Primjena problema preusmjeravanja vozila s vremenskim ograničenjima u uslugama na zahtjev

Smolić-Ročak, Magda

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:036691

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 503

PRIMJENA PROBLEMA PREUSMJERAVANJA VOZILA S VREMENSKIM OGRANIČENJIMA U USLUGAMA NA ZAHTJEV

Magda Smolić-Ročak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 503

PRIMJENA PROBLEMA PREUSMJERAVANJA VOZILA S VREMENSKIM OGRANIČENJIMA U USLUGAMA NA ZAHTJEV

Magda Smolić-Ročak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 503

Pristupnica:	**Magda Smolić-Ročak (0036519844)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Bojan Trkulja

Zadatak: **Primjena problema preusmjeravanja vozila s vremenskim ograničenjima u uslugama na zahtjev**

Opis zadatka:

Tržište rada obuhvaća velik broj poslova I izvođača s različitim vještinama, iskustvom i preferencijama. Dinamičnost tržišta rada, dostupnost radne snage na lokaciji, uvjeti rada, trajanje posla samo su neki od činitelja koji uvećavaju složenost procesa uparivanja zadatka na lokaciji i izvođača. U okviru ovog rada bit će razvijen pristup zakazivanju termina radnika na različitim lokacijama u gradu prilagođavanjem načela vremenski ograničenog problema usmjeravanja vozila (TCVRP). Problem se formulira temeljem lokacija rada uključujući vremenska ograničenja, koja se provode u model planiranja koji uključuje varijable odlučivanja za zadatke radnika unutar vremenskog razdoblja i na lokaciji. Implementacija odabranog algoritma daje rješenje za planiranje koje dodjeljuje radnike lokacijama s određenim vremenskim ograničenjima te se koristi za učinkovito upravljanje uslugama na zahtjev.

Rok za predaju rada: 28. lipnja 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 503

PRIMJENA PROBLEMA USMJERAVANJA VOZILA S VREMENSKIM OGRANIČENJIMA U USLUGAMA NA ZAHTJEV

Magda Smolić - Ročak

Zagreb, rujan 2024.

Sadržaj

1. Uvod **3**

2. Teorijski okvir **5**

2.1. Problem usmjeravanja vozila 5

2.2. Rješavanje problema usmjeravanja vozila 10

2.3. Problem usmjeravanja vozila s vremenskim okvirima 11

2.4. Metode za rješavanje problema usmjeravanja vozila s vremenskim okvirima 15

2.4.1. Egzaktne metode za rješavanje VRPTW-a 15

2.4.2. Heurističke metode za rješavanje VRPTW-a 16

2.4.3. Metaheurističke metode za rješavanje VRPTW-a 17

2.4.4. Genetski algoritmi 18

3. Implementacija **19**

3.1. Implementacija statičkog VRPTW-a s ograničenjima radnog vremena . . 20

3.1.1. Korišteni ulazni podatci 21

3.1.2. Formulacija problema 23

3.1.3. Rješavanje problema usmjeravanja vozila s vremenskim okvirima genetskim algoritmom 25

3.1.4. Rezultati 31

3.2. Optimizacija rute: različiti tipovi vremenskih zahtjeva 33

3.2.1. Ulazni podatci 34

3.2.2. Formulacija problema 35

3.2.3. Rješavanje problema genetskim algoritmom 36

3.2.4. Rezultati 39

3.3. Dinamičko dodjeljivanje zadataka u stvarnom vremenu 40
3.3.1. Ulazni podatci 40
3.3.2. Formulacija problema 41
3.3.3. Rješavanje problema genetskim algoritmom 42
3.3.4. Rezultati 47

4. Praktična primjena u aplikacijama na zahtjev 48

5. Zaključak 49

Sažetak 50

Abstract 51

Literatura 52

1. Uvod

U današnjemu dinamičnom okruženju, **usluge na zahtjev** ključne su za učinkovito povezivanje korisnika i pružatelja usluga. Fleksibilnost tih usluga, koje odgovaraju na trenutne potrebe korisnika, čini ih temeljem modernih ekonomskih modela, od prijevoza (npr. Uber) do kućanskih poslova. Korisnici zatraže uslugu u stvarnom vremenu, dok pružatelji potom moraju optimalno rasporediti posao na zaposlenike. Upravo zbog te potrebe za brzim i učinkovitim ispunjenjem zahtjeva, optimizacija ruta postaje ključna: rješavanjem problema usmjeravanja vozila (**VRP**) minimiziraju se troškovi odnosno maksimizira dobit te se pronalaze efikasna rješenja polaznog problema.

Ovaj diplomski rad bavi se **teorijskom i praktičnom primjenom** problema usmjeravanja vozila s vremenskim ograničenjima (engl. Vehicle Routing Problem with Time-Windows, skraćeno **VRPTW**) u kontekstu usluga na zahtjev. **Usluge na zahtjev** (engl. on-demand services) predstavljaju vrstu usluga kod kojih se dostupne **pružatelje usluga** (radnike) raspoređuje za obradu svaki put kada neki **primatelj usluge** (klijent) postavi određeni zahtjev. Primatelj usluge zahtjev može postaviti u proizvoljnom trenutku. Takve usluge nalazimo u raznim područjima, a to su, primjerice, poslovi čišćenja, sastavljanja namještaja i podučavanja.

Problem usmjeravanja vozila s vremenskim ograničenjima (VRPTW) predstavlja izazov u mnogim industrijama, a njegovo rješavanje zahtijeva primjenu naprednih metoda optimizacije. Ovaj problem rješava se različitim metodama, među kojima su **egzaktni**, **heuristički** i **metaheuristički** pristup. U ovom radu, poseban naglasak bit će stavljen na primjenu **genetskog algoritma** zbog njegove sposobnosti pretraživanja velikih prostora rješenja i prilagodljivosti dinamičnim uvjetima. Genetskim će se algoritmom stvarati rasporedi za probleme koji oponašaju stvarne uvjete usluga na zahtjev.

Iako primarni cilj ovog rada nije izrada aplikacije za dijeljenje usluga na zahtjev niti izravna integracija algoritma u takvu aplikaciju, ideja za diplomski rad proizašla je iz moje ideje za izradu i rada na aplikaciji **CleanApp**. CleanApp platforma je koja spaja karakteristike Ubera i Airbnba radi primjene na usluge čišćenja. Omogućuje pružateljima i primateljima usluga da se brzo i jednostavno povežu. U ovom radu rješavaju se jednostavniji problemi optimizacije, koji mogu poslužiti kao temelj za složenije algoritme potrebne u stvarnim aplikacijama poput CleanAppa. Prikazano je kako se **VRPTW algoritam** može prilagoditi i nadograditi kako bi poboljšao učinkovitost takvih aplikacija.

2. Teorijski okvir

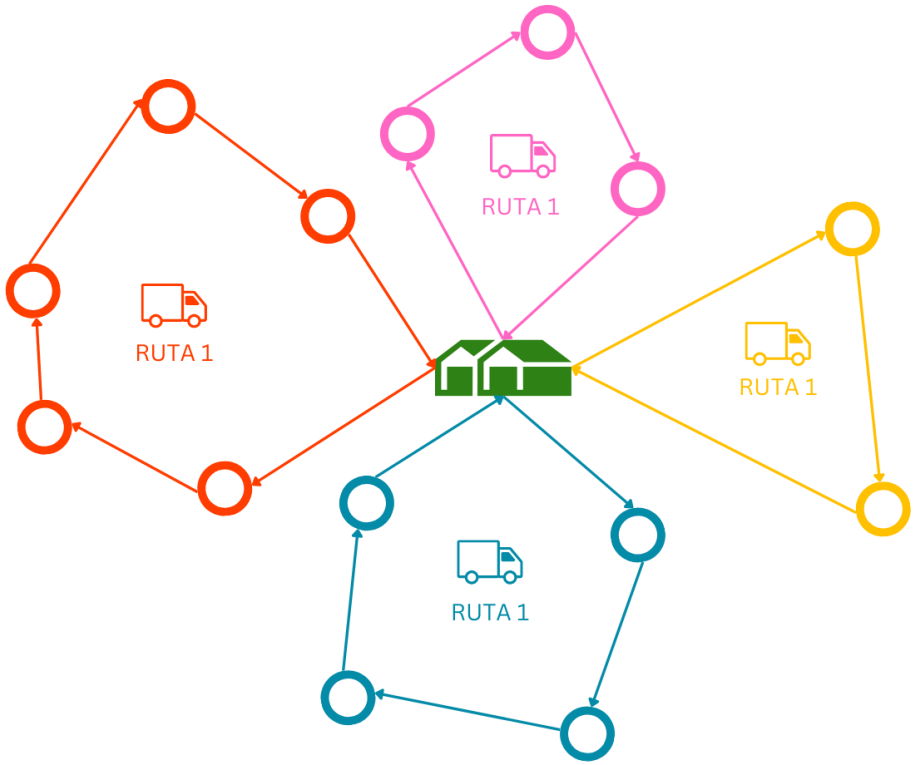
Optimizacija ruta važna je za povećanje efikasnosti i smanjenje troškova u industrijskim i logističkim procesima [1]. S razvojem tehnologije u transportu, proizvođači su prepoznali potrebu za unapređenjem svojih distribucijskih sustava kako bi bolje odgovorili na rastuće zahtjeve tržišta. U tom kontekstu, problem usmjeravanja vozila (VRP) bitan je jer omogućuje pronalaženje optimalnih ruta za isporuku, što rezultira uštedama i poboljšanom operativnom učinkovitošću.

