rc101-0.1	0	0	0
rc101-0.5	9	8.6	9.6
rc101-1.0	42	42.4	42.2
rc102-0.0	0	0	0
rc102-0.1	0	0	0
rc102-0.5	3	3	4.6
rc102-1.0	31	30.8	31

Tablica 6.7: Rezultati dinamičke simulacije za CVRP sa parametrom dinamičnosti promjene instanci 0.2

Instanca	NN	GP	AP
c101-0.0	0	0	0
c101-0.1	0	0	0
c101-0.5	92	67.6	50.2
c101-1.0	125	114.6	102.2
rc101-0.0	0	0	0
rc101-0.1	0	0	0
rc101-0.5	22	17.4	14.2
rc101-1.0	39	42.6	47
rc102-0.0	0	0	0
rc102-0.1	0	0	0
rc102-0.5	3	5.8	5.8
rc102-1.0	35	37	35

Tablica 6.8: Rezultati dinamičke simulacije za CVRP sa parametrom dinamičnosti promjene instanci 0.5

Instanca	NN	GP	AP
c101-0.0	0	0	0
c101-0.1	0	0	0
c101-0.5	139	132.6	62
c101-1.0	210	208	166
rc101-0.0	0	0	0
rc101-0.1	0	0	0
rc101-0.5	37	34.2	17
rc101-1.0	52	48	51.4
rc102-0.0	0	0	0
rc102-0.1	0	0	0
rc102-0.5	3	4.8	6.2
rc102-1.0	37	42.8	43

Tablica 6.9: Rezultati dinamičke simulacije za CVRP sa parametrom dinamičnosti promjene instanci 0.9

Instanca	NN	GP	AP
c101-0.0	0	0	0
c101-0.1	0	0	1.8
c101-0.5	126	107.8	84
c101-1.0	335	310.8	202.4
rc101-0.0	0	0	0
rc101-0.1	0	0	0
rc101-0.5	6	10	25.2
rc101-1.0	81	70.8	58.4
rc102-0.0	0	0	0
rc102-0.1	0	0	0
rc102-0.5	21	12.6	9.6
rc102-1.0	56	58.2	54.8

Tablica 6.10: Rezultati dinamičke simulacije za VRPTW sa parametrom dinamičnosti promjene instanci 0

Instanca	EF	UF	LC	NN	GP	AP
c101-0.0	0	0	0	0	0	0
c101-0.1	2	0	0	8	8	3.2
c101-0.5	0	2	0	46	46	11.4
c101-1.0	2	1	2	32	32	2.4
rc101-0.0	0	0	0	0	0	0
rc101-0.1	10	10	10	10	10	10
rc101-0.5	20	21	37	37	37	22.4
rc101-1.0	28	34	30	42	42	34
rc102-0.0	0	0	0	0	0	0
rc102-0.1	6	5	1	6	6	6
rc102-0.5	13	28	28	28	28	28
rc102-1.0	55	46	55	55	55	55

Tablica 6.11: Rezultati dinamičke simulacije za VRPTW sa parametrom dinamičnosti promjene instanci 0.2

Instanca	EF	UF	LC	NN	GP	AP
c101-0.0	0	0	0	0	0	0
c101-0.1	21	0	0	133	133	106.4
c101-0.5	0	1	0	203	203	164.6
c101-1.0	16	5	7	64	51.6	47
rc101-0.0	0	0	0	0	0	0
rc101-0.1	36	36	36	36	36	29.8
rc101-0.5	65	22	11	65	65	57.8
rc101-1.0	52	55	52	49	49.2	59
rc102-0.0	0	0	0	0	0	0
rc102-0.1	40	28	9	40	40	33.2
rc102-0.5	17	53	53	53	53	53
rc102-1.0	37	66	73	73	66	69

Tablica 6.12: Rezultati dinamičke simulacije za VRPTW sa parametrom dinamičnosti promjene instanci 0.5

Instanca	EF	UF	LC	NN	GP	AP
c101-0.0	0	0	0	0	0	0
c101-0.1	80	0	83	273	226.8	64
c101-0.5	94	211	14	384	332.4	157.4
c101-1.0	38	3	1	63	16.8	3.4
rc101-0.0	0	0	0	0	0	0
rc101-0.1	54	54	54	54	54	54
rc101-0.5	88	56	11	88	77.6	47.6
rc101-1.0	57	60	57	41	39.6	54.4
rc102-0.0	0	0	0	0	0	0
rc102-0.1	13	37	7	70	62.2	60
rc102-0.5	27	75	75	75	52	75
rc102-1.0	96	59	96	96	96	96

Tablica 6.13: Rezultati dinamičke simulacije za VRPTW sa parametrom dinamičnosti promjene instanci 0.9

