SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 137

Rješavanje problema usmjeravanja vozila u pomorskim lukama

Luka Matijević

Zagreb, srpanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 137

Pristupnik:	**Luka Matijević (0036517220)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Rješavanje problema usmjeravanja vozila u pomorskim lukama**

Opis zadatka:

Proučiti problem usmjeravanja vozila u pomorskim lukama. Istražiti postupke korištene u literaturi za rješavanje danog problema. Pronaći odgovarajuće primjerke problema koji će biti korišteni za ispitivanje. Osmisliti i implementirati metode koje su prilagođene za rješavanje problema usmjeravanja vozila u pomorskim lukama temeljene na metodama dostupnima u literaturi. Primijeniti implementirane metode na odabrane primjerke problema te ocijeniti i analizirati njihovu uspješnost za razmatrani problem. Predložiti prilagodbe i poboljšanja algoritama s ciljem poboljšanja dobivenih rezultata. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 23. lipnja 2023.

Hvala mojem mentoru, doc. dr. sc. Marku Đuraseviću, na brojnim zanimljivim predavanjima, usmjeravanju tijekom studija i na velikoj pomoći pri izradi ovog diplomskog rada.

SADRŽAJ

1. Uvod

Problem usmjeravanja vozila interesantna je tema istraživanja zbog rastućeg utjecaja na stvarne procese u industriji. Dobro isplaniran skup ruta vozila može uvelike smanjiti ukupan trošak i resursne zahtjeve sektora koji se bavi nekom vrstom prijevoza. Jednostavni ciljevi optimizacije kao što su minimizacija udaljenosti prijevoza robe ili smanjenje potrošnje goriva vožnjom po bržim cestama mogu napraviti veliku razliku i rezultirati iznimno kvalitetnim sustavima. Budući da često to nije dovoljno, naprednije metode imaju i dodatne komponente koje uzimaju u obzir. Neke od njih su kvalitetno balansiranje količine posla između svih vozila, usklađivanje susjednih smjena, izbjegavanje prometnih gužvi i dinamičko ažuriranje putanja vozila.
Ciljevi optimizacije ovise o konkretnom problemu i potrebama klijenata te postoji veliki broj različitih ciljeva koje je nerijetko potrebno uskladiti. Osim velikih financijskih prednosti, dobrim planiranjem ruta i smjena vozača može se poboljšati sigurnost vozača, ali i njihovo zadovoljstvo na radnom mjestu, smanjiti broj vozila i samim time utjecaj na okoliš. Stoga tvrtke čiji se određen udio poslovanja sastoji od prijevoza robe imaju veliki interes za navedenu temu kako bi iskoristile navedene prednosti optimizacije.
Konkretan primjer jest Ningbo luka na obali Istočnog Kineskog mora koja ima status najveće luke na svijetu što se tiče prevezenog tereta. U 2021. godini kroz luku je prevezeno 1,224 milijardi tona tereta [1]. Ta brojka predstavlja veliki logistički izazov jer tolika količina tereta povlači sa sobom potrebu izvrsne organizacije i planiranja prijevoza robe. Osim samog planiranja ruta, potrebno je dobro uskladiti sve povezane procese (priprema i održavanje vozila, dojava promjena u prometu, pregled lokacija vozila,...). U isto vrijeme ukazuje se velika prilika za uštedom budući da mala poboljšanja sustava mogu dovesti do značajnih smanjenja troškova u globalu. U takvim slučajevima važno je pokušati minimizirati potrošene resurse za prijevoz robe.

Pomorske luke posebno su zanimljive budući da se cijeli sustav sastoji od nekoliko vrsta vozila koja je potrebno uskladiti. Brodovi prevoze robu preko mora i dovoze ju u luku, kamioni zatim raznose tu robu u okolne gradove te se na kraju manja vozila, kao što su dizalice i viličari uključuju i slažu te iste kontejnere u samim skladištima. U navedenom procesu

značajnu ulogu imaju vremenski uvjeti zbog prijevoza robe preko mora. Iako su brodovi koji prevoze takve kontejnere iznimno veliki, vremenske nepogode jakog intenziteta mogu ograničiti njihove mogućnosti prijevoza ili pak oštetiti robu. Neke od nepogoda čestih za morsko područje su tsunamiji uzrokovani potresima i tajfuni [2]. Taj aspekt prijevoza robe također je važno uzeti u obzir budući da može doći do kašnjenja dolaska robe u luku, no ovaj rad neće ga detaljnije razmatrati.

Sve navedeno poziva na razvoj sustava čiji je cilj usmjeravanje vozila na optimalan način. Opisani problem je dobro poznati problem usmjeravanja vozila (engl. *Vehicle Routing Problem - VRP*). Taj problem istražuje se više od 60 godina, od kada su ga prvi put u svom radu spomenuli Dantzig i Ramser [3]. Tijekom godina istraživanja, VRP je uspješno integriran s ostalim tehnologijama kako bi se razvili takozvani *pametni gradovi* i *pametne luke* [4]. Ideja iza takvih projekata jest razviti iznimno efikasne sustave koji nastoje optimizirati, između ostalog, prijevoz ljudi i robe s ciljem povećanja zadovoljstva i sigurnosti građana ili zaposlenika. Takvi sustavi koriste nekoliko različitih tehnologija kako bi prikupili što više podataka; GPS, radarska i senzorska tehnologija, 5G mreže i povezane tehnologije interneta stvari. Kasnije se metodama analize velikih skupova podataka ti podaci pretvaraju u korisne informacije pomoću kojih se procjenjuju potrebe prijevoza. Dobar algoritam usmjeravanja vozila čini jezgru takvog sustava i iznimno je bitan za dobar završni proizvod. U tom smislu postoje različite varijante VRP-a, ovisno o konkretnim ograničenjima koje algoritam mora uzeti u obzir. U ovom konkretnom slučaju radi se o problemu usmjeravanja vozila s vremenskim prozorima (engl. *VRP with Time Windows - VRPTW*). Mreža svake pomorske luke sastoji se od nekoliko okolnih čvorova (gradova) između kojih je potrebno obavljati prijevoz kontejnera. Taj prijevoz od početne lokacije do odredišta jest konstruiran kao jedan zadatak koji ima početno i završno vrijeme unutar kojeg je potrebno prevesti navedenu robu.

Ovaj rad detaljno će pregledati postojeću literaturu za navedeni problem prijevoza robe i predložiti prikladan pristup rješavanja problema kojem je cilj generirati kvalitetan skup ruta vozila za navedene potrebe. Provest će se eksperimentalna testiranja te proučiti rezultati predloženog rješenja. Na kraju rada izvest će se zaključak te prijedlog za daljnji razvoj pristupa.

2. Pregled literature

VRP je od velikog interesa zbog značajnih prednosti koje nudi implementacija kvalitetnog sustava. Velika prilika za uštedom resursa privlači mnoge istraživače i vodeće ljude industrije za istraživanje i pokušaj poboljšanja postojećih metoda. To navodi da postoji veliki broj istraživanja koja se bave ovom temom što i jest slučaj. Naravno, postoje različiti pristupi jer postoje drugačije varijante ovog problema, odnosno postoje razni ciljevi i ograničenja optimizacije što dovodi do velikog broja metoda i algoritama. Budući da je jako teško (gotovo nemoguće) u razumnom vremenu pronaći optimalno rješenje za najteže varijante problema ovoga tipa, izazov je procijeniti koja rješenja su *dobra*. Budući da je to riječ koja može imati subjektivno značenje, pogotovo u ovom kontekstu, važno je znati uspoređivati različite pristupe i pokušati shvatiti prednosti i mane svakog od njih. Naime, ne postoji univerzalni pristup koji bi mogao riješiti sve takve probleme na najbolji mogući način (teorem *Nema besplatnog ručka*).

Sama priroda problema isključuje egzaktan pristup zbog velikog prostora pretraživanja. Čak i na malim instancama, postojeće determinističke metode ne mogu ponuditi kvalitetna rješenja u razumnom vremenu [5]. Naime, svaki dan je potrebno prevesti na stotine kontejnera čiji se vremenski prozori mogu prostirati i kroz desetak mogućih smjena [6]. Točan, deterministički pristup dolazi u obzir samo za manje probleme ili za usporedbu s drugim algoritmima kada vrijeme nije presudan faktor [7]. Realnost je da su takvi slučajevi u manjini pa je nužno pronaći naprednije pristupe koji mogu rješavati realne, kompleksne probleme.
Stoga su veliku pažnju kroz povijest uživali evolucijski pristupi koji mogu relativno uspješno pronaći *dovoljno dobro* rješenje u razumnom vremenu. Međutim, takvi pristupi također imaju manjkavosti kada su u pitanju jako velike instance, često se kao značajna negativna strana ističe neefikasnost tih algoritama. Loše implementirani evolucijski algoritmi mogu biti iznimno neefikasni u slučaju da algoritam ne usmjerava pretraživanje u dobrom smjeru. To može rezultirati lošim rješenjem iako algoritam ima puno vremena na raspolaganju. Ako je pri rješavanju problema presudno vrijeme, onda nam je potreban pristup koji relativno brzo i efikasno pretražuje prostor mogućih rješenja. Pokazalo se da populacijski pristupi nisu najefikasniji pri rješavanju ovakvih problema s velikim prostorom pretraživanja jer dodavanje

dodatne složenosti (broj rješenja u jednoj generaciji) u sustav previše usporava algoritam [5]. S druge strane, ako nam vrijeme nije presudno, zanimljiviji su pristupi koji su sporiji, ali na kraju izvršavanja daju bolje rješenje. U tom smislu, populacijski pristupi mogu isto biti dobra opcija, ovisno o složenosti problema i implementaciji. Na primjer, [8] i [9] uspješno implementiraju genetski algoritam za rješavanje jednostavnijih problema, a [10] i [11] implementiraju algoritam kolonije mrava za rješavanje VRP-a. Algoritmi su u tom kontekstu relativno kvalitetni jer se radi o manjim problemima koji su u današnjem svijetu manje učestali.

Što se tiče inicijalizacije početnog rješenja, postoji dobro poznata tehnika umetanja koju su izmislili Clarke i Wright [12]. Kroz godine su se razvile drugačije tehnike inicijalizacije koje su brže i jednostavnije. Naime, kako se pokazalo da dobro početno rješenje ne rezultira nužno boljim završnim rješenjem, istraživači se često odlučuju za najjednostavniju i najbržu opciju inicijalizacije kako bi smanjili složenost algoritma [5]. Stvarni problemi su iznimno velike prirode pa je složenost inicijalizacije početnog izvedivog rješenja važan aspekt algoritma koji ne treba zanemariti. Idealno bi bilo kreirati početno rješenje relativno brzo jer možda problem ne dopušta dodatno vrijeme s kojim možemo naknadno poboljšati rješenje. Ako je to vrijeme ipak dostupno, moguće je iterativno poboljšavati trenutno najbolje rješenje.
Efikasna inicijalizacija dobrog rješenja stoga je temelj dobrog sustava koji može brzo pronaći dovoljno dobro rješenje za navedeni problem. Osim toga, složenost je iznimno važna jer konstantno imamo informaciju ažuriranu u stvarnom vremenu o tome gdje se vozila nalaze pa često ovi sustavi imaju grafičko sučelje pomoću kojeg se promatraju vozila i rute što zahtijeva brzinu izvođenja. Također, ponekad želimo postupno ažurirati rute ovisno o situaciji (gužve, radovi, prepreke,...) na cestama. U [13] se razmatra utjecaj složenih ograničenja na složenost inicijalizacije za općenit slučaj VRP-a te predlaže efikasnu heuristiku umetanja sa složenošću $O(n^3)$, a u nekim, rijetkim slučajevima $O(n^3 logn)$. To je značajno poboljšanje od uobičajenog pristupa koji ima složenost $O(n^4)$. Smanjenje složenosti nudi upravo prije opisano; efikasnu inicijalizaciju i mogućnost iterativnog poboljšanja nekim evolucijskim pristupom i/ili lokalnom pretragom ako je to potrebno.