Ovo poglavlje započinje izlaganjem osnovnog problema usmjeravanja vozila (engl. **V**ehicle **R**outing **P**roblem, skraćeno **VRP**), koji je temelj za problem usmjeravanja vozila s vremenskim okvirima (engl. **V**ehicle **R**outing **P**roblem with **T**ime **W**indows, skraćeno **VRPTW**). Zatim se pruža pregled metoda za njihovo rješavanje. Na kraju poglavlja nalaze se scenariji uporabe VRPTW-a.

2.1. Problem usmjeravanja vozila

Problem usmjeravanja vozila (engl. Vehicle Routing Problem, skraćeno **VRP**) optimizacijski je problem minimizacije ukupne udaljenosti koju prelazi određeni broj vozila, uz različita ograničenja, pri čemu svaki klijent mora biti posjećen točno jednom od strane bilo kojeg vozila [2]. VRP jedan je od problema kombinatorne optimizacije i poznato je da je NP-težak [2]. To znači da za većinu praktičnih primjena ne postoji efikasan algoritam koji može pronaći optimalno rješenje u razumnom vremenu za sve moguće instance problema. U praksi se, stoga, često koriste aproksimativni algoritmi koji pronalaze zadovoljavajuća, ali ne nužno optimalna rješenja. Na slici 2.1., slikovito je prikazan primjer rasporeda kod problema usmjeravanja vozila, čime se prethodno vizualno dočarava.

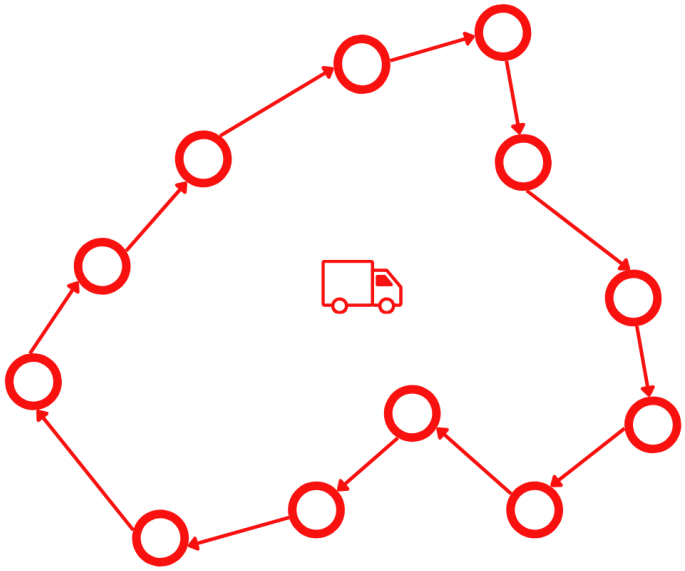
VRP predstavlja generalizaciju problema putujućeg trgovca (engl. Traveling Sale-



Slika 2.1. VRP

sman Problem, skraćeno **TSP**) [3]. TSP odgovara na sljedeće pitanje: ako su nam dostupne udaljenosti između svih gradova i polazni grad, koja je najkraća moguća ruta trgovca kojom on posjećuje svaki grad točno jednom i vraća se u polazni grad? S druge strane, VRP odgovara na pitanje pronalaska optimalnoga skupa ruta za određeni skup vozila kako bi se dostava obavila određenom skupu kupaca. Dakle, iz VRP-a dobije se TSP kada se koristi samo jedno vozilo bez ikakvih drugih ograničenja. Na slici 2.2. vidljiv je primjer TSP-a.

Postoji nekoliko varijanti problema usmjeravanja vozila. Kod problema usmjeravanja vozila s ograničenim kapacitetom (engl. Capacitated Vehicle Routing Problem, skraćeno **CVRP**) vozila imaju ograničeni kapacitet, stoga se nakon obrade određenog broja zahtjeva moraju vratiti u polazno mjesto kako bi si tamo povećali kapacitet na početnu razinu. Problem usmjeravanja vozila s vremenskim okvirima (engl. Vehicle Routing Problem with Time Windows, skraćeno **VRPTW**) nameće vremenski okvir za svakog klijenta unutar kojeg on mora biti posjećen. Kod problema usmjeravanja vozila s preuzi-

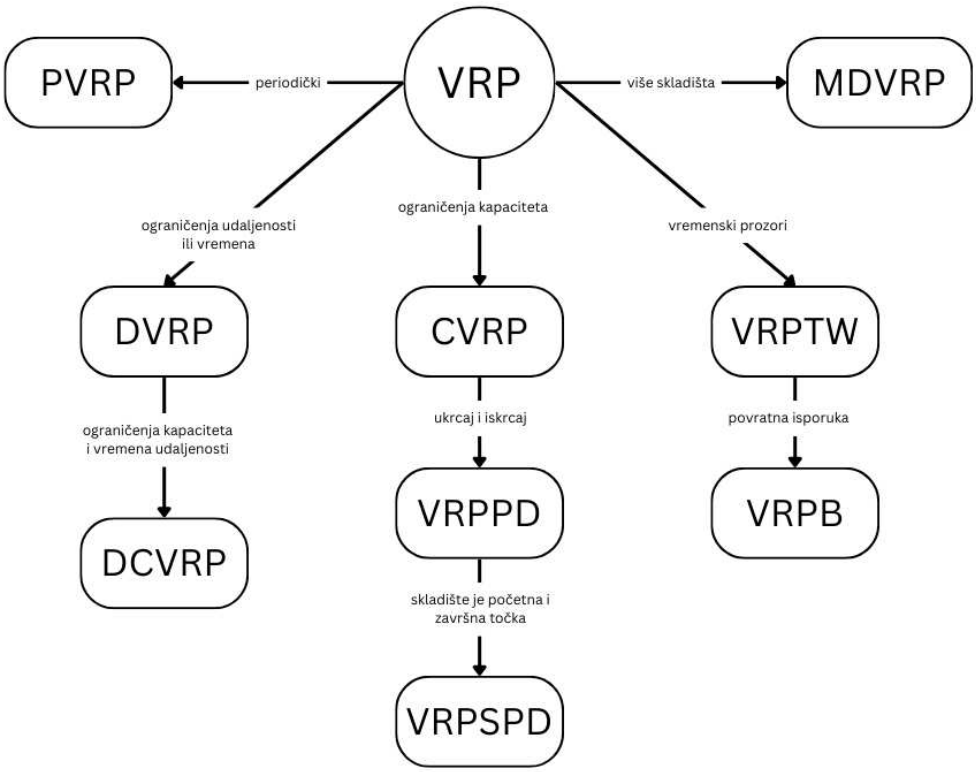


Slika 2.2. TSP

manjem i isporukom (engl. Vehicle Routing Problem with Pickup and Delivery, skraćeno **VRPPD**) vozila moraju svaki paket klijenta prvo pokupiti s određene lokacije, a zatim ga dostaviti na određeno mjesto. Konačno, stohastički problem usmjeravanja vozila (engl. Stochastic Vehicle Routing Problem, skraćeno **SVRP**) uvodi stohastičnost u različite parametre problema, kao što su vrijeme putovanja ili vrijeme obrade određenog zahtjeva, koji su inače konstantni. Spomenute varijante osnovnog problema, kao i one prikazane na slici 2.3., pripadaju najpoznatijima. Slika 2.3. prikazuje hijerarhijski prikaz različitih varijanti problema usmjeravanja vozila (VRP) i kako su one međusobno povezane različitim ograničenjima poput kapaciteta, udaljenosti i vremenskih okvira.