Instanca	EF	UF	LC	NN	GP	AP
c101-0.0	0	0	0	0	0	0
c101-0.1	137	0	119	499	482.8	0
c101-0.5	156	103	143	654	571	263.8
c101-1.0	3	0	55	174	65	231.2
rc101-0.0	0	0	0	0	0	0
rc101-0.1	104	104	104	104	104	104
rc101-0.5	134	93	38	134	122.2	87.4
rc101-1.0	61	65	80	104	56.6	61.8
rc102-0.0	0	0	0	0	0	0
rc102-0.1	128	87	77	128	122	128
rc102-0.5	66	120	120	120	97	120
rc102-1.0	126	65	126	126	126	126

Tablica 6.14: Rezultati dinamičke simulacije za VRPPD sa parametrom dinamičnosti promjene instanci 0

Instanca	EF	UF	LC	NN	GP	AP
lc101-0.0	0	0	0	0	0	0
lc101-0.1	8	8	0	8	1.6	4.8
lc101-0.5	28	28	28	28	28	28
lc101-1.0	10	24	4	8	6.8	5.2
lr201-0.0	0	0	0	0	0	0
lr201-0.1	8	16	12	16	13.2	12.8
lr201-0.5	44	44	44	44	44	44
lr201-1.0	68	68	68	68	68	68
lrc201-0.0	0	0	0	0	0	0
lrc201-0.1	16	10	16	16	15.2	16
lrc201-0.5	52	52	52	52	52	52
lrc201-1.0	68	68	68	68	68	68

Tablica 6.15: Rezultati dinamičke simulacije za VRPPD sa parametrom dinamičnosti promjene instanci 0.2

Instanca	EF	UF	LC	NN	GP	AP
lc101-0.0	0	0	0	0	0	0
lc101-0.1	17	17	12	17	11	15
lc101-0.5	50	50	50	50	50	50
lc101-1.0	20	18	16	33	14.4	14
lr201-0.0	0	0	0	0	0	0
lr201-0.1	23	27	19	28	20.2	26.8
lr201-0.5	63	63	63	63	63	63
lr201-1.0	83	83	83	83	83	83
lrc201-0.0	0	0	0	0	0	0
lrc201-0.1	28	29	29	29	26	27.4
lrc201-0.5	69	69	69	69	69	69
lrc201-1.0	82	82	82	82	82	82

Tablica 6.16: Rezultati dinamičke simulacije za VRPPD sa parametrom dinamičnosti promjene instanci 0.5

Instanca	EF	UF	LC	NN	GP	AP
lc101-0.0	0	0	0	0	0	0
lc101-0.1	27	27	22	27	20.6	25
lc101-0.5	63	63	63	63	63	63
lc101-1.0	10	28	8	54	20.4	33.2
lr201-0.0	0	0	0	0	0	0
lr201-0.1	39	41	42	42	32.4	35.6
lr201-0.5	68	68	68	68	68	68
lr201-1.0	87	87	87	87	87	87
lrc201-0.0	0	0	0	0	0	0
lrc201-0.1	40	44	44	44	40.6	42.6
lrc201-0.5	74	74	74	74	74	74
lrc201-1.0	90	90	90	90	90	90

Tablica 6.17: Rezultati dinamičke simulacije za VRPPD sa parametrom dinamičnosti promjene instanci 0.9

Instanca	EF	UF	LC	NN	GP	AP
lc101-0.0	0	0	0	0	0	0
lc101-0.1	51	51	32	51	38.4	46
lc101-0.5	109	109	109	109	109	109
lc101-1.0	75	80	75	78	71.4	55.8
lr201-0.0	0	0	0	0	0	0
lr201-0.1	69	71	54	72	59.8	57.6
lr201-0.5	108	108	108	108	108	108
lr201-1.0	120	120	120	120	120	120
lrc201-0.0	0	0	0	0	0	0
lrc201-0.1	79	82	82	82	77.2	78.8
lrc201-0.5	114	114	114	114	114	114
lrc201-1.0	122	122	122	122	122	122

Tablica 6.18: Rezultati za ukupan prijeđeni put dinamičke simulacije za CVRP sa parametrom dinamičnosti promjene instanci 0

Instanca	NN	GP	AP
c101-0.0	2128.17	2006.786	2070.204
c101-0.1	1798.11	1789.468	1955.968
c101-0.5	1357.02	1297.302	1334.174
c101-1.0	534.626	547.272	583.796
rc101-0.0	1465.96	1532.83	1861.18
rc101-0.1	1707.81	1677.542	2058.692
rc101-0.5	1277.8	1329.52	1510.984
rc101-1.0	732.494	681.206	906.1702
rc102-0.0	1465.96	1532.83	1861.18
rc102-0.1	1731.06	1671.786	1989.964
rc102-0.5	1428.02	1370.296	1671.564
rc102-1.0	870.848	882.788	1163.367