Efikasna implementacija inicijalizacije početnog izvedivog rješenja izgledno je bitan dio suvremenog sustava za usmjeravanje vozila, međutim, tu priča ne bi trebala stati. Pronađeno rješenje potrebno je poboljšati i taj dio može biti iznimno zahtjevan, ovisno o postojećim ograničenjima. Osim uobičajenih evolucijskih pristupa, u zadnje vrijeme oni su korišteni u kombinaciji s tehnikama treniranja neuronskih mreža za poboljšanje pristupa. Istraživanja u tom području su novija pa ih nema jako puno, no rezultati su obećavajući [14] i [15]. Te

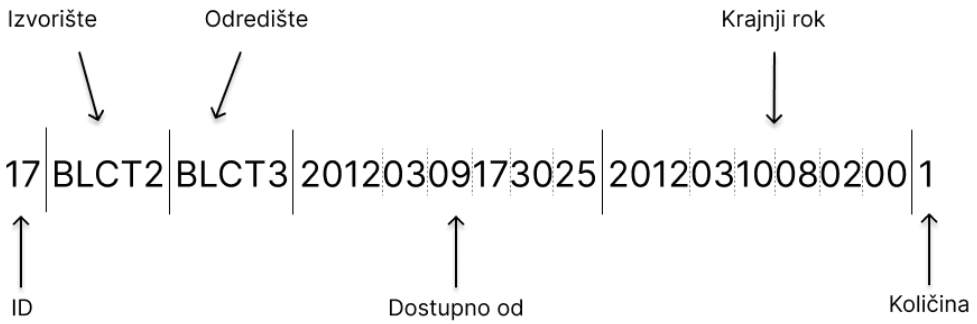
tehnike (podržano učenje - engl. *Reinforcement Learning*) sve se više ukomponiraju u postojeće evolucijske pristupe s ciljem povećanja efikasnosti algoritama. Ukratko, algoritmi pamte napredak algoritma do određenog vremena u prošlosti i pokušavaju analizirati okolno područje kako bi reducirali nasumično pretraživanje prostora. Na temelju analiziranih podataka, algoritam dobiva uvid u okolna stanja i ažurira određene parametre s ciljem usmjerenog pretraživanja i povećanja efikasnosti algoritma. Razvijene su razne tehnike usmjeravanja koje nastoje dobro usmjeriti pretraživanje kako bi algoritam bio što efikasniji.

U sklopu rada [16] promatra se susjedstvo te se ažuriraju težine putanja koje služe kao mjera dobrote kako bi se u kombinaciji s genetskim algoritmom pronašlo što bolje rješenje problema trgovačkog putnika. Susjedstvo označava rješenja koja se mogu dobiti određenim promjenama trenutačnog rješenja. U kontekstu usmjeravanja vozila često se radi o uništavanju određenog broja ruta i umetanju dotičnih zadataka na druga mjesta unutar rješenja. Osim toga, moguće su manje promjene, na primjer, zamjena redoslijeda nekoliko zadatka unutar rute ili promjena lokacije zadatka na drugu rutu. U [17] se koristi tehnika pretraživanja varijabilnog susjedstva pa se za kreaciju susjedstva predlaže niz operatora uništavanja i popravka rješenja koji su se pokazali uspješnima.
U [5] se koristi uobičajena tehnika pretraživanja varijabilnog susjedstva uz značajno napredniji sustav usmjeravanja pretraživanja. Naime, algoritam proučava okolna stanja (rješenja) i na temelju njih ažurira određene parametre koji pomažu u daljnjem usmjeravanju pretraživanja. Ako su okolna stanja obećavajuća, znači da je trenutačno rješenje u kvalitetnom području pretraživanja te algoritam nastavlja s pretragom u istom smjeru. Međutim, ako stanja oko trenutačnog nisu visoke kvalitete, smjer pretraživanja se mijenja s ciljem pronalaska potprostora bolje kvalitete. Ovakav pristup pokazao se obećavajućim, no veliki broj parametara zahtijeva iskusno podešavanje istih.

Veliki broj potpuno različitih algoritama koji relativno uspješno rješavaju varijante sličnog problema dodatno potvrđuje da nema univerzalnog pristupa koji jednako dobro rješava sve navedene probleme. Međutim, brojna istraživanja i konstantno napredovanje u području urodili su sve robusnijim pristupima koji uz analizu podataka (mreža unutar koje se raznosi roba, učestalost prijenosa kontejnera na određenim relacijama, vremenska prognoza, stanje na cestama,...) mogu dobiti dobar uvid u problem, a time i prilagoditi pristup i konkretne parametre kako bi se algoritam što bolje prilagodio dotičnom problemu. Tako se povećava efikasnost algoritma i samim time dobiva se bolje završno rješenje.

3. Opis problema

Broj različitih rješenja problema jednak je broju mogućih razmještaja zadataka unutar horizonta planiranja ruta. Lako je ustanoviti da je ovo NP-težak problem i da broj rješenja eksponencijalno ovisi o veličini problema (broju vozila, širini horizonta i broju zadataka) [5]. Stoga, kao što je prije spomenuto, nije moguće koristiti egzaktan pristup za stvarne (velike) probleme. Dakle, kvalitetan pristup koristi algoritam koji nastoji smanjiti složenost pretraživanja bez gubitka kvalitete konačnog rješenja. S ciljem predstavljanja što boljeg pristupa, važno je točno definirati problem.



Slika 3.1: Primjer zadatka jedne instance kojeg je potrebno obaviti

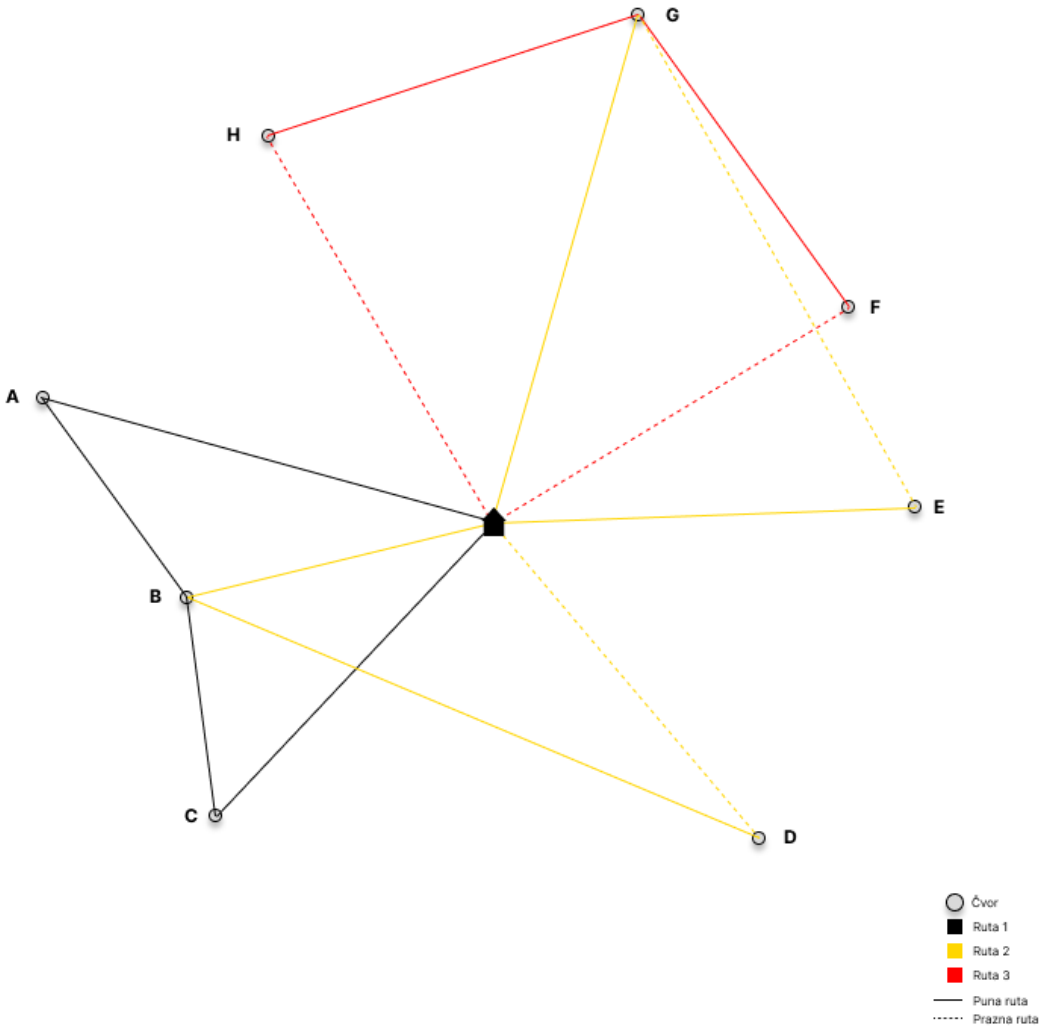
Između gradova unutar mreže potrebno je prevesti određen broj kontejnera. Arsenal vozila kojim raspolaže Ningbo luka sastoji se od 100 jednakih vozila [6] koja mogu prevesti samo jedan kontejner odjednom. Poštivajući lokalne zakone o radu, smjene u sklopu kojih se obavljaju zadaci su podijeljene na dnevnu i noćnu, svaka u trajanju od 12 sati. Dakle, prva smjena u danu počinje u 08:00 sati, a druga smjena počinje u 20:00 sati. Smjene su indeksirane sortirano po vremenu, dnevne smjene neparnim, a noćne parnim indeksima. Analiza podataka krajnjih rokova zadataka pokazala je da većina tih rokova pripada dnevnim smjenama. Međutim, većina zadataka ima širok raspon mogućih smjena pa je moguće podjednako rasporediti zadatke između dnevnih i noćnih smjena. Tijekom tih smjena vozači obavljaju zadatke, odnosno prenose kontejnere iz jednog grada u drugi. Primjer jednog zadatka može se vidjeti na slici 3.1. Važno je napomenuti da se vozila po završetku dnevne smjene ne vraćaju u centralno skladište, već to rade samo na kraju noćne smjene. Stoga je

ideja tijekom kreiranja putanje za dnevnu smjenu konstruirati put koji će u obzir uzeti sljedeću smjenu i završiti u onom čvoru (gradu) gdje sljedeća smjena započinje s radom kako bi te dvije smjene činile svojevrsnu cjelinu. Tako se smanjuje udio takozvanih *praznih putova* u sklopu kojih vozila ne prevoze robu, već samo putuju do lokacije koja predstavlja izvorište sljedećeg zadatka. Takvi dijelovi rute predstavljaju veliki trošak budući da se u sklopu njih ne obavlja nikakav posao, već vozač samo priprema vozilo za sljedeći zadatak. Budući da je ukupna udaljenost prijevoza robe fiksirana (određena je zadanim zadacima), glavni cilj implementacije jest minimizirati udio praznih putova. To će biti pokazatelj dobrote rješenja i skraćeno se može označiti s *LDR* (engl. *Loaded Distance Rate*) što je često korištena mjera u logističkim sektorima. Ako je udaljenost prijevoza robe iskazana mjerom *LD* (engl. *Loaded Distance*), a udaljenost praznih putova mjerom *UD* (engl. *Unloaded Distance*), LDR se može izračunati na sljedeći način:

$$LDR = LD/(LD + UD)$$

Dakle, pojedini zadaci imaju vremenske prozore unutar kojih se moraju obaviti, a vozila se moraju vratiti u skladište na kraju svake druge smjene. To znači da je ovo otvoreni problem usmjeravanja vozila s vremenskim prozorima (engl. *Open VRP with Time Windows - OPVRPTW*). Opisani problem razlikuje se od klasičnog VRP-a jer se roba prenosi između svih čvorova, a ne od centralnog skladišta do ostalih čvorova. Problem ima element VRP-a s preuzimanjem i dostavom (engl. *VRP with Pickup and Delivery*), ali se razlikuje po tome što je kapacitet vozila samo jedan kontejner te vozilo može obavljati samo jedan posao u istom trenutku. (Vozilo nikada ne prevozi robu za dva različita odredišta u istom trenutku.) Analiza konkretnih primjera problema pokazala je da je učestalost ruta između svih čvorova podjednaka što znači da su svi čvorovi podjednako važni (osim za skladištenje vozila, gdje je centralno skladište jedina opcija). Međutim, posebnu pažnju vrijedi posvetiti čvorištu koje spaja dnevnu smjenu s noćnom. Kao što je spomenuto, u idealnoj situaciji odredište posljednjeg zadatka dnevne smjene jednako je izvorištu prvog zadatka noćne smjene. Ovakav ishod poželjan je jer se promjena smjene ne odvija uz praznu putanju, no teško je da će se moći uvijek realizirati. U slučajevima kada takvo usmjeravanje vozila nije moguće, postupa se na sljedeći način. Ako vozač dnevne smjene stigne za vrijeme svog radnog vremena pripremiti vozilo za prvi zadatak noćne smjene, on će odvesti prazno vozilo od odredišta svog posljednjeg zadatka do dotičnog čvora. Ako vozač to ne bi stigao napraviti do kraja svog radnog vremena, vozilo će ostaviti na odredištu posljednjeg zadatka koji je obavio. Vozač noćne smjene tada će kao prvi dio svoje rute morati obaviti praznu putanju kako bi došao do izvorišta svog prvog zadatka. Slika 3.2 prikazuje primjer kreiranih ruta za tri vozila u dvije smjene. Dakle, svaka boja označava jednu rutu koju obavlja jedno vozilo. Na početku dnevne smjene, vozilo kreće iz skladišta i obavlja zadatke. Kada dođe vrijeme za promjenu smjene, vozilo preuzima vozač zadužen za noćnu smjenu i obavlja zadatke. Na kraju svoje

smjene, on mora vratiti vozilo u skladište. Razmotrimo kvalitetu svake pojedine rute. Ako znamo da su punim linijama označeni dijelovi ruta na kojima se obavljaju zadaci, a iscrtkane linije prikazuju prazne putanje u sklopu kojih vozila ne obavljaju nikakav posao, očito je da je najefikasnija crno označena ruta jer ona nema praznih putanja. S druge strane, crveno označena ruta nije kvalitetna jer ima značajan udio praznih putanja.



Slika 3.2: Primjeri ruta u sklopu kojih vozila obavljaju zadatke

Zbog toga što se vozila vraćaju u skladište na kraju svake druge smjene, algoritam traži putove, a ne cikluse kao što je to slučaj kod klasičnog VRP-a. Međutim, ovaj problem nije jednak problemu usmjeravanja školskog busa kod kojega su dnevna i noćna ruta identične. Naime, dvije rute jednog vozila koje pripadaju dnevnoj i sljedećoj noćnoj smjeni ne moraju nužno biti identične i u pravilu neće ni biti. Nadalje, često se u sklopu VRP-a može ignorirati

vrijeme utovara i istovara robe budući da je ono relativno kratko u usporedbi s vremenom prijevoza robe. U ovom slučaju, to nije tako. Vrijeme utovara i istovara robe je značajno te se mora uračunati u izračun vremena. Naime, gradovi su relativno blizu te je vrijeme prijevoza robe nije značajno dulje od vremena utovara i istovara robe.

Tablica 3.1: Lista stvarnih instanci u Ningbo luci.

Ime instance	Broj smjena	Broj zadataka
NP4-1	4	465
NP4-2	4	405
NP4-3	4	526
NP4-4	4	565
NP4-5	4	765
NP6-1	6	1073
NP6-2	6	920
NP6-3	6	384
NP6-4	6	746
NP6-5	6	557
NP8-1	8	913
NP8-2	8	827
NP8-3	8	786
NP8-4	8	1008
NP8-5	8	798

Raspoložive instance pomoću kojih se analizira kvaliteta algoritma dijele se na *stvarne* i *umjetno generirane* instance. Svaka od ovih grupa ima podgrupe koje čine 25%, 50%, 75% i 100% količine zadataka iz navedene instance zbog lakšeg testiranja. U okviru analize predloženog pristupa većinom će se promatrati instance pune veličine s ciljem što reprezentativnijih rezultata. Također, svaka umjetno generirana instanca ima određenu konfiguraciju koja opisuje zadatke koje ona sadrži. Naime, zadaci unutar instance su *labavi* (engl. *Loose*) ili *tijesni* (engl. *Tight*) te *balansirani* (engl. *Balanced*) ili *nebalansirani* (engl. *Unbalanced*). Preciznije, zadatak je labav ako je njegov vremenski prozor relativno širok (čak i do tri dana). To znači da je taj zadatak moguće smjestiti u nekoliko susjednih smjena. S druge strane, smještaj tijesnog zadatka puno je složeniji jer je njegov vremenski prozor nešto uži. Takav zadatak možda je moguće smjestiti samo u jednu smjenu jer su krajnje vrijeme i početno vrijeme izvršavanja zadatka udaljeni samo nekoliko sati. Balansiranost govori o rasporedu zadataka kroz horizont. Balansirane instance sastoje se od dobro raspoređenih zadataka kroz

Tablica 3.2: Lista umjetno generiranih instanci u Ningbo luci.

Ime instance	Opis instance	Broj smjena	Broj zadataka
LB4-1	Labava, balansirana	4	484
LB4-2	Labava, balansirana	4	396
TB4-3	Tijesna, balansirana	4	282
TB4-4	Tijesna, balansirana	4	368
LU4-5	Labava, nebalansirana	4	448
LU4-6	Labava, nebalansirana	4	479
TU4-7	Tijesna, nebalansirana	4	217
TU4-8	Tijesna, nebalansirana	4	354
LB8-1	Labava, balansirana	8	592
LB8-2	Labava, balansirana	8	657
TB8-3	Tijesna, balansirana	8	497
TB8-4	Tijesna, balansirana	8	621
LU8-5	Labava, nebalansirana	8	551
LU8-6	Labava, nebalansirana	8	559
TU8-7	Tijesna, nebalansirana	8	607
TU8-8	Tijesna, nebalansirana	8	525
2000	Mješovita, nebalansirana	8	2614

cijeli horizont. Nebalansirane instance malo su kompleksnije jer njihovi zadaci nisu dobro raspoređeni pa se može dogoditi da je prevelik zahtjev za vozila u određenoj smjeni, dok u nekoj drugoj smjeni uopće nema potrebe za velikim brojem vozila.

Prva slova engleskih naziva navedenih karakteristika i broj smjena u horizontu umjetno generirane instance koriste se za njeno imenovanje pa je iz samog imena očito kakva je ta instanca. Na primjer, jedna od instanci koja ima labave i balansirano raspoređene zadatke te horizont od četiri smjene naziva se 'LB4-1'. Tablica 3.1 sadrži listu svih *stvarnih* instanci, dok tablica 3.2 sadrži listu svih *umjetno generiranih* instanci. Instance su preuzete sa stranice na kojoj se nalazi nekoliko radova vezanih uz istu temu (https://sites.google.com/nottingham.ac.uk/port-management). Skup je raznovrstan i očito je da sadrži jednostavnije instance s manje zadataka i jako složene instance s velikim brojem nebalansiranih zadataka s uskim vremenskim prozorima. Ovaj skup je stoga dobar jer testiranje nad njim može pokazati je li algoritam robustan.

3.1. Formulacija problema

Tablica 3.3 prikazuje notacije korištene za definiciju matematičkog modela. Cilj te definicije jest jasno označiti sva ograničenja i cilj optimizacije.

Tablica 3.3: Lista oznaka korištenih pri definiciji matematičkog modela.

Oznaka	Opis
K	Ukupan broj vozila dostupan u svakoj smjeni
N	Skup čvorova (gradova) između kojih se prevozi roba
D	Skup vremenski sortiranih dana
S	Skup vremenski sortiranih smjena
$[S_s, S_e]$	Vremenski prozor smjene *S*
S_{day}	Skup vremenski sortiranih dnevnih smjena, indeksiranih neparno
S_{night}	Skup vremenski sortiranih noćnih smjena, indeksiranih parno
T	Skup zadataka koje je potrebno obaviti
$[a_i, b_i]$	Vremenski prozor zadatka *i*. Zadatak se mora početi obavljati unutar definiranog vremenskog prozora.
l_i	Vrijeme potrebno za obavljanje zadatka *i*
T_i	Vremenska oznaka dolaska na čvor izvorišta zadataka *i*
B_i	Vremenska oznaka početka obavljanja zadataka *i*
t_{ij}	Vrijeme putovanja (u minutama) od čvora *i* do čvora *j*
d_{ij}	Udaljenost putovanja od čvora *i* do čvora *j*
x_{ij}^s	Binarna varijabla odluke, označava je li zadatak s izvorištem *i* i odredištem *j* odrađen u smjeni *s*

Broj vozila *K* označava veličinu homogenog arsenala vozila koja su dostupna u svakoj smjeni. Svako pojedino vozilo može prevesti maksimalno jedan kontejner odjednom. Kontejneri se prevoze između čvorova *N* za svakog od kojih je poznato vrijeme potrebno za utovar i istovar jednog kontejnera. Dani koji ulaze u horizont konkretne instance imaju oznaku *D* i indeksirani su redom, počevši od broja jedan. Svaki dan ima dnevnu (S_{day}) i/ili noćnu smjenu (S_{night}), ovisno o točnoj granici horizonta, to jest, o tome pripada li dotična smjena horizontu. Naime, rubni dani možda nemaju obje smjene ako one ne pripadaju horizontu u sklopu kojeg je potrebno obaviti zadatke *T*. Naravno, svaka smjena ima svoje početno i završno vrijeme što je jednako početnom i završnom vremenu radne smjene vozača koji je zadužen za tu smjenu. Svaki od zadataka ima izvorišni (engl. *Origin*) i odredišni grad (engl. *Destination*) te prvo moguće a_i i krajnje vrijeme b_i početka obavljanja zadatka. Važno je napomenuti da ako vozilo dođe do početnog čvora zadatka prije njegovog vremenskog pro-

zora, ono mora čekati do a_i. Zadaci koji imaju više kontejnera podijeljeni su na više manjih zadataka tako da je za svaki pojedini zadatak potrebno prenijeti samo jedan kontejner. Ova promjena pojednostavljuje formulaciju problema i povećava efikasnost predloženog algoritma zbog lakšeg razmještanja zadataka. Nadalje, vrijeme potrebno za obavljanje zadataka *i* označeno je s l_i. Ono uključuje utovar robe na izvorištu, prijevoz robe do odredišta te istovar robe na tom istom odredištu. Za vrijeme i udaljenost putovanja između dva čvora koriste se oznake t_{ij} i d_{ij}. Za potrebe definiranja ograničenja potrebna je i vremenska oznaka početka obavljanja zadatka *i* za što se koristi A_i. Vrijeme putovanja izraženo je u minutama što je bitno za izračun izvedivosti rješenja, dok su duljine putovanja izražene u kilometrima. Međutim, mjerna jedinica za udaljenost može se zanemariti jer se za procjenu kvalitete rješenja koristi udio udaljenosti prijevoza robe u ukupnoj sumi putanja vozila (prijevoz robe i prazne putanje). Završno, binarna varijabla odluke x_{ij}^s označava obavlja li se određeni zadatak s izvorištem *i* i odredištem *j* u smjeni *s*. Navedene notacije korištene su i u [5], uz manje preinake.