Matematički model problema usmjeravanja vozila

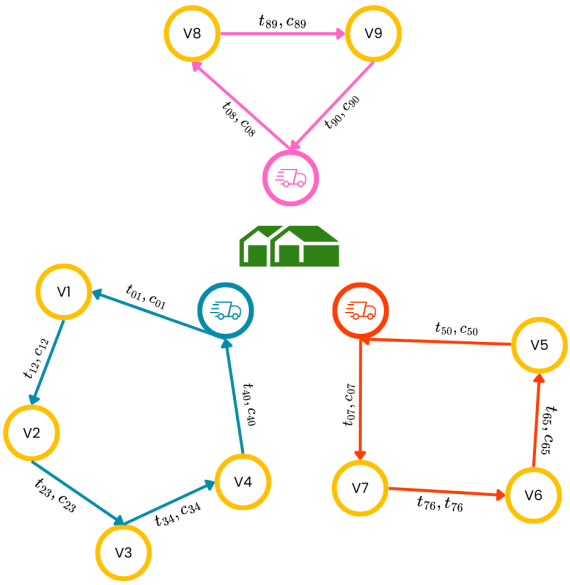
Problem usmjeravanja vozila (VRP) formalno je definiran za flotu vozila označenih s $k \in K$, skup klijenata $c \in C$, i usmjereni graf $G = (V, E)$, gdje V predstavlja skup vrhova (engl. vertices), a E skup lukova (engl. arcs). Graf se sastoji od $|C| + 2$ čvora, gdje su klijenti označeni s $1, 2, \ldots, n$, dok je skladište označeno čvorovima 0 (početni čvor) i $n+1$ (krajnji čvor) [3]. Početni čvor može biti istovremeno i završni, kao što je to obično i u stvarnosti: u tom je slučaju ukupan broj čvorova u grafu zapravo jednak $|C| + 1$.



Slika 2.3. Varijante VRP-a [5]

Optimizacija ruta provodi se tako da se svakom vozilu k dodijeli jedna ruta σ_k, pri čemu je cilj učinkovito iskoristiti dostupna vozila i minimizirati troškove. Svako vozilo treba posjetiti klijente na svojoj ruti σ_k ne prelazeći kapacitet vozila c. Pravovremena je isporuka ključna jer kašnjenja ili preuranjeni dolasci mogu dovesti do gubitaka. Za svaki luk (i, j) na grafu definiran je **trošak** prolaska njime c_{ij} i **vrijeme putovanja** t_{ij}. Dakako, trošak se, između ostalog, određuje iz vremena putovanja.

Na slici 2.4. vidljiv je primjer problema VRP-a s definiranim parametrima. Ovaj graf sadrži ukupno 10 vrhova, pri čemu se 9 vrhova odnosi na klijente V_1 do V_9, dok jedan vrh predstavlja sjedište poduzeća (depot). Uz svaku granu jasno su označeni pripadajući troškovi i vrijeme putovanja. Sva vozila kreću iz skladišta, te se nakon obavljene rute i vraćaju u skladište.



Slika 2.4. VRP

Cilj je problema usmjeravanja vozila minimizirati troškove putovanja, što je formalno izraženo funkcijom cilja u nastavku:

Funkcija cilja

$$\min \quad \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij}^{k} \tag{2.1}$$

gdje:

- c_{ij} predstavlja trošak putovanja između čvorova i i j,
- x_{ij}^{k} binarna je varijabla koja poprima vrijednost 1 ako vozilo k putuje izravno od čvora i do čvora j, a inače 0.

Ograničenja

Uz funkciju cilja, postoje i ograničenja koja je potrebno definirati:

$$\sum_{i \in V} q_i \cdot x_{ij}^{k} \leq c_k, \quad \forall k \in K, \forall j \in V \tag{2.2}$$

Ograničenje (2.2) osigurava da kapacitet c_k vozila k nije prekoračen, pri čemu q_i predstav-

lja količinu tereta koju vozilo k mora prenijeti od klijenta i do klijenta j, a x_{ij}^k predstavlja binarnu varijablu koja pokazuje prelazi li vozilo k iz čvora i u čvor j.

$$\sum_{j \in V} x_{0j}^k = 1 \quad \forall k \in K \tag{2.3}$$

Ograničenje (2.3) osigurava da svako vozilo započne svoju rutu u skladištu.

$$\sum_{i \in V} x_{i,n+1}^k = 1 \quad \forall k \in K \tag{2.4}$$

Ograničenje (2.4) osigurava da svako vozilo završi svoju rutu u skladištu.

Proširivanjem ovoga matematičkog modela VRP-a dobiju se druge obrađene varijante problema VRP-a.

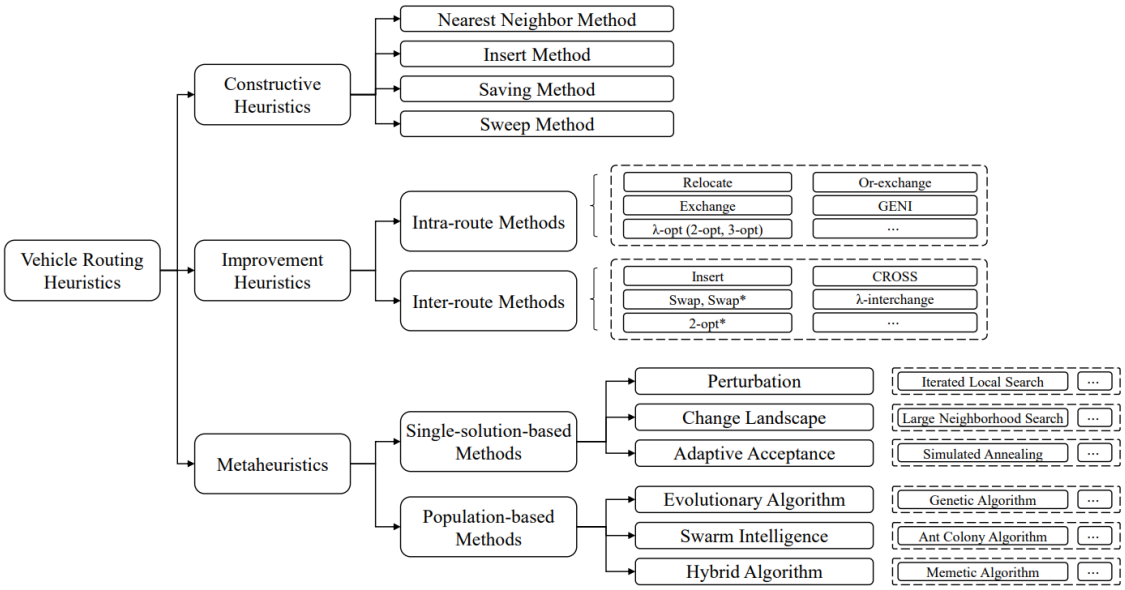
2.2. Rješavanje problema usmjeravanja vozila

Metode rješavanja problema usmjeravanja vozila općenito se mogu podijeliti u dvije glavne skupine:

1. egzaktne metode
2. aproksimacijske metode

Egzaktne metode uvijek pronalaze optimalno rješenje, ali njihovo izračunavanje zbog velike složenosti može trajati jako dugo. Za praktične potrebe od veće je važnosti pronaći dovoljno dobro rješenje, nego riskirati da se zbog dugog trajanja rješenje ne pronađe uopće. Zbog toga se u praksi koriste **aproksimacijske metode**: one umjesto traženja optimalnog rješenja, pronalaze zadovoljavajuće rješenje. Kao podvrste aproksimacijskih metoda razlikujemo heurističke i metaheurističke [4].

Heurističke i metaheurističke metode koriste pristupe za brzo pronalaženje dobrih rješenja unutar ograničenog vremenskog okvira, premda ta rješenja često nisu optimalna. **Heurističke metode** obično se temelje na specifičnim pravilima ili jednostavnim strategijama koje su prilagođene određenom problemu, što ih čini bržim, ali manje fleksibilnim. **Metaheurističke metode**, s druge strane, osmišljene su tako da istražuju



Slika 2.5. Podjela heuristika za VRP [5]

širi prostor rješenja, čime povećavaju vjerojatnost za pronalaženje boljih rješenja i izbjegavanje lokalnih optimuma. Za razliku od heuristika, metaheuristike su općenitije i mogu se primijeniti na širi raspon problema. Metaheuristike su također bolje u balansiranju između eksploatacije (intenzivnog pretraživanja područja oko trenutnih dobrih rješenja) i eksploracije (istraživanja novih obećavajućih područja prostora rješenja), što ih čini učinkovitijima za rješavanje složenijih problema. Na slici 2.5. prikazana je vrlo detaljna podjela korištenih heurističkih i metaheurističkih metoda za rješavanje klasičnoga problema usmjeravanja vozila.

S obzirom na to da je fokus ovog rada na rješavanju problema usmjeravanja vozila s vremenskim okvirima, bit će opisane konkretne egzaktne i aproksimacijske metode koje se prvenstveno koriste za tu inačicu klasičnoga problema usmjeravanja vozila.

2.3. Problem usmjeravanja vozila s vremenskim okvirima

Problem usmjeravanja vozila s vremenskim okvirima (engl. Vehicle Routing Problem with Time Windows, skraćeno **VRPTW**) proširenje je klasičnog problema usmjeravanja vozila (VRP), gdje su uz standardne zahtjeve dodani vremenski okviri unutar kojih se određeni klijent mora posjetiti [5]. Vremenski okviri definiraju intervale unutar kojih

dostava mora biti izvršena, što bitno povećava složenost problema u odnosu na osnovni VRP. Neki od korisnijih primjena VRPTW-a uključuju dostave za banke, poštanske dostave, prikupljanje otpada i planiranje ruta za školske autobuse [3].