Tablica 6.19: Rezultati za ukupan prijeđeni put dinamičke simulacije za VRPTW sa parametrom dinamičnosti promjene instanci 0

Instanca	EF	UF	LC	NN	GP	AP
c101-0.0	2217.87	3721.37	2016.31	1232.54	1232.54	3245.772
c101-0.1	2315.52	3792.28	2008.49	1240.19	1240.19	3257.59
c101-0.5	2276.81	2900.55	2195.16	1178.08	1178.08	2680.592
c101-1.0	2890.78	3171.8	2461.47	1093.8	1093.8	3123.302
rc101-0.0	2466.52	3078.06	2386.2	1691.83	1691.83	2887.808
rc101-0.1	2207.8	2912.91	2096.43	1760.61	1760.61	2695.554
rc101-0.5	1616.45	1924.55	1442	1343.66	1343.66	1886.302
rc101-1.0	1150.93	1204.67	1121.71	1081.78	1081.78	1191.258
rc102-0.0	2380.69	3418.52	2546.23	1938.38	1938.38	3366.484
rc102-0.1	2246.22	3088.99	2283.81	1896.68	1896.68	3064.614
rc102-0.5	1847.25	2177.65	1750.79	1586.73	1586.73	2115.688
rc102-1.0	983.568	1556.97	956.021	983.568	983.568	1322.858

Tablica 6.20: Rezultati za ukupan prijeđeni put dinamičke simulacije za VRPPD sa parametrom dinamičnosti promjene instanci 0

Instanca	EF	UF	LC	NN	GP	AP
lc101-0.0	828.937	2047.56	828.937	1222.86	875.2296	1701.645
lc101-0.1	1699.98	1743.32	891.874	1299.19	1004.498	1709.572
lc101-0.5	1054.1	1473.05	1016.13	876.917	1077.167	1192.984
lc101-1.0	970.996	954.753	1122.29	935.132	931.997	1228.589
lr201-0.0	2280.42	2935.15	2353.41	1567.97	2401.612	2724.578
lr201-0.1	2497.54	2700.38	2396.05	2065.75	2157.044	2413.006
lr201-0.5	1142.41	1352.66	1280.05	1182.11	1205.742	1297.04
lr201-1.0	105.912	105.912	105.912	105.912	105.912	105.912
lrc201-0.0	2578.97	3886.92	3098.81	2773.46	3037.232	3514.03
lrc201-0.1	2504.47	3077.2	2724.43	2485.24	2428.54	2925.024
lrc201-0.5	1439.96	1760.36	1578.15	1408.97	1519.232	1599.518
lrc201-1.0	187.098	227.432	187.098	305.656	206.788	206.788

7. Zaključak

Kroz zadnjih nekoliko desetljeća istraživanje rješenja problema VRP-a poprimilo je velike razmjere te su isprobane brojne metode za rješavanje statičke i dinamičke varijante tog problema. Budući da se već na mnogim problemima simbolička regresija pokazala kao vrlo učinkovita metoda rješavanja složenih optimizacijskih problema, koristila se i u ovom radu, no, uz već široko korišten GP, isprobana je i njegova modernija varijanta sa genotipom polja realnih brojeva AP.

Prilikom učenja na statičkim primjerima, GP i AP pokazali su otprilike slične performanse, a u svakoj varijanti je neki od njih bio nešto bolji. Ipak, na ispitnom dinamičkom skupu instanci pokazalo se da je GP u većini slučajeva bolji ili približan jednostavnim heuristikama s kojima smo uspoređivali performanse, dok je AP bio kod mjere odbijenih zahtjeva otprilike na razini GP-a, dok je kod mjere ukupne udaljenosti u pravilu bio uvijek gori od GP-a i većine jednostavnih heuristika, što se može pripisati premalom broju evaluacija te nedovljnoj optimizaciji parametara u slučaju AP-a.

Trenutna rješenja bi se mogla poboljšati dodavanjem nekih novih terminala, boljom kontrolom konstruiranja rješenja kako bi se u što većem broju slučajeva mogli poslužiti svi klijenti, opsežnijom optimizacijom parametara te učenjem i na dinamičkim primjerima, a ne samo statičkim. Ipak, i sa ovakvim algoritmima dobiveni su solidni rezultati, usporedivi sa jednostavnim heuristikama te u nekim slučajevima čak i bolji, a to je dobra početna pozicija za daljnji razvoj algoritama te optimizaciju koja bi dovela trenutne algoritme do još boljih rješenja nego sada.

LITERATURA

[1] AbdelMonaem Fouad Moustafa AbdAllah. *Using Metaheuristics to solve Dynamic Vehicle Routing Problems*. Doktorska disertacija, UNSW)@ Canberra, 2013.