Koristeći navedene notacije, moguće je definirati optimizacijski problem koji je potrebno riješiti:

$$Minimiziraj \quad F(x) = \sum_{s \in S} \sum_{i \in T} \sum_{j \in T} d_{ij} \cdot x_{ij}^s \tag{3.1}$$

gdje vrijedi:

$$\sum_{s \in S} \sum_{i \in T} x_{ij}^s = 1, \quad \forall j \in T \tag{3.2}$$

$$\sum_{s \in S} \sum_{j \in T} x_{ij}^s = 1, \quad \forall i \in T \tag{3.3}$$

$$a_i \leq B_i \leq b_i - l_i \tag{3.4}$$

$$x_{ij}^s * S_s \leq x_{ij}^s * T_i \tag{3.5}$$

$$x_{ij}^s * (B_i + l_i) \leq x_{ij}^s * E_s \tag{3.6}$$

$$x_{ij}^s \in \{0, 1\} \tag{3.7}$$

$$\sum_{j \in N} x_{0j}^s = K, \quad \forall s \in S_{day} \tag{3.8}$$

$$\sum_{i \in N} x_{i0}^s = K, \quad \forall s \in S_{night} \tag{3.9}$$

Funkcija cilja označena je s *(3.1)*, ona označava da je cilj minimizirati ukupnu putanju svih vozila. Budući da je duljina prijevoza robe fiksna jer je određena konkretnim zadacima

koji čine instancu, cilj jest minimizirati udio praznih putanja, odnosno maksimizirati LDR poštivajući sva dana ograničenja. Ograničenja *(3.2) (3.2)* označavaju da je svaki zadatak obavljen točno jednom i da su svi zadaci obavljeni. Ograničenje *(3.4)* označava da je rad na svakom pojedinom zadatku započet unutar njegovih vremenskih prozora. Ograničenja *(3.5)* i *(3.6)* označavaju da je rad jednog vozila odrađen unutar vremenskih prozora jedne smjene. Ograničenje *(3.7)* označava da je varijabla odluke binarna, odnosno da može poprimiti samo vrijednosti jedan ili nula. Ponovimo, ovo je OVRP pa se vozila vraćaju u skladište na kraju svake druge smjene. Navedeno je osigurano definiranjem ograničenja *(3.8)* (za dnevnu smjenu) i *(3.9)* (za noćnu smjenu). Osim toga, ta ograničenja osiguravaju poštivanje ograničenja o maksimalnom broju vozila u smjeni.

Zaključak analize i formalne definicije problema jest da je ovo nelinearno ograničen problem s ogromnim prostorom pretrage. Veličina prostora pretrage određena je duljinom horizonta ($|S|$), brojem vozila (K) i brojem zadataka ($|T|$). Budući da je ukupan broj mogućih ruta u rješenju jednak $|S| * K$, a broj permutacija zadataka jest $|T|!$, veličina prostora pretrage jest $|S| * K * |T|!$ [5]. Stoga je logičan zaključak da je za pronalazak relativno dobrog rješenje potreban kvalitetan algoritam koji će dobro usmjeravati pretraživanje prostora.

4. Pristup

Razmatrajući postojeće zahtjeve, moguće je zaključiti da algoritam ne treba biti jako brz. Naime, u morskim lukama dobro su isplanirani dani u bliskoj budućnosti. Dolazak kontejnera poznat je najmanje nekoliko dana unaprijed [2], rijetko se događa da se iznenadno pojavi neka pošiljka koju je potrebno smjestiti u neku rutu. Međutim, ne treba očekivati da je takvo rješenje zadovoljavajuće za današnje sustave pa je cilj osigurati dobru efikasnost algoritma zbog prije navedenih prednosti vezanih uz brzinu inicijalizacije, ali i jer se uvijek mogu dogoditi nepredvidive stvari zbog kojih se ne mogu realizirati planirane rute. Konkretno, realno je za očekivati da svaki dan barem jedno vozilo ima određeni kvar ili zakazano održavanje te je stoga važno imati robustan sustav koji se može nositi s takvim izazovima. Na primjer, [6] u obzir uzima 29 od mogućih 30 vozila te tako uzima u obzir spomenute situacije. Još jedan adekvatan pristup koji u obzir uzima buduće promjene je kreirati rjeđe rute i tako ostaviti mjesta za smještaj nadolazećih zadataka u njih. Međutim, ovakav pristup je u globalu skuplji jer u slučajevima kada nema problema (koji su puno češći), rute nisu optimalno konstruirane. Dakle, dobar cilj bio bi kreirati optimalne rute s postojećim zadacima, ali uzeti u obzir probleme s vozilima te pokušati generirati rješenje s manjim brojem vozila nego je ukupno na raspolaganju. Tako je moguće smanjiti broj vozila koja su potrebna sektoru (iako to u ovom slučaju nije tako budući da je broj vozila fiksan). Upravo ovakva rješenja nudi predloženi algoritam (u nastavku VNTS - Variable Neighbourhood Tabu Search).

VNTS algoritam podijeljen je u tri dijela. Na samom početku sustav analizira danu mrežu; proučava broj, raspored i udaljenost čvorova i vremena najduljeg i najkraćeg vremana utovara i istovara robe. Na temelju tih podataka ažuriraju se parametri algoritma s ciljem što boljeg usmjeravanja sustava. Iako te promjene nisu velike, one su značajne zbog prilagodbe na veličinu instance. Primjerice, u slučaju velike instance, algoritam prilagođava veličinu susjedstva i način izračuna susjedstva s ciljem smanjenja složenosti i brzine izračuna prvotnog izvedivog rješenja. Sustav također određuje raspon mogućih smjena za svaki pojedini zadatak. To znači da svakom zadatku zadaje indeks prve smjene u kojoj je moguće taj zadatak rasporediti i indeks zadnje takve smjene. Ovaj korak iznimno je važan za cjelokupan sustav jer se pokazalo da dobrom analizom podataka algoritam lakše raspo-

ređuje zadatke i efikasnije radi. Općenito, optimalni parametri u evolucijskim algoritmima temelj su dobrog algoritma. Odabir optimalnih parametara zahtijeva proces detaljnog eksperimentiranja s ciljem odabira vrijednosti parametara koje rezultiraju najefikasnijom verzijom algoritma. Stoga ovaj algoritam pokušava sam ažurirati parametre ovisno o prije navedenim osobinama instance. Kvalitetna inicijalizacija parametara samo je početni korak koji usmjerava algoritam. Parametri se kasnije i dinamički ažuriraju tijekom izvođenja algoritma, o čemu će biti riječi kasnije.

Inicijalizacija početnog izvedivog rješenja sljedeći je korak algoritma. Nekoliko tipova inicijalizacije dobro je poznato u ovom području. Međutim, pokazalo se da taj aspekt algoritma nije nužno presudan jer bolje inicijalno rješenje ne vodi nužno do boljeg završnog rješenja pa se najčešće bira najbrža ili najjednostavnija metoda inicijalizacije. Ali, kao što je i spomenuto, složenost pristupa i kvaliteta brzo generiranog rješenja važan su aspekt današnjih sustava te su stoga testirane brojne metode inicijalizacije. Najprije je testirana Clarke-Wright metoda inicijalizacije koja svakom vozilu dodijeli jednu rutu i onda pokušava smanjiti broj ruta spajajući postojeće rute na način koji maksimalno poboljšava usmjeravanje (pohlepan pristup). Ovaj pristup dobro je poznat i često korišten u literaturi. Ovaj rad detaljnije ga neće razmatrati zbog toga što se pokazao sporijim od drugih metoda i dobiveno rješenje je imalo tendenciju zapinjanja u lokalnom optimumu.
Nadalje, predložena je nova vrsta inicijalizacije koja kreira cikluse, a ne putove. Metoda promatra svaku dnevnu i sljedeću noćnu smjenu kao cjelinu te kreira rutu za tu cjelinu. Takav pristup i dalje poštuje sva dana ograničenja (trivijalno je takvu rutu podijeliti na dnevnu i noćnu smjenu), ali znatno pojednostavljuje implementaciju i način pretraživanja prostora stanja. Osim toga, takav pristup implicitno nastoji minimizirati udio praznih putanja između spomenute dvije smjene što dodatno povećava kvalitetu rješenja.
U sklopu rada implementirana je i *inicijalizacija po smjenama* koja uzima u obzir prije navedeni raspon mogućih smjena za svaki zadatak. Ovaj pristup uzima u obzir činjenicu da je u svakoj smjeni moguće iskoristiti sva dostupna vozila. Dakle, za svaku smjenu je moguće generirati *K* ruta. Maksimalan broj ruta koji je moguće generirati ovisi o broju vozila i broju smjena; $K \times |S|$. Navedeni pristup iterira po svim danima u horizontu instance budući da svaki dan ima dvije smjene; dnevnu i noćnu i tako pokušava generirati izvediv skup ruta čija veličina je ograničena brojem dostupnih vozila. Dakle, ovaj pristup izvršava prvu fazu Clarke-Wright algoritma, ali ne pokušava reducirati broj ruta, osim ako broj ruta u smjeni nije radikalno veći od dopuštenog. Ideja je da takvo inicijalno rješenje ima manju vjerojatnost zapinjanja u lokalnom optimumu pri kasnijem pretraživanju. Međutim, testiranja su pokazala da nije tako te je korištena inicijalizacija koja kreira cikluse budući da se ona pokazala najjednostavnijom. Naime, predložena inicijalizacija pokazala se jako brzom čak i na

velikim instancama.

Tablica 4.1 prikazuje rezultate različitih varijanta ove inicijalizacije na stvarnim instancama veličine 100%, a tablica 4.2 prikazuje isto to na umjetno generiranim instancama veličine 100%.
Inicijalizacija koja najprije smješta zadatke s najranijim krajnjim rokom (engl. *Urgency Based Insertion Heuristic*) ima oznaku *UBIH*. Metoda inicijalizacije koja najprije smješta zadatke s najužim vremenskim prozorima (Width Based Insertion Heuristic) označena je s *WBIH*, a inicijalizacija koja razmješta zadatke slučajnim redoslijedom (engl. *Random Shuffle Insertion Heuristic*) označena je s *RSIH*. Tijekom umetanja svakog pojedinog zadatka, on namjerno nije smješten na optimalno poziciju upravo da bi se izbjegao problem zapinjanja u lokalnom optimumu. Sve spomenute inicijalizacije rezultiraju izvedivim rješenjem, osim ako instanca nije prevelika. U tom slučaju, moguće je da određene smjene imaju više vozila nego je to dozvoljeno. To ograničenje rješava se kasnije, u trećoj fazi algoritma. Iako je moguće generirati kvalitetnija početna rješenja, to nije tako napravljeno jer se pokazalo da dobra početna rješenja ne vode nužno do boljih završnih rješenja. Štoviše, inicijalna rješenja visoke kvalitete mogu brzo zapeti u lokalnom optimumu. Dakle, prioritet je brzina inicijalizacije uz poštivanje ograničenja. U krajnjoj verziji algoritma najbolja se iskazala nasumična inicijalizacija zbog brzine, ali i kvalitete. Naime, nasumična inicijalizacija na većini instanci bila je kvalitetnija od ostalih varijanti, kao što se može vidjeti u tablicama.

Algoritam 1 Algoritam lokalne pretrage - LS

Ulaz: trenutačno rješenje *S*, parametri *P*
while $!optimum\_dosegnut$ **do**
    $zadaci \leftarrow ukloni\_praznu\_rutu(S, P)$
    $zadaci \leftarrow ukloni\_skupi\_zadatak(S, P)$
    $ubaci\_zadatke(S, P, zadaci)$ ▷ Umetni zadatke na optimalna mjesta
end while

Treći dio algoritma implementira pretraživanje promjenjivog susjedstva uz tabu listu u sklopu kojeg algoritam pretražuje susjedstva različitih tipova i veličina s ciljem inkrementalnog poboljšanja najboljeg rješenja. Ovaj dio algoritma u svakoj iteraciji pretražuje nekoliko slojeva susjedstva trenutnog rješenja, a na kraju iteracije radi lokalnu pretragu s ciljem intenzifikacije. Pokazalo se da izvođenje lokalne pretrage samo na kraju svake iteracije predstavlja dobar balans između lokalnog istraživanja i diverzifikacije. Lokalna pretraga povećava efikasnost algoritma, a u isto vrijeme ne pridonosi stagnaciji jer algoritam ne sprema pronađeno rješenje kao trenutačno. Pseudokod algoritma lokalne pretrage dan je u algoritmu 1.

Tablica 4.1: Prosječni rezultati različitih tipova inicijalizacije na stvarnim instancama veličine 100%.