Svaki klijent i ima **vremenski okvir** $[t_i^e, t_i^l]$, gdje t_i^e predstavlja najranije moguće vrijeme početka usluge (engl. **earliest start time**), a t_i^l najkasnije dopušteno vrijeme za početak usluge (engl. **last possible start time**). Vozilo mora stići do klijenta prije vremena t_i^l. Može stići i prije vremena t_i^e, ali usluga kod klijenta neće započeti prije tog vremena [3]. Skladište također ima svoj vremenski okvir $[t_0^s, t_0^r]$, gdje t_0^s označava najranije moguće vrijeme izlaska vozila iz skladišta (engl. **earliest departure time**), a t_0^r najkasnije dopušteno vrijeme povratka vozila u skladište (engl. **last possible return time**). Vozila ne smiju napustiti skladište prije vremena t_0^s i moraju se vratiti prije ili u vrijeme t_{n+1}^r [3]. Prikaz opisanog može se vidjeti na slici 2.6., na kojoj se nalazi usmjereni graf s čvorovima i bridovima, uključujući vrijeme putovanja između lokacija t_{ij} i vremenske okvire klijenata $[t_i^e, t_i^l]$ koji prikazuju najranije i najkasnije moguće vrijeme za započinjanje obavljanja usluge.

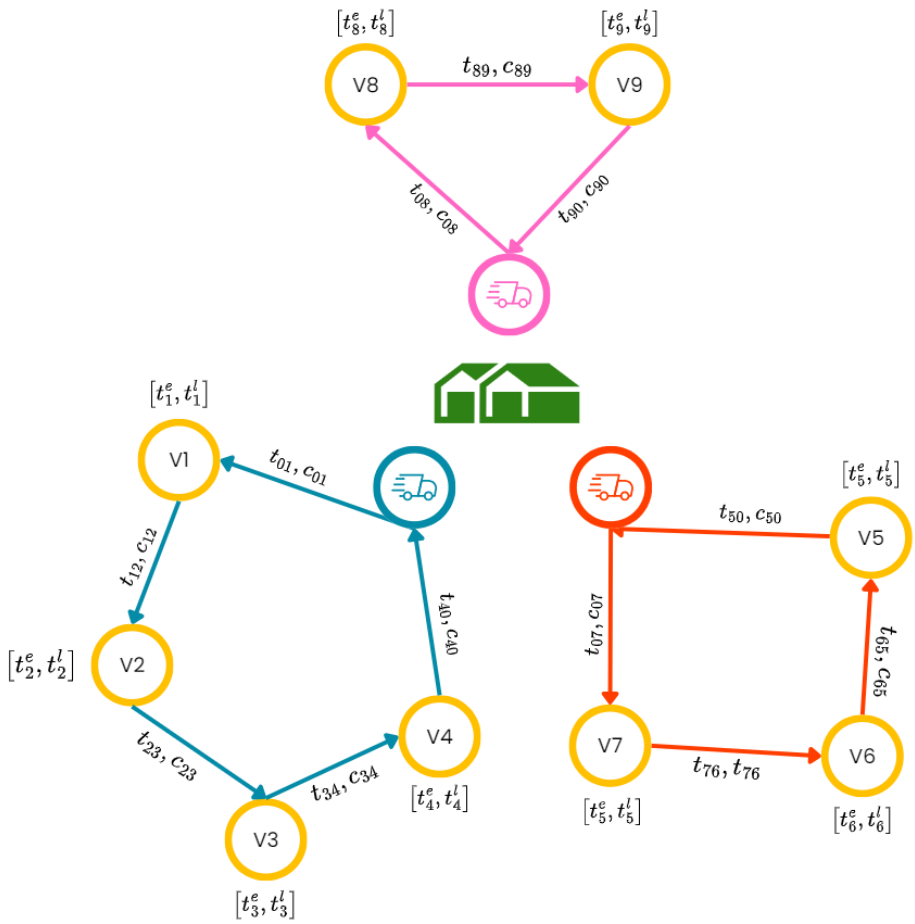
Kao što je već spomenuto, cilj je VRPTW-a optimizirati rute vozila kako bi se minimizirali ukupni troškovi, uključujući prijeđenu udaljenost, vrijeme putovanja i čekanja, uz poštivanje kapaciteta vozila i vremenskih okvira klijenata. **Funkcija cilja** stoga glasi:

$$\min \quad \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ijk} [3] \tag{2.5}$$

gdje c_{ij} predstavlja trošak putovanja između čvorova i i j, a x_{ijk} je binarna varijabla koja poprima vrijednost 1 ako vozilo k putuje direktno od čvora i do čvora j, inače poprima vrijednost 0 [3]. Funkcija cilja minimizira ukupne troškove putovanja svih vozila, uzimajući u obzir sve relevantne ograničavajuće faktore kao što su vremenski okviri i kapaciteti vozila. Uz funkciju cilja, problem se definira **nizom ograničenja**:

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1 \quad \forall i \in C \tag{2.6}$$

Ograničenje 2.6 osigurava da **svaki klijent i bude posjećen točno jednom** od strane jednog vozila k. Zbroj svih varijabli x_{ijk} za sve moguće vozne rute j i sva vozila k mora biti jednak 1 za svakog klijenta i, čime se osigurava da je svaki klijent posjećen od strane



Slika 2.6. VRPTW

samo jednoga vozača.

$$\sum_{i \in V} q_i \cdot x_{ijk} \leq c_k \quad \forall k \in K \tag{2.7}$$

Ograničenje 2.7 osigurava da **kapacitet** c_k **vozila** k **nije prekoračen**. Varijabla q_i označava stvarnu količinu tereta (ili potražnju) kod klijenta i. Ograničenje nalaže da ukupna količina tereta koju vozilo k prevozi tijekom svoje rute ne smije premašiti njegov kapacitet c_k. Ako se radi o homogenoj floti vozila, onda su kapaciteti svih vozila jednaki c, no ako se radi o heterogenoj floti vozila onda je kapacitet za svako vozilo različit i označava se oznakom c_k. Pomoću ovog ograničenja, model osigurava da svako vozilo može

prevesti sve terete na svojoj ruti bez prekoračenja maksimalnog kapaciteta vozila.

$$\sum_{j \in V} x_{0jk} = 1 \quad \forall k \in K \tag{2.8}$$

Ograničenje 2.8 osigurava da svako **vozilo započne svoju rutu u skladištu**.

$$\sum_{j \in V} x_{j,n+1,k} = 1 \quad \forall k \in K \tag{2.9}$$

Ograničenje 2.9 osigurava da svako **vozilo završi svoju rutu u skladištu**.

$$s_{ik} + \tau_i + t_{ij} \leq s_{jk} + M(1 - x_{ijk}) \quad \forall i, j \in V, \forall k \in K \tag{2.10}$$

Ograničenje 2.10 osigurava da vozila poštuju vremenske okvire klijenata i da se **usluge obavljaju u skladu s definiranim redoslijedom obavljanja usluga**. Ovdje τ_i predstavlja trajanje usluge kod klijenta i, s_{ik} je stvarno vrijeme početka usluge kod klijenta i za vozilo k, dok je M velika pozitivna konstanta koja se koristi za omogućavanje da se ograničenje zadovolji kada varijabla x_{ijk} poprima vrijednost 0, što se postiže primjenom "velike M metode" u matematičkom programiranju.

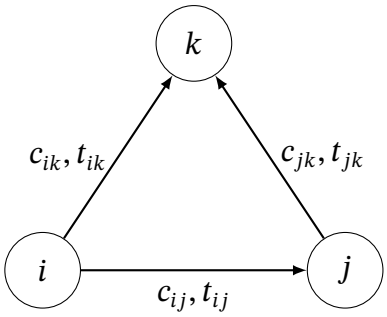
$$t_i^e \leq s_{ik} \leq t_i^l \quad \forall i \in V, \forall k \in K \tag{2.11}$$

Ograničenje 2.11 osigurava da **obavljanje usluga započne unutar zadanih vremenskih okvira**.

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in V, \forall k \in K \tag{2.12}$$

Ograničenje 2.12 definira da su varijable x_{ijk} **binarne**, tj. poprimaju vrijednosti 0 ili 1, ovisno o tome prolazi li vozilo k izravno između čvorova i i j.

Nadalje, **nejednakost trokuta** osigurava da trošak odnosno vrijeme putovanja između dviju lokacija **posredno**, tj. preko treće lokacije, ne može biti manji od sume troškova ili vremena putovanja između istih dviju lokacija **izravno** [3]. Drugim riječima, za slučaj sa slike ispod mora vrijediti: $c_{ij} + c_{jk} \geq c_{ik}$ i $t_{ij} + t_{jk} \geq t_{ik}$.