[2] Russell W Bent i Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6): 977–987, 2004.

[3] 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.

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

[5] Hermann Gehring i Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. U *Proceedings of EUROGEN99*, svezak 2, stranice 57–64. Citeseer, 1999.

[6] Hideki Hashimoto, Toshihide Ibaraki, Shinji Imahori, i Mutsunori Yagiura. The vehicle routing problem with flexible time windows and traveling times. *Discrete Applied Mathematics*, 154(16):2271–2290, 2006.

[7] Penny Louise Holborn. *Heuristics for dynamic vehicle routing problems with pickups and deliveries and time windows*. Doktorska disertacija, Cardiff University, 2013.

[8] Josiah Jacobsen-Grocott, Yi Mei, Gang Chen, i Mengjie Zhang. Evolving heuristics for dynamic vehicle routing with time windows using genetic programming. U *2017 IEEE Congress on Evolutionary Computation (CEC)*, stranice 1948–1955. IEEE, 2017.

[9] John R Koza i John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, svezak 1. MIT press, 1992.

[10] Haibing Li i Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(02): 173–186, 2003.

[11] Raluca Necula, Mihaela Breaban, i Madalina Raschip. Tackling dynamic vehicle routing problem with time windows by means of ant colony system. U *2017 IEEE Congress on Evolutionary Computation (CEC)*, stranice 2480–2487. IEEE, 2017.

[12] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4): 421–451, 1993.

[13] Ted K Ralphs, Leonid Kopman, William R Pulleyblank, i Leslie E Trotter. On the capacitated vehicle routing problem. *Mathematical programming*, 94(2-3): 343–359, 2003.

[14] Ryoji Tanabe i Alex Fukunaga. Success-history based parameter adaptation for differential evolution. U *2013 IEEE congress on evolutionary computation*, stranice 71–78. IEEE, 2013.

[15] Niaz A Wassan i Gábor Nagy. Vehicle routing problem with deliveries and pickups: Modelling issues and meta-heuristics solution approaches. *International Journal of Transportation*, 2(1):95–110, 2014.

[16] Haitao Xu, Pan Pu, i Feng Duan. Dynamic vehicle routing problems with enhanced ant colony optimization. *Discrete Dynamics in Nature and Society*, 2018, 2018.

[17] Ivan Zelinka, Zuzana Oplatkova, i Lars Nolle. Analytic programming–symbolic regression by means of arbitrary evolutionary algorithms. *Int. J. of Simulation, Systems, Science and Technology*, 6(9):44–56, 2005.

Uporaba simboličke regresije za rješavanje problema usmjeravanja vozila

Sažetak

U ovom radu glavno je pitanje kako riješiti statičku i dinamičku varijantu problema koristeći metode simboličke regresije. Opisan je temeljni sadržaj problema usmjeravanja vozila kao i njegove varijante koje su se rješavale u radu, a to su CVRP (kaacitivni problem usmjeravanja vozila), VRPTW (problem usmjeravanja vozila s vremenskim prozorima) te VRPPD (problem usmjeravanja vozila s prikupljanjima i dostavama). Potom je napravljen detaljan uvid u neke od najpoznatijih radova koji su doprinijeli riješavanju obiju varijanti problema, a zatim su opisane metode koje se koriste u ovom radu, a to su genetičko programiranje i analitičke programiranje. Slijede detalji implementacije sustava za rješavanje VRP poput korištenih instanci, pseudokoda algoritama te korištenja razvijenog sustava. Na kraju su prikazani rezultati za statičke i dinamičke varijante te su metode simboličke regresije uspoređene s jednostavnim heuristikama.

Ključne riječi: VRP, simbolička regresija, genetičko programiranje, analitičko programiranje, hiperheuristike, dinamička simulacija

Solving vehicle routing problems with symbolic regression

Abstract

In this paper, the main question is how to solve the static and dynamic variants of the problem using symbolic regression methods. The basic contents of the vehicle routing problem as well as its variants that have been dealt with in the work are described, namely CVRP (Capacitated Vehicle Routing Problem), VRPTW (Vehicle Routing Problem with Time Windows) and VRPPD (Vehicle Routing Problem with Pickup and Delivery). Then a detailed insight into some of the most famous papers contributed to the solution of both variants of the problem and then the methods used in this paper were described, namely genetic programming and analytical programming. Following are the details of deploying VRP solution systems such as instances used, pseudocode of used algorithms, and the usage of developed system. Finally, the results for static and dynamic variants are presented, and the methods of symbolic regression are compared with simple heuristics.

Keywords: VRP, symbolic regression, genetic programming, analytical programming, hyperheuristics, dynamic simulation