	Metoda inicijalizacije		
Instanca	**RSIH**	**UBIH**	**WBIH**
NP4-1	57.56%	56.22%	55.12%
NP4-2	55.35%	52.12%	52.06%
NP4-3	56.61%	52.64%	53.81%
NP4-4	54.12%	52.96%	52.61%
NP4-5	58.49%	52.93%	52.88%
NP6-1	56.11%	51.15%	52.47%
NP6-2	54.39%	51.57%	50.45%
NP6-3	56.21%	52.02%	51.42%
NP6-4	56.31%	52.95%	52.21%
NP6-5	56.67%	53.32%	54.18%
NP8-1	56.20%	54.28%	55.74%
NP8-2	56.94%	54.52%	54.81%
NP8-3	54.97%	51.42%	50.75%
NP8-4	53.64%	52.57%	52.9%
NP8-5	56.62%	54.06%	54.15%

Tijekom izvođenja, algoritam redovito ažurira svoje parametre kako bi povećao efikasnost pretrage. Konkretno, ažurira se broj slojeva susjedstva koji se kreiraju u svakoj iteraciji i parametri bitni za kreaciju konkretnog susjednog rješenja; količina uklonjenih zadataka i ruta te način umetanja dotičnih zadataka. Osim toga, ako algoritam prepozna da stagnira, pokušat će diverzifikacijom trenutačnog rješenja prestati istraživati lokalno susjedstvo i preusmjeriti pretragu negdje drugdje. Važan element algoritma predstavlja tabu lista koja pridonosi efikasnosti i smanjuje vjerojatnost zapinjanja u lokalnom optimumu. Pseudokod cjelokupnog algoritma dan je u algoritmu 2.

Budući da je ovo iznimno kompleksan problem za koji nije lako pronaći kvalitetno rješenje, važno je implementirati algoritam na način koji efikasno radi. Konkretno, korak (*3.2*) algoritma najskuplji je korak budući da je potrebno generirati skup evaluiranih rješenja. Stoga se za izvršavanje tog dijela koda nastoje maksimalno iskoristiti mogućnosti sustava na kojem se algoritam pokreće pa se taj dio izvršava paralelizacijom koda pomoću alata OpenMP. Ova preinaka rezultira velikim ubrzanjem koje je od velike koristi. Osim toga, budući da se za svaku rutu može izračunati njena LDR mjera, od velike je koristi implementirana priručna memorija (engl. *cache*) koja naznačuje postoje li promjene u ruti i je li potrebno nanovo

Tablica 4.2: Prosječni rezultati različitih tipova inicijalizacije na umjetno generiranim instancama veličine 100%.

	Metoda inicijalizacije		
Instanca	**RSIH**	**UBIH**	**WBIH**
LB4-1	55.63%	52.45%	53.35%
LB4-2	59.86%	59.78%	58.96%
TB4-3	55.72%	54.39%	54.55%
TB4-4	55.06%	53.01%	52.35%
LU4-5	52.98%	52.24%	51.18%
LU4-6	53.03%	50.21%	51.24%
TU4-7	49.67%	49.08%	48.44%
TU4-8	50.71%	48.73%	49.83%
LB8-1	57.28%	55.45%	55.63%
LB8-2	57.17%	53.62%	55.14%
TB8-3	56.70%	54.93%	55.74%
TB8-4	55.60%	53.90%	53.79%
LU8-5	55.30%	53.46%	55.35%
LU8-6	56.61%	55.26%	55.93%
TU8-7	48.38%	47.40%	47.33%
TU8-8	49.05%	48.57%	49.00%

evaluirati tu rutu. Prije navedeni indeksi prve i zadnje moguće smjene za svaki pojedini zadatak također povećavaju brzinu izvršavanja. Zaključno, algoritam je implementiran u programskom jeziku C++ zbog što veće brzine izvršavanja. Sve navedeno čini veliku razliku u ukupnoj brzini izvršavanja i omogućuje brže pronalaženje kvalitetnog rješenja. Istraživanjem literature i zahtjeva industrije, zaključeno je da brz algoritam puno doprinosi cjelokupnom sustavu. Na primjer, uz ovakav algoritam moguće je implementirati dinamičku grafičku vizualizaciju svih vozila i njihovih putanja.

4.1. Usmjeravanje pretraživanja

Algoritam koristi jednostavnu heuristiku za spajanje zadataka; udaljenost susjednih gradova. Pomoću tih udaljenosti određuje se kvaliteta pozicije određenog zadatka unutar rute; ako je dotični zadatak povezan s okolnima, to jest, nema praznih ruta koje ga povezuju, onda je on kvalitetno (*jeftino*) smješten. S druge strane, zadatak je *skup* ako nije povezan s okolnim za-

Algoritam 2 VNTS algoritam

1. $P \leftarrow inicijaliziraj\_parametre(instanca)$ ▷ Inicijalizacija parametara
2. $S \leftarrow inicijaliziraj\_pocetno(instanca, P)$ ▷ Inicijalizacija početnog rješenja
3. VNTS pretraga:

while $!uvjet\_zaustavljanja$ **do**
  $P \leftarrow azuriraj\_parametre(P)$
  while $i \leq broj\_slojeva$ **do**
    $i \leftarrow i + 1$
    $susjedstvo \leftarrow generiraj\_susjedstvo(S, P)$
    $S \leftarrow odaberi(susjedstvo, P)$
    $tabu\_lista.umetni(S)$
  end while
  $lokalno \leftarrow lokalna\_pretraga(S, P)$
end while

dacima. Veličina prostora pretraživanja [5] ukazuje na veliki broj takvih razmještaja i poziva na pametno usmjeravanje pretraživanja i samim time dinamičko ažuriranje parametara što nudi predloženi sustav.

Prije spomenuta prva faza algoritma završava inicijalizacijom parametara čije vrijednosti ovise o konkretnim vrijednostima mreže i rasporedom zadataka. Iako se ti parametri podešavaju na početku izvođenja programa, njihove vrijednosti ne ostaju iste tijekom cijelog izvođenja. Analizom okolnih rješenja, nedavnih poboljšanja najboljeg rješenja i ovisno o trenutačnom stanju algoritma (intenzifikacija/diverzifikacija), sustav podešava parametre s ciljem povećanja efikasnosti pretraživanja. Na primjer, veličina susjedstva jedan je od najutjecajnijih parametara evolucijskih algoritama. Sustav prepoznaje potrebu za povećanjem ili smanjenjem veličine susjedstva i ažurira dotičan parametar. Ovaj parametar je od velike pomoći pri diverzifikaciji pretraživanja. Ako sustav prepozna da je pretraživanje zapelo u lokalnom optimumu, povećanje susjedstva uvelike pomaže u bijegu iz tog lokaliteta. Kao što je i spomenuto, pamćenje nedavnog uspjeha također je indikator koji pomaže u ažuriranju parametara. Navedeno je implementirano kao svojevrsna memorija gdje su nedavni rezultati bitniji od onih iz starijih generacija. Ako su rezultati iz nedavne prošlosti iznimno dobri, sustav prepoznaje kvalitetan potprostor te pokušava detaljnije istražiti okolna rješenja (intenzifikacija). Nakon dobrog istraživanja za očekivati je da će sustav zapeti u optimumu što se lako vidi analizom napredovanja trenutačnog rješenja. Tada nastupa diverifikacija pretraživanja; veći broj generiranih susjeda i veće promjene pri kreaciji susjeda; veći broj ruta i zadataka koji se uklanjaju. Potpuni pregled svih parametara dan je u tablici 4.3.

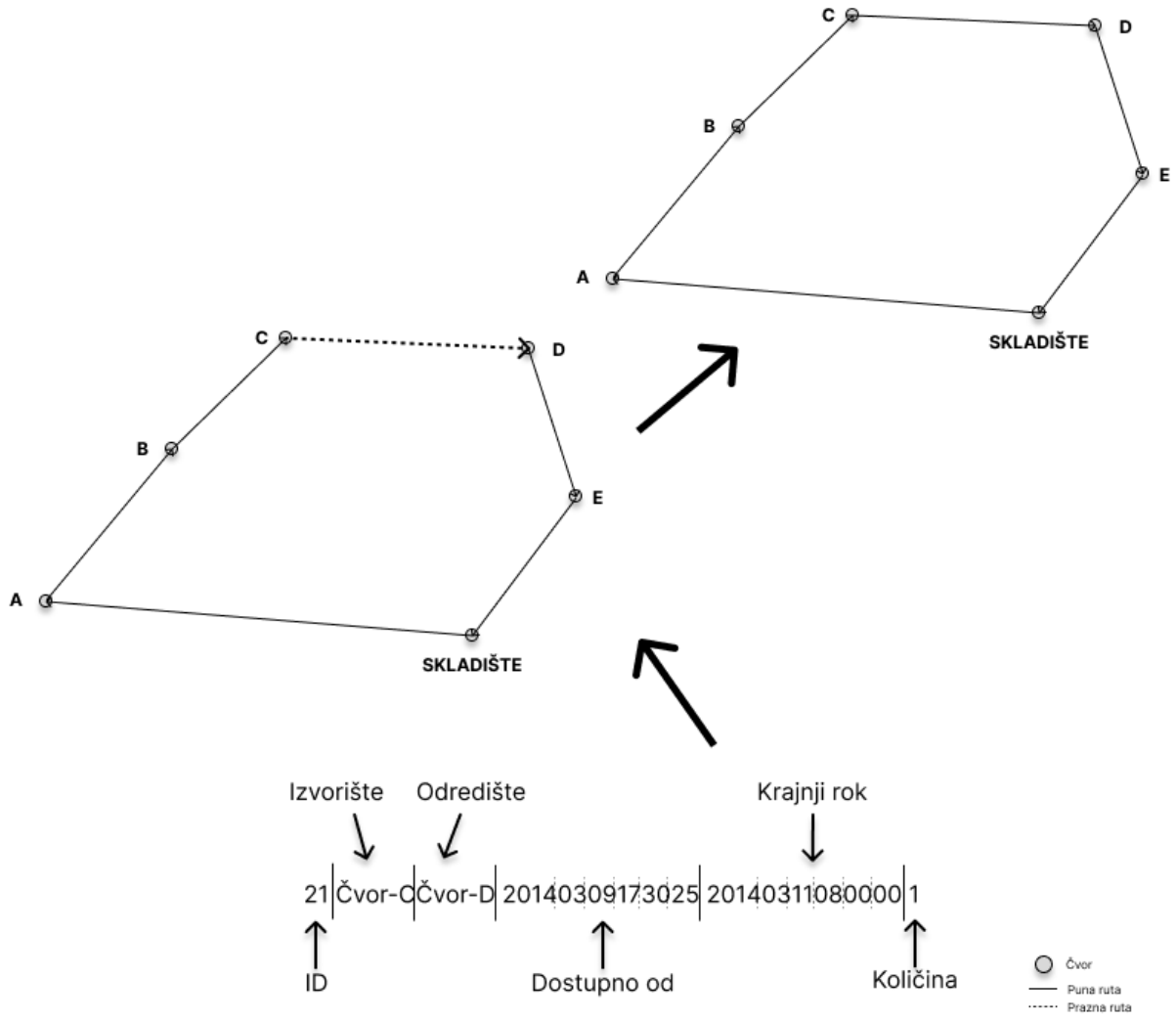
Tablica 4.3: Pregled parametara korištenih u sklopu algoritma.

Opis	Uporaba
iteracija zadnjeg poboljšanja najboljeg rješenja	Za praćenje poboljšanja najboljeg rješenja
iteracija zadnjeg poboljšanja trenutačnog rješenja	Za praćenje poboljšanja trenutačnog rješenja
iteracija diverzifikacije	Praćenje faze intenzifikacije/diverzifikacije
broj ruta	Broj uklonjenih ruta za kreaciju susjednog rješenja, ovisi i o sloju
broj zadatka	Broj uklonjenih zadataka za kreaciju susjednog rješenja, ovisi i o sloju
broj susjeda	Bazni broj susjeda, povećava se ovisno o sloju susjedstva
broj slojeva	Broj slojeva susjedstva, ažurira se ovisno o performansama
biraj najboljeg	Biraj najbolje susjedno rješenje
umetni optimalno	Smjesti zadatke na optimalan način

4.2. Susjedstvo i diverzifikacija

Način generiranja susjedstva uvelike određuje kvalitetu algoritma. Susjedstvo mora biti dovoljno malo kako bi se rješenja poboljšavala precizno (mala poboljšanja) i kako bi se dovoljno dobro istražio lokalni optimum. S druge strane, susjedstvo mora biti dovoljno raznoliko kako bi algoritam mogao izbjeći zapinjanje u lokalnom optimumu. Međutim, susjedstvo također ne smije biti preveliko jer to može značajno usporiti algoritam, ali i smanjiti efikasnost algoritma jer neće nikada moći intenzivno pretražiti susjedstvo i pronaći najbolja rješenja u lokalnom susjedstvu. Naravno, balans između ova dva zahtjeva teško je postići. Često se kao način generiranja susjedstva bira neka prije određena metoda koja se ne mijenja tijekom izvršavanja algoritma. Taj pristup može biti jako dobar, ali zahtijeva dobru analizu i eksperimentiranje s podacima. Predloženi algoritam dinamično određuje broj slojeva susjedstva koji se generira. Na temelju veličine problema određuje se početno susjedstvo; tip kreacije i broj slojeva koji se generira. Kasnije, tijekom izvođenja algoritma, parametri se ažuriraju ovisno o efikasnosti pretrage.