Ovako definiran model pruža dobru podlogu za ostvarenje, koje je izloženo u poglavlju 3.

2.4. Metode za rješavanje problema usmjeravanja vozila s vremenskim okvirima

Metode za rješavanje problema usmjeravanja vozila s vremenskim okvirima (VRPTW) dijele se, kao i kod klasičnog VRP-a, na egzaktne metode te aproksimacijske metode, koje uključuju heuristike i metaheuristike.

Egzaktne metode koriste matematičke modele i algoritme kako bi pronašle optimalno rješenje problema. Egzaktne metode, nažalost, pronalaze rješenje vrlo često presporo [3]. To postavlja pitanje: je li zaista nužno imati savršeno rješenje ako izračunavanje tog rješenja traje, primjerice, danima?

U stvarnim primjenama često je dovoljno dobiti rješenje koje je adekvatno ako se ono može dobiti u prihvatljivom roku. Zbog toga se egzaktne metode najčešće koriste u teorijskim kontekstima, gdje pomažu u boljem razumijevanju i definiranju problema. Heurističke i metaheurističke metode često nude praktičnija rješenja, koja balansiraju između kvalitete rješenja i vremena potrebnog za njegov izračun.

2.4.1. Egzaktne metode za rješavanje VRPTW-a

Egzaktne metode za rješavanje problema usmjeravanja vozila s vremenskim okvirima (VRPTW) mogu se podijeliti u tri skupine [3]:

- **Metode temeljene na Lagrangeovoj relaksaciji**: Lagrangeova relaksacija specifična je primjena koncepta Lagrangeove dualnosti. U osnovi, Lagrangeova dual-

nost koristi Lagrangeove multiplikatore kako bi preformulirala optimizacijski problem s ograničenjima tako da se ta ograničenja uključe u funkciju cilja. To omogućava rad s jednostavnijim verzijama problema i olakšava pronalaženje rješenja.

- **Metode temeljene na generiranju stupaca**: ove metode koriste Dantzig-Wolfeovu dekompoziciju, gdje se originalni problem dijeli na glavni problem i podproblem. Glavni problem jest skup particioniranja, dok je podproblem problem najkraćeg puta s ograničenjima resursa (ESPPRC). Cilj je pronaći stupce s negativnim smanjenim troškovima koji će poboljšati ukupno rješenje.

- **Metode temeljene na dinamičkom programiranju**: ove metode razbijaju problem na manje, međusobno povezane podprobleme koji se rješavaju zasebno.

Pored ovih pristupa, egzaktne metode mogu uključivati i druge tehnike, kao što su cjelobrojno linearno programiranje (MILP) [6], metoda grana i granica (Branch and Bound) [4], metoda grana i rezova (Branch and Cut) [7] te njihove kombinacije. Svaka od ovih metoda ima svoje prednosti i nedostatke, a izbor metode često ovisi o zahtjevima problema i dostupnim resursima.

2.4.2. Heurističke metode za rješavanje VRPTW-a

Heurističke metode za rješavanje VRPTW-a mogu se podijeliti u dvije **skupine**:

- **Konstruktivne heuristike (engl. Route-building heuristics)**: ove metode počinju s praznim rješenjem i postupno grade rute dodavanjem klijenata jedan po jedan. Primjer je Clarke-Wrightov algoritam uštede, gdje se klijenti kombiniraju na temelju maksimalnih ušteda [3].

- **Heuristike za poboljšanje rute (engl. Route-improving heuristics)**: ove metode polaze od valjanog rješenja i pokušavaju ga iterativno poboljšati. Primjeri uključuju 2-opt i 3-opt metode, koje optimiziraju postojeće rute zamjenom tzv. "rubova" kako bi se smanjila ukupna udaljenost [3].

Neke od danas najčešće korištenih heurističkih metoda za rješavanje VRPTW-a uključuju:

- **Clarke-Wrightov algoritam uštede** (engl. Clarke-Wright Savings Algorithm):

popularan je zbog svoje jednostavnosti i često se koristi kao početna metoda za brzo generiranje rješenja.

- **2-opt** i **3-opt** algoritmi: ove metode lokalne pretrage optimiziraju postojeće rute zamjenom tzv. "rubova" rute kako bi se smanjila ukupna udaljenost, s time da 2-opt radi s dva, a 3-opt s tri ruba.

- **Metoda najbližeg susjeda** (Nearest Neighbor Algorithm): ova metoda generira rješenja tako da vozilo uvijek posjećuje najbližeg klijenta koji još nije posjećen.

2.4.3. Metaheurističke metode za rješavanje VRPTW-a

Metaheurističke metode napredniji su pristup optimizaciji od heurističkih te one koriste određene mehanizme pretraživanja kako bi izbjegle lokalne minimume i istražile veći dio prostora rješenja.

Najčešće korištene metaheurističke metode za rješavanje VRP-a, a isto tako i VRPTW-a, jesu sljedeće:

- **Genetski algoritmi (Genetic Algorithms)**: koriste operacije poput selekcije, križanja i mutacije za evoluciju skupa rješenja prema boljim rješenjima.

- **Simulirano kaljenje (Simulated Annealing)**: oponaša proces hlađenja metala; rješenja se prihvaćaju ili odbacuju na temelju vjerojatnosti koja se smanjuje tijekom vremena omogućujući tako izlazak iz lokalnih minimuma.

- **Algoritam roja čestica (Particle Swarm Optimization)**: temelji se na kolektivnom ponašanju čestica (koje predstavljaju rješenja) koje se kreću kroz prostor rješenja pod utjecajem vlastitih iskustava i iskustava susjeda.

- **Tabu pretraga (Tabu Search)**: koristi memoriju (tabu listu) za praćenje već istraženih rješenja i zabranjuje njihovo ponavljanje, čime se poboljšava istraživanje prostora rješenja.

U ovom radu koristit će se genetski algoritmi za rješavanje problema usmjeravanja vozila s vremenskim okvirima.

2.4.4. Genetski algoritmi

Genetski algoritmi jesu metode pretraživanja koje se temelje na prirodnim procesima evolucije: u populaciji rješenja preživljavaju najsposobnije jedinke, koje prenose svoja obilježja na sljedeće generacije. Konačno rješenje predstavlja najbolja evoluirana jedinka u populaciji [8].

Za razliku od drugih metoda optimizacije, poput simuliranog kaljenja i tabu pretrage, genetski algoritmi unapređuju populaciju rješenja, a ne pojedinačnu jedinku, odnosno rješenje [3]. Svako rješenje predstavlja se kao jedinka unutar populacije, a cilj je evoluirati populaciju kroz više generacija koristeći operatore selekcije, križanja i mutacije. **Selekcija** je postupak odabira budućih roditelja. **Križanje** kombinira obilježja roditelja kako bi se stvorili potomci s po mogućnosti još boljim performansama, dok **mutacija** osigurava genetsku raznolikost unoseći slučajne promjene nad djetetom koje pomažu algoritmu da ne zaglavi u nekome od lokalnih minimuma.

Primjena genetskih algoritama, naime, nije ograničena na problem usmjeravanja vozila: oni se mogu koristiti u raznim drugim područjima kao što je bioinformatika ili planiranje proizvodnje. Upravo sposobnost da se prilagode različitim vrstama problema čini ih korisnom optimizacijskom metodom.

Uobičajeni pseudokod genetskog algoritma može se vidjeti ispod.

Genetski algoritam - pseudokod

1: **inicijalizacija**: generiraj početnu populaciju nasumičnih rješenja.
2: **for** svaka generacija **do**
3: **evaluacija**: izračunaj fitnes za svaku jedinku u populaciji.
4: **selekcija**: odaberi najbolje jedinke koje će služiti kao roditelji.
5: **križanje**: kreiraj nove potomke kombiniranjem karakteristika roditelja.
6: **mutacija**: primijeni mutaciju na potomke s određenom vjerojatnošću.
7: **zamjena**: zamijeni staru populaciju novom generacijom potomaka.
8: **end for**
9: **povrat rezultata**: pronađi i vrati najbolju jedinku iz konačne populacije.

3. Implementacija

Ovo poglavlje opisuje **ostvarenje** problema usmjeravanja vozila s vremenskim ograničenjima (**VRPTW**) u kontekstu **usluga na zahtjev**. Usluge na zahtjev, poput usluga autotaksi prijevoza ili dostave hrane, zahtijevaju optimizaciju raspodjele resursa kako bi se efikasno ispunili dinamični zahtjevi korisnika u stvarnom vremenu.

Implementacija je podijeljena u faze: svaka se bavi specifičnim potproblemom. Tehnologije korištene u implementaciji uključuju programski jezik **Python** za algoritme, **PostgreSQL** za pohranu podataka, te knjižnicu **Geodesic** za izračun udaljenosti. Nadalje, **Jupyter Notebook** korišten je za modularni razvoj i dokumentiranje.