Što se tiče operatora koji se koriste za generiranje susjedstva, koristi se nekoliko njih ovisno o fazi pretraživanja. Broj ruta i zadataka koji se uklanjaju za generiranje susjednog rješenja ovisi o parametrima algoritma u datom trenutku. Testirane su određene metode koje su često korištene u literaturi. Izmjena pozicija nasumično odabranih zadataka unutar iste rute ili dvije različite rute pokazala se ne pretežito korisnom jer postojeći algoritam dobro



Slika 4.1: Primjer optimalnog umetanja zadatka u rutu

razmještava zadatke s predloženom heuristikom te zamjena pozicija zadataka često narušava ograničenja vremenskih prozora. Zato ne rezultira boljim rješenjima. Ukratko, veliki udio ishoda poziva ovog operatora rezultira neizvedivim rješenjima što znatno smanjuje efikasnost algoritma. Iako u teoriji takav pristup dugoročno možda može ponuditi bolja rješenja, pokazalo se da nije tako. Takav pristup značajan udio svojeg izvođenja troši na razmatranje neizvedivih rješenja i samim time je neefikasan jer nema konkretnu heuristiku koja ga vodi prema izvedivim rješenjima.

Stoga predloženi algoritam koristi metode zasnovane na boljim heuristikama. Za kreaciju susjednog rješenja, ovisno o fazi i parametrima algoritma, koriste se različite metode uklanjanja ruta i/ili uklanjanja pojedinih zadataka iz trenutačnog rješenja. Navedeni operatori

rezultiraju nepotpunim rješenjem u koje je tada potrebno umetnuti dotične zadatke. Naravno, ti zadaci se tada ne smještaju nasumično u rješenje, već se koristi funkcija cilja za određivanje dobrote generiranog rješenja. Opet, ovisno o parametrima u datom trenutku, zadaci se smještaju u rješenje na određeni način. Algoritam u normalnoj fazi nasumično bira način uklanjanja rute i zadatka, a tijekom faze lokalne pretrage koristi pohlepne operatore koji nastoje maksimalno poboljšati rješenje. Dakle, tijekom faze intenzifikacije koriste se operatori koji nastoje generirati susjedno rješenje koje će biti bolje od trenutnog rješenja. Konkretno, iz trenutačnog rješenja miče se ruta koja ima veliki udio *praznih putanja*, to jest, *skupo* smještenih zadataka. Micanjem te rute preostaju određeni zadaci koji nisu dodijeljeni niti jednoj ruti. Tada se ti zadaci nastoje "ubaciti" na optimalno mjesto u neku od postojećih ruta. Osim toga, u toj fazi se iz trenutačnog rješenja miču *skupi* zadaci i umeću na bolje pozicije. Slika 4.1 prikazuje postojeću rutu i zadatak koji je potrebno smjestiti u istu. Budući da ruta ima praznu putanju od čvora *C* do čvora *D*, logično je da je to optimalno mjesto za ubaciti novi zadatak jer on zahtijeva prijevoz upravo između ta dva čvora i vozilo može napraviti taj dio vožnje bez povećanja troškova i posljedično maksimizirati svoju LDR mjeru. Ovaj zadatak bi se smjestio upravo na ovaj način u fazi intenzifikacije. S druge strane, tijekom faze diverzifikacije, algoritam uklanja i kvalitetne dijelove rješenja te je moguće da dotični zadatak ne bi bio smješten na optimalnu poziciju. Ideja je da će tako algoritam moći izbjeći stagnaciju u lokalnom optimumu. Tablica 4.4 daje potpuni pregled svih korištenih operatora korištenih za kreaciju susjedstva. Ovisno o parametrima u datom trenutku, isti operator može se drugačije ponašati.

Diverzifikacija rješenja također je važan dio implementacije, pogotovo u trenucima stagnacije, pa vrijedi spomenuti detalje te faze algoritma. Najprije, diverzifikacija se implicitno događa tijekom cijelog izvršavanja algoritma jer se operatori za generiranje susjedstva biraju nasumično. Testiranja su pokazala da taj pristup nudi najmanje šanse stagnacije u lokalnom optimumu jer se konstantno iz rješenja miču raznovrsne komponente (*skupi* i *jeftini* zadaci, *pune* i *kratke* rute,...). Svejedno, u određenom trenutku dogodi se stagnacija. S obzirom na uzak prostor pretraživanja uzrokovan brojnim ograničenjima, to je normalno. U tim slučajevima, nastupa faza diverzifikacije u sklopu koje algoritam zaustavlja pretragu u lokalnom susjedstvo i nastoji istražiti susjedne prostore veće udaljenosti. Navedeno se postiže promjenom određenih parametara koji smanjuju intenzifikaciju pretrage. Konkretno, u ovoj fazi radi se puno više izmjena trenutačnog rješenja kako bi se generiralo susjedno rješenje. Nadalje, odabir susjednog rješenja nije pohlepan, ne bira se najbolje generirano rješenje. Osim toga, u toj fazi značajnu ulogu igra *tabu lista* koja ne dozvoljava da se kao trenutačno rješenje bira neko koje je trenutačno na toj listi. *Tabu pretraga* dobro je istražena metoda koja značajno povećava efikasnost pretrage i stoga se često ukomponira s drugim algoritmima.

Tablica 4.4: Pregled operatora korištenih u sklopu algoritma.

Oznaka	Opis
BRR	Ukloni nasumično odabranu rutu
BER	Ukloni najprazniju rutu
BFR	Ukloni najpuniju rutu
BSR	Ukloni najkraću rutu
BLR	Ukloni najdužu rutu
RRT	Ukloni nasumično odabrani zadatak
RRST	Ukloni nasumično odabrani zadatak najkraće rute
RCT	Ukloni najskuplji zadatak
RCHT	Ukloni najjeftiniji zadatak
RDT	Ukloni nepovezani zadatak
ITR	Umetni zadatak na nasumično mjesto
ITO	Umetni zadatak na optimalno mjesto

Zbog svoje jednostavnosti, ne čini veliko povećanje složenosti algoritma, a značajno povećava performanse. Naime, predloženi algoritam bez implementacije tabu liste često stagnira i sporije napreduje prema globalnom optimumu. Konkretno, testiranja su pokazala da bez tabu liste algoritam rezultira lošijim završnim rezultatima. Na stvarnim instancama rješenje bez tabu liste u prosjeku je 1.21% lošije, dok je na umjetno generiranim instancama taj postotak 0.98%.

5. Eksperimenti i rezultati

Testiranje algoritma provedeno je na računalu s Intelovim i5-8250 procesorom, 1.6 GHz i 8 Gb memorije koristeći spomenute instance. U sljedećim tabličnim pregledima razmatrat će se reducirane instance veličine 25% i potpune instance (veličine 100%). Razlog za promatranje manjih instanci je mogućnost usporedbe s rezultatima egzaktnih metoda (CPLEX rješavač).

Za početak, budući da je ranije opisana efikasnost izvođenja algoritma što je djelomično uzrokovano paralelizacijom algoritma, vrijedi usporediti ponašanje paralelizirane i neparalelizirane varijante VNTS algoritma. Tablice 5.1 i 5.2 prikazuju najbolja i prosječna rješenja oba pristupa, kao i standardnu devijaciju. Za usporedbu nisu korištene sve instance, već samo odabrani predstavnici svih vrsta instanci. Tako su odabrane po dvije stvarne instance s četiri, šest i osam smjena te po jedna umjetno generirana instanca svake pojedine konfiguracije. Tako se može dobiti pravedan uvid u korisnost paralelizacije. Zadnji redak svake tablice prikazuje prosjek rezultata nad korištenim podskupom instanci.

Tablica 5.1: Usporedba paraleliziranog i neparalleliziranog VNTS algoritma na podskupu stvarnih instanci veličine 100%. Svako od 20 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

	VNTS-paralelno			VNTS-slijedno		
Instanca	Najbolji rezultat	Prosjek	σ	Najbolji rezultat	Prosjek	σ
NP4-1	84.27%	83.20%	1.03%	83.05%	81.58%	0.71%
NP4-2	69.61%	68.06%	0.49%	67.36%	66.73%	0.26%
NP6-1	77.57%	76.68%	0.96%	75.42%	74.34%	0.72%
NP6-2	70.88%	70.03%	0.72%	69.78%	68.70%	0.47%
NP8-1	71.97%	71.21%	0.50%	69.90%	68.50%	0.75%
NP8-2	73.34%	72.95%	0.34%	72.67%	71.78%	0.58%
PROSJEK	74.61%	73.69%	0.67%	73.03%	71.98%	0.59%

Tablica 5.2: Usporedba paraleliziranog i neparaleliziranog VNTS algoritma na podskupu umjetno generiranih instanci veličine 100%. Svako od 20 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

	VNTS-paralelno			VNTS-slijedno		
Instanca	Najbolji rezultat	Prosjek	σ	Najbolji rezultat	Prosjek	σ
LB4-1	75.37%	74.68%	0.47%	74.13%	73.12%	0.74%
TB4-3	72.92%	71.87%	0.44%	72.40%	71.39%	0.61%
LU4-5	64.65%	64.19%	0.36%	63.29%	62.56%	0.41%
TU4-7	55.95%	55.39%	0.26%	55.07%	54.72%	0.20%
LB8-1	89.72%	88.02%	0.85%	85.79%	84.43%	0.93%
TB8-3	70.75%	69.45%	0.61%	69.05%	68.03%	0.64%
LU8-5	67.66%	66.29%	0.51%	65.36%	64.39%	0.40%
TU8-7	54.98%	54.09%	0.57%	52.81%	52.30%	0.44%
PROSJEK	69%	68%	0.51%	67.24%	66.44%	0.55%

Očito je da je paralelizacija dobra za algoritam jer su rezultati paralelne varijante bolji. Najbolja i prosječna rješenja bolja su najmanje za 0.48% u slučaju paralelizacije algoritma u usporedbi sa algoritmom koji nije paraleliziran. U prosjeku je ta razlika 1.75% na stvarnim instancama, a 1.63% na umjetno generiranim instancama. Iako se to možda ne čini kao puno, u slučaju velikih instanci to je značajno poboljšanje koje može značiti veliku uštedu resursa. U svakom slučaju, bilo kakvo poboljšanje je dobrodošlo te je zato i paralelizacija implementirana u završni algoritam.

Tablice 5.3 i 5.4 prikazuju performanse algoritma nad smanjenim instancama. Oznaka '-' navedena je kraj svake instance čija gornja granica nije poznata ili kod koje CPLEX nije uspio pronaći izvedivo rješenje u zadanom vremenu. Usporedivši dobivena rješenja s onima dobivenim korištenjem CPLEX-a, vidljivo je da su u skoro svim slučajeva rezultati znatno bolji, a i dobiveni su u puno kraćem periodu. Naime, vremensko ograničenje za CPLEX u ovom slučaju jest 24 sata, dok je izvršavanje predloženog algoritma bilo ograničeno na jedan sat. Dakle, možemo zaključiti da je pristup bolji od trenutačno najboljih rješavača. Ova usporedba napravljena je samo kako bi se naglasila prednost nad postojećim deterministčkim pristupima. Dalje je potrebno analizirati kako algoritam radi na instancama veličine 100% kako bi se uvjerili u njegovu kvalitetu.
Tablice 5.5 i 5.6 prikazuju kvalitetu algoritma na stvarnim, odnosno umjetno generiranim instancama veličine 100%. Predložena implementacija VNTS algoritma ne može pronaći gornju granicu na niti jednoj instanci, iako je na određenim primjerima pronađeno rješenje

jako blizu te granice. Međutim, algoritam se ispostavio kao stabilan i pronalazi relativno dobra rješenja, i to relativno brzo.

Detaljnim testiranjem pokazalo se da jača diverzifikacija pri kreaciji susjedstva ne rezultira nužno boljim završnim rezultatom. Štoviše, pokazalo se da je presudno pretraživati prostor koristeći male promjene u rješenju jer se tako može detaljno istražiti lokalni prostor. Ako se za kreaciju susjedstva koriste jaki operatori koji puno mijenjaju postojeće rješenje, ne može se postići dobar balans između diverzifikacije i istraživanja lokalnog prostora. Osim toga, broj slojeva susjedstva koji se generiraju jedan je od najvažnijih parametara predloženog algoritma. Naime, pokazalo se da veliki broj slojeva značajno usporava algoritam koji tada nema priliku detaljno istražiti lokalno susjedstvo i samim time rezultira lošijim rješenjima. Stoga se veći broj slojeva koristi samo u fazi nakon stagnacije.
Uspoređujući kvalitetu dobivenih rješenja na svim dostupnim tipovima instanci, može se zaključiti da je algoritam relativno robustan, on jednako dobro radi i na usko ograničenim instancama kao i na labavo raspoređenim zadacima. Navedeno pokazuje da se isplati inicijalno ažuriranje parametara te da ono radi dobro. Međutim, vidljivo je da na problemima stvarne veličine algoritam bolje radi na stvarnim instancama, nego na umjetno generiranim.

Tablica 5.3: Performanse VNTS algoritma na stvarnim instancama veličine 25%. Svako od 30 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

Instanca	Najbolji rezultat	Prosjek	σ	Gornja granica	CPLEX (24h)
NP4-1	85.05%	83.55%	0.60%	92.36%	78.36%
NP4-2	65.05%	64.67%	0.19%	97.04%	65.14%
NP4-3	72.68%	71.88%	0.57%	100%	64.83%
NP4-4	63.28%	62.77%	0.32%	97.72%	54.39%
NP4-5	76.85%	76.51%	0.27%	100%	-
NP6-1	80.43%	79.68%	0.61%	-	-
NP6-2	70.33%	69.25%	0.60%	-	-
NP6-3	65.98%	65.80%	0.10%	95.20%	54.30%
NP6-4	79.55%	78.69%	0.57%	-	-
NP6-5	85.29%	84.19%	0.52%	98.39%	66.11%
NP8-1	75.66%	74.53%	0.60%	98.98%	-
NP8-2	79.04%	77.87%	0.56%	100%	-
NP8-3	77.60%	77.05%	0.37%	100%	-
NP8-4	62.2%	62.01%	0.10%	-	-
NP8-5	72.37%	71.99%	0.24%	100%	-

Tablica 5.4: Performanse VNTS algoritma na umjetno generiranim instancama veličine 25%. Svako od 30 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

Instanca	Najbolji rezultat	Prosjek	σ	Gornja granica	CPLEX (24h)
LB4-1	78.35%	74.78%	1.08%	100%	66.62%
LB4-2	84.56%	82.67%	0.66%	94.87%	76.41%
TB4-3	72.17%	71.01%	0.68%	86.31%	69.91%
TB4-4	76.86%	75.55%	0.85%	83.51%	69.30%
LU4-5	69.38%	63.40%	1.79%	79.94%	-
LU4-6	59.21%	58.35%	0.45%	73.90%	58.65%
TU4-7	52.38%	52.29%	0.06%	52.17%	50.37%
TU4-8	56.32%	55.98%	0.12%	66.38%	55.36%
LB8-1	89.58%	88.116%	0.71%	100%	-
LB8-2	87.7%	85.726%	0.72%	100%	-
TB8-3	70.45%	67.76%	0.89%	82.33%	56.85%
TB8-4	72.83%	71.35%	1.14%	88.75%	52.40%
LU8-5	66.15%	65.42%	0.42%	78.33%	57.42%
LU8-6	65.33%	64.68%	0.43%	86.64%	-
TU8-7	55.64%	54.90%	0.45%	71.59%	47.65%
TU8-8	52.89%	52.56%	0.14%	70.43%	50.74%

5.1. Usporedba sa suvremenim pristupima

Dobar uvid u kvalitetu algoritma može se dobiti usporedbom s najboljim pristupima korištenim u suvremeno doba. Pristup koji ima dobre performanse predstavljen u [5] kreiran za potrebe istog problema također se zasniva na VNS algoritmu i ima oznaku VNS-RLS. Značajna izmjena u usporedbi s klasičnim VNS-om stoji u analizi izvedivosti okolnih rješenja i adaptaciji parametara na temelju kvalitete te okoline. Algoritam tako informaciju o okolini koristi za usmjeravanje pretraživanja što povećava njegovu efikasnost, ali i možda važnije, stabilnost. VNTS uspoređen je s algoritmom VNS-RLS u tablici 5.7 i tablici 5.8. Važna napomena jest razlika u vremenu izvođenja. Iz rada nije jasno koja je točno granica za vrijeme izvođenja algoritma (vrijeme, broj evaluacija/iteracija,...), no očito je da je trajanje algoritma VNS-RLS bilo puno dulje (dok je VNTS izvršavan s limitom od 3,600 sekundi, najkraće prosječno vrijeme izvršavanja algoritma VNS-RLS jest preko 100,000 sekundi). To je velika prednost predložene implementacije, no očito je da su rezultati podjednaki.

Naime, na većini instanci rezultati su jako slični. Razlika između pronađenog rješenja i gornje granice ponekad je čak i 35%. Ni jedan algoritam ne može usmjeriti pretraživanje

Tablica 5.5: Performanse VNTS algoritma na stvarnim instancama veličine 100%. Svako od 30 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

Instanca	Najbolji rezultat	Prosjek	σ	Gornja granica
NP4-1	84.27%	83.20%	1.03%	90.43%
NP4-2	69.61%	68.06%	0.49%	70.23%
NP4-3	74.84%	73.95%	0.59%	79.58%
NP4-4	69.39%	68.76%	0.35%	73.72%
NP4-5	78.75%	78.07%	0.41%	81.20%
NP6-1	77.57%	76.68%	0.96%	83.93%
NP6-2	70.88%	70.03%	0.72%	76.67%
NP6-3	65.68%	65.30%	0.29%	66.90%
NP6-4	79.09%	78.71%	0.33%	80.97%
NP6-5	80.79%	80.35%	0.44%	84.30%
NP8-1	71.97%	71.21%	0.50%	77.04%
NP8-2	73.34%	72.95%	0.34%	77.55%
NP8-3	75.30%	74.54%	0.40%	78.82%
NP8-4	61.80%	61.58%	0.12%	62.53%
NP8-5	73.61%	73.21%	0.27%	76.09%

kroz ovaj veliki i usko ograničen prostor pretraživanja pa oba algoritma imaju poteškoća s pronalaskom optimalnog rješenja. S druge strane, na određenim instancama algoritmi mogu relativno jednostavno pronaći rješenje koje je samo 1% lošije od optimalnog rješenja.
Što se tiče cjelokupne kvalitete rješenja na cijelom skupu instanci, pristupi su podjednaki na velikoj većini instanci, veće razlike su očite samo na nekolicini instanci. Nažalost, nije moguće odrediti tip instanci (uske/labave, (ne)balansirane, veličina,...) na kojima se događaju ove razlike. Sve u svemu, kvaliteta rješenja dobivenog algoritmom VNTS usporediva je s onom dobivenom algoritmom VNS-RLS. Oba algoritma pronalaze bolje najbolje rješenje u otprilike 50% slučajeva. Nadalje, prosječan rezultat je također podjednak, VNTS ima bolji prosjek na otprilike 50% instanci.

Razlika između ova dva algoritma malo je očitija ako se razmatra njihova stabilnost. VNTS algoritam ispada manje stabilan što se može objasniti sofisticiranijom analizom okoline algoritma VNS-RLS. Iako se stabilnosti ova dva algoritma razlikuju i nisu jednake, one prate osobine instance i međusobno koreliraju. Na nekim instancama je VNS-RLS malo stabilniji, a na drugima je to VNTS, ali te razlike nikada nisu jako velike.

Tablica 5.6: Performanse VNTS algoritma na umjetno generiranim instancama veličine 100%. Svako od 30 pokretanja imalo je maksimalno vrijeme izvođenja 60 minuta. σ označava standardnu devijaciju.

Instanca	Najbolji rezultat	Prosjek	σ	Gornja granica
LB4-1	75.37%	74.68%	0.47%	79.47%
LB4-2	78.23%	76.91%	0.55%	86.33%
TB4-3	72.92%	71.87%	0.44%	84.05%
TB4-4	71.46%	70.04%	0.53%	88.74%
LU4-5	64.65%	64.19%	0.36%	74.11%
LU4-6	66.21%	64.63%	0.63%	74.47%
TU4-7	55.95%	55.39%	0.26%	64.05%
TU4-8	56.42%	56.23%	0.13%	63.50%
LB8-1	89.72%	88.02%	0.85%	98.26%
LB8-2	85.11%	83.42%	0.99%	97.97%
TB8-3	70.75%	69.45%	0.61%	87.06%
TB8-4	71.25%	70.33%	0.68%	92.44%
LU8-5	67.66%	66.29%	0.51%	74.27%
LU8-6	66.91%	66.36%	0.33%	71.36%
TU8-7	54.98%	54.09%	0.57%	70.29%
TU8-8	54.09%	53.98%	0.13%	56.54%

Tablica 5.7: Usporedba performansi VNTS algoritma i VNS-RLS algoritma opisanog u drugom radu nad stvarnim instacama veličine 100%. Podebljani rezultati na razini svake instance su bolji. σ označava standardnu devijaciju.

	VNTS			VNS-RLS		
Instanca	Najbolji rezultat	Prosjek	σ	Najbolji rezultat	Prosjek	σ
NP4-1	**84.27%**	**83.20%**	1.03%	82.99%	81.88%	**0.53%**
NP4-2	69.61%	68.06%	0.49%	**69.78%**	**69.33%**	**0.19%**
NP4-3	**74.84%**	**73.95%**	0.59%	73.13%	72.11%	**0.53%**
NP4-4	**69.39%**	**68.76%**	**0.35%**	66.76%	66.06%	0.42%
NP4-5	78.75%	78.07%	0.41%	**80.82%**	**80.37%**	**0.16%**
NP6-1	77.57%	76.68%	0.96%	**79.60%**	**78.96%**	**0.43%**
NP6-2	70.88%	70.03%	0.72%	**74.10%**	**73.77%**	**0.23%**
NP6-3	**65.68%**	**65.30%**	0.29%	58.86%	58.39%	**0.21%**
NP6-4	79.09%	78.71%	**0.33%**	**80.19%**	**79.29%**	0.49%
NP6-5	**80.79%**	**80.35%**	**0.44%**	80.15%	78.44%	0.69%
NP8-1	71.97%	71.21%	0.50%	**73.69%**	**73.10%**	**0.26%**
NP8-2	73.34%	72.95%	0.34%	**75.09%**	**74.52%**	**0.32%**
NP8-3	**75.30%**	**74.54%**	0.40%	74.31%	73.77%	0.40%
NP8-4	61.80%	61.58%	0.12%	**61.94%**	**61.85%**	**0.05%**
NP8-5	**73.61%**	**73.21%**	0.27%	73.28%	72.84%	**0.21%**

Tablica 5.8: Usporedba performansi VNTS algoritma i VNS-RLS algoritma opisanog u drugom radu nad umjetno generiranim instacama veličine 100%. Podebljani rezultati na razini svake instance su bolji. σ označava standardnu devijaciju.