Primijenjeni su **genetski algoritmi** zbog svoje sposobnosti da efikasno pretražuju veliki broj mogućih rješenja, što je ključno za dinamičke scenarije. Implementacija je strukturirana kroz rješavanje nekoliko ključnih potproblema, koji su postupno povećavali složenost sustava.

Potproblemi od najjednostavnijeg do najtežeg:

1. **VRPTW s ograničenjima radnog vremena**: fokus je na raspodjeli zadataka radnicima unutar njihovog radnog vremena. Optimiziramo raspored za naredni tjedan, od ponedjeljka do petka, s unaprijed definiranim zadacima, bez očekivanja novih poslova, što predstavlja statičko raspoređivanje.

2. **Uvođenje različitih tipova zadataka**: dodajemo fleksibilno radno vrijeme za svakog radnika i uključujemo različite tipove vremenskih zahtjeva za poslove (*Scheduled*, *Anytime*, *Time Windows*).

3. **Dinamičko dodjeljivanje zadataka u stvarnom vremenu**: ostvarenje algoritma koji može dinamički dodjeljivati zadatke radnicima u stvarnom vremenu, uzima-

jući u obzir trenutne lokacije radnika i stanja na terenu. Ovdje se uvodi i tip posla *Now*.

3.1. Implementacija statičkog VRPTW-a s ograničenjima radnog vremena

Cilj je problema usmjeravanja vozila s vremenskim okvirima (VRPTW) optimizirati rute flote vozila kako bi se minimalizirala ukupna prijeđena udaljenost, uzimajući u obzir postavljena ograničenja. U ovom radu, kao prvi potproblem, razmatra se problem **optimizacije rasporeda radnika** u poduzeću koje pruža usluge čišćenja. Konkretno, fokus je na optimizaciji rasporeda **10 čistača** unutar jednoga poduzeća, s ciljem povećanja profita i smanjenja ukupne udaljenosti koju radnici trebaju prijeći prilikom obavljanja svojih zadataka.

Čistačice su raspoređene u **dvije smjene**: jutarnju (08:00 - 16:00) i popodnevnu (16:00 - 20:00), s po 5 radnika u svakoj smjeni. Optimizacija rasporeda provodi se **statički** za radni tjedan. Svaki posao ima unaprijed određeno **trajanje** i iznos **zarade**. Bez smanjenja općenitosti, pretpostavlja se da radnici prevoze potrebnu opremu i alate te da su oni već uračunati u maksimalni kapacitet vozila.

Svaki radnik započinje i završava svoju smjenu u sjedištu poduzeća, što osigurava da radnici svakodnevno započinju i završavaju na istoj lokaciji, što je važno za precizno planiranje ruta. Radnici mogu raditi najviše određeni broj sati dnevno (8 sati), što se mora uzeti u obzir prilikom raspodjele poslova kako bi se izbjegla prekomjerna opterećenost.

Prije optimizacije definirane su konstante, parametri i hiperparametri koji opisuju problem. To uključuje maksimalno radno vrijeme po danu (8 sati), broj radnih dana (5), te financijske parametre poput troška goriva po kilometru (0.1 €/km), satnice radnika (7 €/h). Hiperparametri genetskog algoritma uključuju veličinu populacije (100), broj generacija (50), stopu križanja (80%) i stopu mutacije (5%). Ovi parametri prikazani su na slici 3.1. koja ilustrira ovaj dio koda.

Iako je model primijenjen na poduzeće koja pruža usluge čišćenja, on bi se mogao prenamijeniti i za druge vrste usluga. Na primjer, može biti koristan za optimizaciju

```
# Parametri za problem VRPTW
max_work_hours_per_day = 8  # Maksimalno radno vrijeme po danu (u satima)
num_days = 5  # Broj radnih dana za koje određujemo raspored poslova

# Financijski parametri
fuel_cost_per_km = 0.1  # Trošak goriva po kilometru - 0.1 €/km
wage_per_hour = 7  # Satnica radnika - 7 €/h
overtime_wage_per_hour = 15.0  # Satnica za prekovremeni rad - 15 €/h

# Hiperparametri za genetski algoritam
population_size = 100  # Veličina populacije
generations = 50  # Broj generacija
crossover_rate = 0.8  # Stopa križanja
mutation_rate = 0.05  # Stopa mutacije
```

Slika 3.1. Parametri korišteni za problem

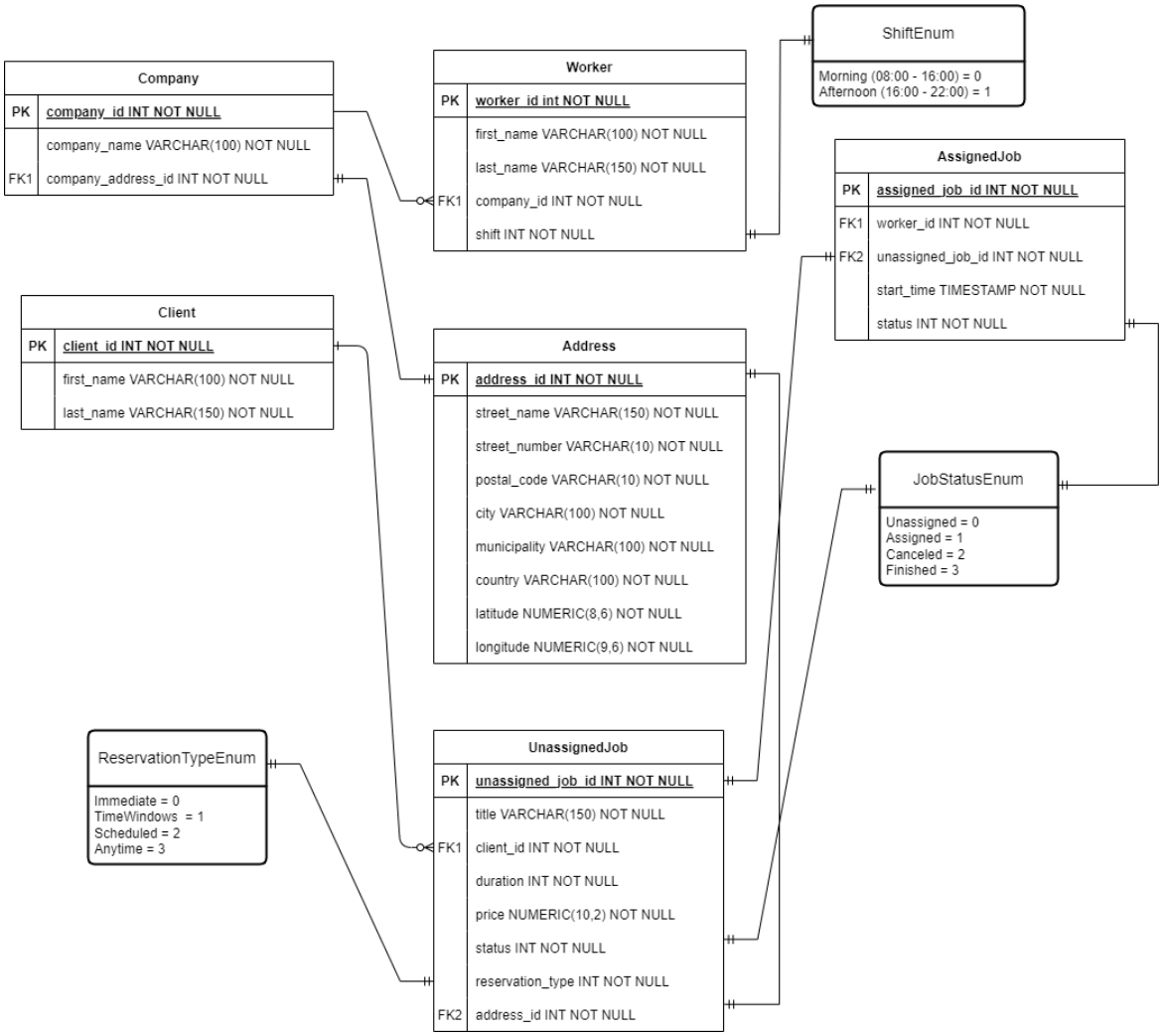
rasporeda radnika u poduzeću koja se bavi sklapanjem namještaja, ili u agenciji koja nudi usluge servisiranja kućanskih uređaja na mjestu. Općenito, rješenje je primjenjivo u svakom kontekstu gdje postoji skup radnika koji pružaju usluge klijentima tamo gdje se ti klijenti nalaze.