	VNTS			VNS-RLS		
Instanca	Najbolji rezultat	Prosjek	σ	Najbolji rezultat	Prosjek	σ
LB4-1	**75.37%**	**74.68%**	**0.47%**	73.57%	72.79%	0.51%
LB4-2	**78.23%**	76.91%	0.55%	78.02%	**77.52%**	**0.37%**
TB4-3	**72.92%**	**71.87%**	**0.44%**	69.52%	68.78%	0.53%
TB4-4	71.46%	70.04%	0.53%	**72.91%**	**72.09%**	**0.51%**
LU4-5	**64.65%**	64.19%	0.36%	64.64%	**64.22%**	**0.24%**
LU4-6	66.21%	64.63%	0.63%	**67.89%**	**67.50%**	**0.26%**
TU4-7	**55.95%**	**55.39%**	0.26%	53.07%	52.90%	**0.19%**
TU4-8	**56.42%**	**56.23%**	0.13%	53.78%	53.58%	**0.09%**
LB8-1	**89.72%**	**88.02%**	**0.85%**	85.86%	83.48%	1.46%
LB8-2	85.11%	83.42%	0.99%	**94.94%**	**93.21%**	**0.82%**
TB8-3	**70.75%**	**69.45%**	0.61%	69.41%	69.01%	**0.29%**
TB8-4	**71.25%**	**70.33%**	**0.68%**	66.08%	65.24%	0.81%
LU8-5	67.66%	66.29%	**0.51%**	**67.95%**	**67.24%**	0.52%
LU8-6	66.91%	66.36%	0.33%	**68.40%**	**67.87%**	**0.29%**
TU8-7	54.98%	54.09%	0.57%	**59.72%**	**59.31%**	**0.28%**
TU8-8	54.09%	53.98%	0.13%	**54.36%**	**54.23%**	**0.12%**

6. Zaključak

U sklopu ovog rada detaljno je istražena tema pretraživanja varijabilnog susjedstva za rješavanje problema usmjeravanja vozila u pomorskim lukama. S ciljem razvoja što boljeg algoritma, istražena je relevantna literatura kako bi se detaljno istražio problem i proučile postojeće metode za rješavanje istog. Utvrđeno je da već postoje jako dobri pristupi koji nude zadovoljavajuća rješenja implementacijom dobro istraženih pristupa. Međutim, ne postoji veliki broj OVRPTW modela pa je ovo svakako područje gdje postoji prilika za razvoj. Nadalje, problem je matematički formuliran kako bi bila očita kompleksnost rješavanja uzrokovana usko ograničenim i velikim prostorom pretraživanja. Stoga je utvrđeno da je potreban algoritam koji će smisleno usmjeravati pretraživanje.
Predložen je efikasan algoritam pretraživanja varijabilnog susjedstva s tabu listom za rješavanje navedenog optimizacijskog problema. Algoritam je razvijen s brzinom kao primarnim zahtjevom jer je utvrđeno da brz algoritam može puno pomoći cjelokupnom sustavu. Koristeći instance reducirane veličine, zaključeno je da VNTS pronalazi bolja rješenja od postojećih deterministčkih metoda na skoro svim instancama, i to u puno kraćem vremenu. Osim toga, korištene su stvarne i umjetno generirane instance potpune veličine s ciljem analize kvalitete algoritma na problemima realističnih razmjera. Pokazalo se da VNTS algoritam može biti iznimno efikasan čak i na iznimno velikim instancama. Na određenim vrstama instanci pokazao se boljim i od sličnih algoritama u području. Te razlike nisu značajne, ali s obzirom na brzinu izvršavanja algoritma VNTS, predloženi pristup se nameće kao kvalitetan za rješavanje OVRPTW problema. Međutim, razlika između gornje granice i pronađenih rješenja i dalje je značajna na brojnim instancama i za samo neke od njih algoritam pronalazi rješenje unutar 1% od optimuma.

Dakle, potrebna su značajna poboljšanja koja bi mogla pomoći u pronalasku optimalnog rješenja. Iako se čini da VNTS dobro balansira između intenzifikacije i diverzifikacije, izgledno je da su potrebni jači operatori u fazi diverzifikacije koji bi mogli napraviti veće izmjene rješenja s ciljem pronalaska optimuma. Može biti da je za kvalitetnije rješenje potrebno smisliti naprednije heuristike za usmjeravanje pretraživanja. Osim toga, možda je korisno koristiti neke druge indikatore, osim geografske udaljenosti gradova, koji bi smisle-

nije razmještavali zadatke.

Pokazalo se da implementacijom algoritma u efikasnom programskom jeziku s paralelizacijom koda i implementacijom *cache-a* algoritam može biti jako brz. To može biti od velike koristi ako je za sustav potrebno dinamičko ažuriranje ruta uzrokovano neplaniranim izmjenama u horizontu. Osim toga, jednostavno upravljanje parametrima koje značajno ne usporava izvršavanje algoritma pokazalo se kao adekvatan pristup koji balansira između istraživanja jako velikog prostora stanja i detaljnog istraživanja lokalnog susjedstva. Nadalje, lokalna pretraga i tabu lista značajno povećavaju efikasnost sustava te su svakako korisni pristupi za rješavanje problema koji imaju veliki prostor pretraživanja.
Zaključno, algoritam VNTS adekvatan je pristup za rješavanje OVRPTW problema, ali svakako postoje prilike za poboljšanje kako bi se efikasnije pokušalo pronaći optimalno rješenje.

LITERATURA

1. Službena stranica Ningbo luke: https://www.nbport.com.cn/gfww/

2. Učinak nepogoda na promjene u usmjeravanju vozila: https://icecargo.com.au/weather-impacts-shipping/

3. Dantzig, G. B., and J. H. Ramser. "The Truck Dispatching Problem." Management Science, vol. 6, no. 1, 1959, pp. 80–91. JSTOR, http://www.jstor.org/stable/2627477. Accessed 12 June 2023.

4. Učinak kvalitetne implementacije algoritma za rješavanje problema usmjeravanja vozila: https://commission.europa.eu/eu-regional-and-urban-development/topics/cities- and-urban-development/city-initiatives/smart-cities

5. B. Chen, R. Qu, R. Bai, W. Laesanklang. A Reinforcement Learning Based Variable Neighborhood Search Algorithm for Open Periodic Vehicle Routing Problem with Time Windows. 2018.

6. J. Chen, R. Bai, R. Qu, G. Kendall, A Task Based Approach for A Real-World Commodity Routing Problem, 2013 IEEE Symposium Series on Computational Intelligence (SCCI 2013), 16-19 Travanj, Singapur

7. Ibrahim, Abdullahi & Abdulaziz, Rabiat & Ishaya, Jeremiah & Sowole, Samuel. (2019). Vehicle Routing Problem with Exact Methods. 5-15. 10.9790/5728-1503030515.

8. Saadatseresht, Mohammad & Mansourian, Ali & Taleai, Mohammad. (2009). Evacuation Planing Using Multiobjective Evolutionary Optimization Approach. European Journal of Operational Research. 198. 305-314. 10.1016/j.ejor.2008.07.032.

9. Jeong, Kiyoung & Hong, Jae-Dong & Xie, Yuanchang. (2014). Design of emergency logistics networks, taking efficiency, risk and robustness into consideration. International Journal of Logistics. 17. 10.1080/13675567.2013.833598.

10. Tatomir, B., Rothkrantz, L. (2006). Ant Based Mechanism for Crisis Response Coordination. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds) Ant Colony Optimization and Swarm Intelligence. ANTS 2006.

11. Yi, Wei & Kumar, Arun. (2007). Ant Colony Optimization for Disaster Relief Operations. Transportation Research Part E: Logistics and Transportation Review. 43. 660-672. 10.1016/j.tre.2006.05.004.

12. Pichpibul, Tantikorn & Kawtummachai, Ruengsak. (2012). An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem. ScienceAsia. 38. 307. 10.2306/scienceasia1513-1874.2012.38.307.

13. Campbell, Ann Melissa, and Martin Savelsbergh. "Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems." Transportation Science, vol. 38, no. 3, 2004, pp. 369–78. JSTOR, http://www.jstor.org/stable/25769207. Accessed 12 June 2023.

14. Nazari, Mohammadreza & Oroojlooy jadid, Afshin & Snyder, Lawrence & Takáč, Martin. (2018). Deep Reinforcement Learning for Solving the Vehicle Routing Problem.

15. Nazari, Mohammadreza & Oroojlooy, Afshin & Snyder, Lawrence V. & Martin Takáč. (2018). Reinforcement Learning for Solving the Vehicle Routing Problem.

16. Kovács, László & Agárdi, Anita & Bányai, Tamás. (2020). Fitness Landscape Analysis and Edge Weighting-Based Optimization of Vehicle Routing Problems. Processes. 8. 1363. 10.3390/pr8111363.

17. Chen, Binhui & Qu, Rong & Ishibuchi, Hisao. (2017). Variable-Depth Adaptive Large Neighbourhood Search Algorithm for Open Periodic Vehicle Routing Problem with Time Windows.

18. Jianjun Chen, Ruibin Bai, Haibo Dong, R. Qu and G. Kendall, "A dynamic truck dispatching problem in marine container terminal," 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8, doi: 10.1109/SSCI.2016.7850081.

19. Chen, B., Qu, R., Bai, R. et al. A hyper-heuristic with two guidance indicators for bi-objective mixed-shift vehicle routing problem with time windows.

20. Fajardo, Mario & Sudholt, Dirk. (2021). Self-Adjusting Population Sizes for Non-Elitist Evolutionary Algorithms: Why Success Rates Matter.

21. Nalepa, Jakub & Blocho, Miroslaw. (2018). Parameter-less (meta)heuristics for vehicle routing problems. 27-28. 10.1145/3205651.3208215.

22. Chen, Binhui & Qu, Rong & Bai, Ruibin & Ishibuchi, Hisao. (2016). A Variable Neighbourhood Search Algorithm with Compound Neighbourhoods for VRPTW. 25-35. 10.5220/0005661800250035.

23. Bai, Ruibin & Kendall, Graham & Qu, Rong & Atkin, Jason. (2012). Tabu assisted guided local search approaches for freight service network design. Information Sciences. 189. 266–281. 10.1016/j.ins.2011.11.028.

Rješavanje problema usmjeravanja vozila u pomorskim lukama

Sažetak

Usmjeravanje vozila često je prisutan element mnogih industrija pa tako i morskih luka. Ovaj problem predstavlja ogroman logistički izazov zbog velike količine robe koja se mora prevesti. Kvalitetno rješenje navedenog problema može rezultirati velikom uštedom sredstava. Postoje različiti pristupi rješavanju ovog problema, no i dalje je ovo česta tema istraživanja zbog rastuće složenosti uzrokovane povećanjem industrija. Evolucijski algoritmi igraju veliku ulogu u ovom području jer nude obećavajuće rezultate. Međutim, i dalje ima puno prostora za poboljšanja i istraživanje novih, efektivnijih metoda. Ovaj rad opisuje problem usmjeravanja vozila u morskim lukama i predlaže prikladan evolucijski algoritam koji nastoji kreirati optimalan skup ruta za vozila. Opisana je korištena metoda te analizirani rezultati u usporedbi s relevantnim suvremenim pristupima.

Ključne riječi: metaheuristika, evolucijski algoritam, optimizacija, problem usmjeravanja vozila, problem raspoređivanja zadataka

Solving the vehicle routing problem in maritime ports

Abstract

Vehicle routing is an often present element of many industries, including seaports. This problem presents a huge logistical challenge due to the large amount of goods that have to be transported. A high-quality solution to the above-mentioned problem can result in a large saving of funds. There are different approaches to solving this problem, but it is still a frequent topic of research due to the growing complexity caused by the increasing demands of industries. Evolutionary algorithms play a major role in this field as they offer promising results. However, there is still a lot of room for improvement and research into new, more effective methods. This paper describes the vehicle routing problem in seaports and proposes a suitable evolutionary algorithm that seeks to create an optimal set of vehicle routes. The method used is described and the results analyzed in comparison with relevant modern approaches.

Keywords: metaheuristics, evolutionary algorithm, optimization, vehicle routing problem, task scheduling problem