3.1.1. Korišteni ulazni podatci

Za rješavanje problema usmjeravanja vozila s vremenskim okvirima (VRPTW) izrađena je **baza podataka** nazvana **"worker_scheduling_vrptw"**, koja pohranjuje ključne informacije potrebne za optimizaciju raspodjele zadataka. Na slici 3.2. prikazan je ER dijagram te baze podataka.

Baza podataka sadrži nekoliko povezanih tablica koje omogućuju pohranu ključnih informacija o poduzećima, zaposlenicima, klijentima, adresama, te poslovima (zadatcima) koji se trebaju izvršiti.

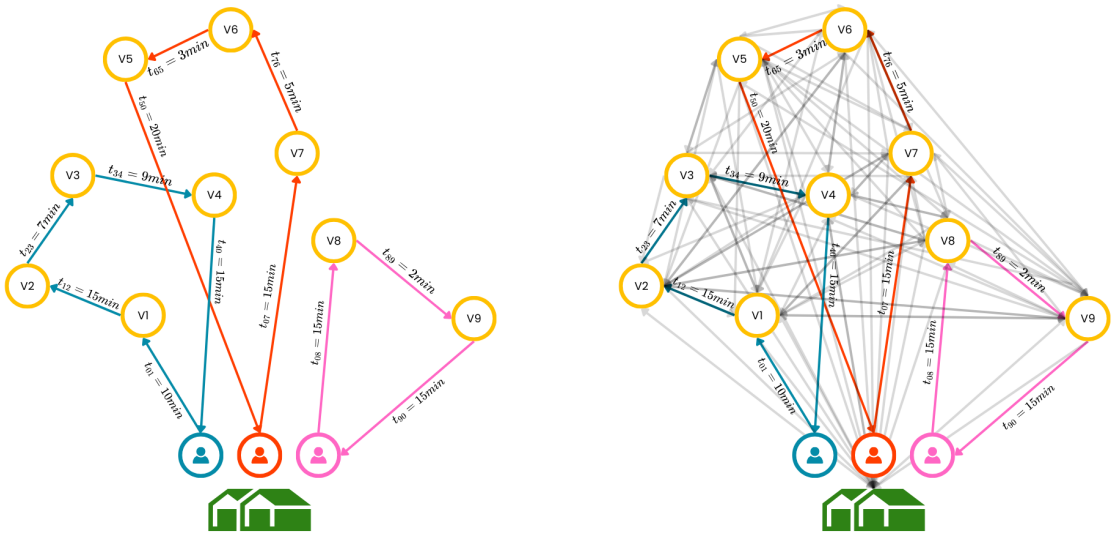
Tablice u bazi podataka uključuju *company*, *worker*, *client*, *address*, *unassigned_job*, *assigned_job*, te su nacrtane tri pomoćne tablice za pohranu različitih enumeracija (*ShiftEnum*, *JobStatusEnum*, *ReservationTypeEnum*). Ove pomoćne tablice nisu pohranjene u bazi podataka, već su prikazane kako bi bolje ilustrirale strukturu i funkcionalnost sustava.



Slika 3.2. ER dijagram baze podataka "vrptw$_s hi f ts_d b\varepsilon$

3.1.2. Formulacija problema

Problem formuliramo grafički kao **potpuni usmjereni graf** s vrhovima $V = \{0, 1, \dots, N\}$ i lukovima $E = \{(i, j) \mid i, j \in V, i \neq j\}$. Potpuni usmjereni graf ili *turnir* je **digraf** u kojem svaki par različitih vrhova ima dva usmjerena brida, odnosno luka. Svaki od $n+1$ vrhova u takvom grafu povezan je s svakim drugim vrhom s dvama lukovima, rezultirajući s $2 * n * (n + 1)$ usmjerenih bridova. Dakle, sve su moguće **veze** definirane (sive strelice sa slike 3.3.). Kao što vidimo na slici 3.3., svaki luk ima pridjeljenu vrijednost t_{ij} koja predstavlja trajanje puta automobilom od lokacije i do lokacije j.



Slika 3.3. Primjer - VRP kao potpuni usmjereni graf

Veze su definirane kroz **matricu udaljenosti** između svih točaka, koja **nije nužno simetrična**. Matrica udaljenosti (ili vremena putovanja) sadrži vrijednosti koje predstavljaju udaljenost ili vrijeme **između parova lokacija** u mreži i izračunava se koristeći **Geodesic**. Na slikama 3.4. i 3.5. vidimo primjere takvih matrica za graf s 6 čvorova, odnosno 6 lokacija. Izračunavanje matrice udaljenosti između svih točaka ima složenost $O(n^2)$, gdje je n broj vrhova. Matrica udaljenosti se računa prije pokretanja genetskog algoritma kako bi se lagano dohvatile sve udaljenosti kroz samu optimizaciju.

Kako bi se formalno opisao problem i omogućilo njegovo rješavanje pomoću algoritamskih metoda, potrebno je matematički definirati elemente ovog modela. To uključuje funkciju cilja te ograničenja koja osiguravaju da su svi zadaci ispravno raspoređeni

$$D_{\text{km}} = \begin{bmatrix} 0 & 5 & 10 & 15 & 20 & 25 \\ 6 & 0 & 7 & 12 & 18 & 22 \\ 11 & 8 & 0 & 5 & 13 & 17 \\ 16 & 13 & 6 & 0 & 8 & 11 \\ 21 & 19 & 14 & 9 & 0 & 6 \\ 26 & 23 & 18 & 12 & 7 & 0 \end{bmatrix}$$

Slika 3.4. Primjer matrice udaljenosti u kilometrima

$$D_{\text{min}} = \begin{bmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 12 & 0 & 15 & 25 & 35 & 45 \\ 22 & 17 & 0 & 10 & 25 & 35 \\ 32 & 27 & 12 & 0 & 18 & 28 \\ 42 & 37 & 27 & 20 & 0 & 12 \\ 52 & 47 & 37 & 30 & 14 & 0 \end{bmatrix}$$

Slika 3.5. Primjer matrice udaljenosti u minutama

i obavljeni unutar zadanih uvjeta.

Funkcija cilja

Cilj je **maksimizirati profit** poduzeća, koji se definira kao razlika između prihoda i ukupnih troškova, uključujući troškove putovanja i troškove isplaćivanja plaće radnicima za radni dan od 8 h. Funkcija cilja jest sljedeća:

$$\max \quad \text{Profit} = \text{Prihod} - \text{Trošak} \tag{3.1}$$

U matematičkom obliku, funkcija cilja se može izraziti kao:

$$\max \quad \left(\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} r_{ij} \cdot x_{ij}^k - \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij}^k \right) \tag{3.2}$$

gdje:

- r_{ij} predstavlja prihod ostvaren obavljanjem posla između lokacija i i j.
- c_{ij} predstavlja ukupne troškove putovanja između lokacija i i j.
- x_{ij}^k je binarna varijabla koja poprima vrijednost 1 ako vozilo k obavlja posao između lokacija i i j, inače poprima vrijednost 0.

Ograničenja

Uz navedenu funkciju cilja, potrebno je precizno navesti sva ograničenja koja modeliraju problem usmjeravanja vozila u skladu s ciljem minimizacije troškova. Uz sva ograničenja opisana u poglavlju 2.3. kao što su: polazak iz skladišta, povratak u skladište, kapacitet, kontinuitet rute i slično, specifično u ovom problemu potrebno je pripaziti na to da se

posao obavi unutar granica radnog vremena radnika.

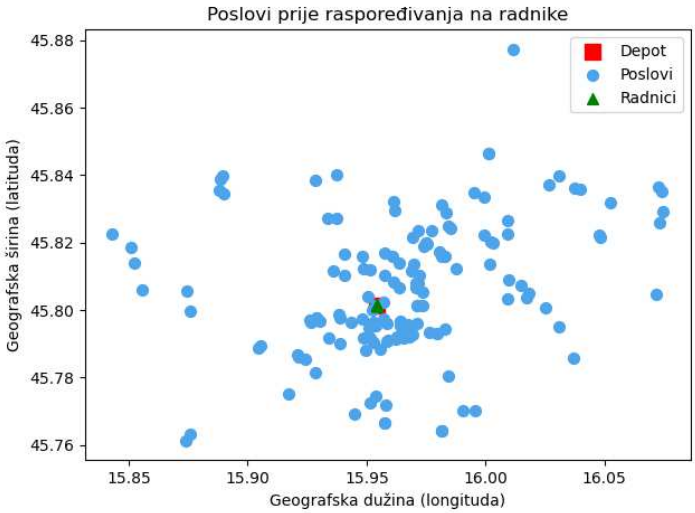
1. **Ograničenje radnog vremena**: osigurava da radnik ne prekorači maksimalno radno vrijeme. Ako je τ_j trajanje obavljanja posla j, a t_{ij} je vrijeme potrebno za putovanje od posla i do posla j, odnosno njihov je zbroj ukupno vrijeme koje radnik k provede na lokaciji j i u putovanju do nje, tada:

$$\sum_{j \in V \setminus \{0\}} (\tau_j + t_{ij}) \cdot x_{ijk} \leq T_k \quad \forall k \in K$$

gdje je T_k maksimalno radno vrijeme radnika k.

3.1.3. Rješavanje problema usmjeravanja vozila s vremenskim okvirima genetskim algoritmom

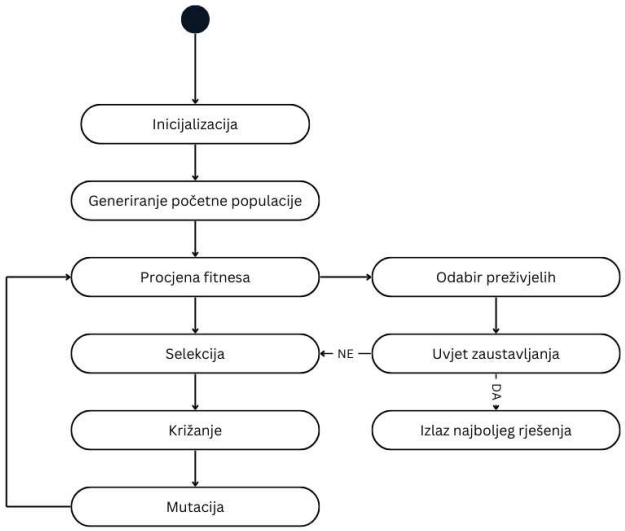
Genetski algoritam odabran je zbog svoje učinkovitosti u pretraživanju velikog prostora rješenja i prilagodljivosti složenim ograničenjima. Ovdje se prikazuje njegova praktična primjena. Prvi korak uključuje dohvat svih poslova s pripadajućim lokacijama. Za izračun udaljenosti između tih lokacija računa se matrica udaljenosti. Matrica je udaljenosti izračunata koristeći knjižnicu **Geodesic**. Na slici su prikazani poslovi prije nego što su dodijeljeni radnicima (vidi sliku 3.6.).



Slika 3.6. Poslovi prije raspoređivanja na radnike

GA započinje **inicijalizacijom** i generiranjem početne populacije rješenja. Zatim se procjenjuje **dobrota** svakog rješenja, što omogućava odabir najboljih rješenja za daljnje

iteracije. U svakoj iteraciji provodi se **selekcija**, **križanje** i **mutacija** rješenja kako bi se generirale nove, potencijalno bolje jedinke. Proces se ponavlja sve dok se ne ispuni **uvjet zaustavljanja**. Na slici 3.7. prikazan je **dijagram toka genetskog algoritma**.

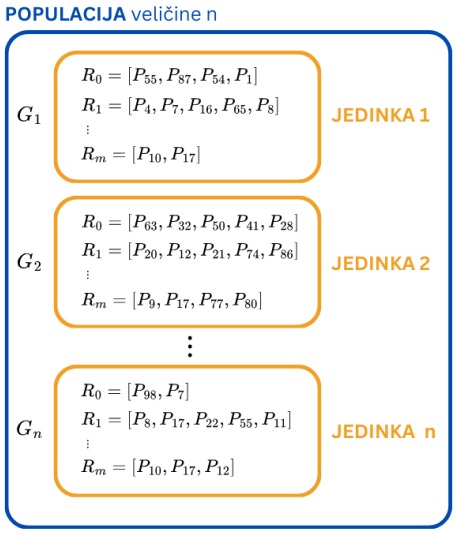


Slika 3.7. Dijagram toka genetskog algoritma

Inicijalizacija početne populacije

Prvi korak genetskog algoritma (GA) jest **inicijalizacija**. U ovoj fazi populacija se nasumično inicijalizira kako bi se osigurala raznolikost rješenja. Svaka jedinka predstavlja moguće rješenje problema raspoređivanja poslova na radnike te sadrži skup ruta (dodijeljenih poslova) i listu poslova koji nisu mogli biti dodijeljeni zbog ograničenja. Tijekom inicijalizacije, poslovi se nasumično dodjeljuju radnicima uz poštivanje svih zadanih ograničenja.

Na slici 3.8. prikazan je primjer strukture populacije i jedinki u okviru genetskog algoritma. Populacija se sastoji od n jedinki (označenih kao $G_1, G_2, \dots, G_n$), gdje svaka jedinka predstavlja moguće rješenje problema. Svaka jedinka sadrži više ruta $R_0, R_1, \dots, R_m$, a svaka ruta obuhvaća niz poslova P_i dodijeljenih pojedinom radniku. U ovom pojednostavljenom modelu, svaka ruta odgovara smjeni jednog radnika. Ukupno je 10 radnika koji rade u dvije smjene: jutarnjoj (08:00 - 16:00) i popodnevnoj (16:00 - 22:00), od ponedjeljka do petka. Zadatak je statički rasporediti poslove na 10 radnika tijekom 5 radnih



Slika 3.8. Primjer strukture populacije u genetskom algoritmu

dana. Iz toga vidimo da problem raspoređivanja na više dana nije ništa drugo nego raspoređivanje poslova u potencijalnih **50 ruta** (5 smjena po radniku kroz 5 dana), što znači da je m u ovom slučaju 50. Svaka ruta traje maksimalno 8 sati, što odgovara radnom vremenu radnika. U ovom problemu se vremenska ograničenja smatraju **tvrdim ograničenjima**, što znači da u niti jednom slučaju neće se dogoditi kršenje tih ograničenja. Ako se posao ne može rasporediti tako da poštuje ograničenja, neće se onda uopće ni rasporediti, čak ni u inicijalnoj populaciji.

Procjena dobrote

Procjena dobrote (engl. fitness evaluation) mjeri kvalitetu rješenja na temelju **razlike** između prihoda i troškova za svaku jedinku (skup ruta). **Prihod** se generira obavljanjem poslova na svakoj ruti, dok **troškovi** uključuju troškove goriva za putovanje između poslova i troškove isplate plaće koju poduzeće treba isplatiti radnicima. Na slici 3.9. prikazan je kod funkcije za procjenu dobrote u programskom jeziku Python. Cilj je da je dobrota konačnog rješenja što veća, jer to znači da je što veći profit. Kroz višestruka pokretanja algoritma u ovom problemu, konačni profit obično je iznosio oko 3000 € čiste zarade za poduzeće za jedan radni tjedan, nakon pokrića svih troškova.

```python
def evaluate_fitness(genome: List[int], jobs: List[UnassignedJob], distance_matrix: np.ndarray, time_matrix: np.ndarray,
                     max_work_minutes_per_day: int, fuel_cost_per_km: float, wage_per_hour: float, overtime_wage_per_hour: float) -> float:
    total_revenue = 0.0   # Prihod
    total_cost = 0.0      # Trošak
    overtime_cost = 0.0   # Trošak prekovremenih sati

    routes = genome["routes"]

    for route in routes:
        if not route:
            continue

        # Obrada prihoda i troškova za rutu
        route_revenue, route_cost, route_duration = process_route(route, jobs, distance_matrix, time_matrix, fuel_cost_per_km, wage_per_hour)

        # Dodavanje prekovremenih troškova ako postoje
        overtime_cost += calculate_overtime(route_duration, max_work_minutes_per_day, overtime_wage_per_hour)

        total_revenue += float(route_revenue)  # Pretvorba u float ako je Decimal
        total_cost += float(route_cost)        # Pretvorba u float ako je Decimal

    total_cost += float(overtime_cost)         # Pretvorba u float ako je Decimal
    return float(total_revenue) - float(total_cost)  # Pretvorba u float
```

Slika 3.9. Funkcija za procjenu dobrote

Selekcija

Selekcija u genetskom algoritmu odnosi se na proces odabira najboljih jedinki iz trenutne populacije kako bi sudjelovale u stvaranju sljedeće generacije. Postoji nekoliko različitih metoda selekcije u genetskim algoritmima:

- **Turnirska selekcija**: nasumično se odabire nekoliko jedinki iz populacije, a najbolja postaje roditelj. Ova metoda daje prednost boljim jedinkama i osigurava raznolikost.
- **Selekcija prema rangu**: jedinke se rangiraju prema dobroti, a vjerojatnost odabira ovisi o njihovom rangu. Pomaže očuvanju raznolikosti populacije.
- **Ruletska selekcija**: jedinke imaju vjerojatnost odabira proporcionalnu svojoj dobroti vrijednosti. Bolje jedinke imaju veće šanse, ali i slabije jedinke mogu nekad biti odabrane što doprinosi istraživanju šireg prostora rješenja.
- **Elitizam**: najbolja jedinka iz trenutne generacije uvijek prelazi u sljedeću generaciju.

U ovom problemu kombinirane su **turnirska selekcija** i **elitizam**. Turnirska selekcija omogućuje nasumičan odabir nekoliko jedinki iz populacije, pri čemu se najbolja jedinka iz tog skupa odabire kao roditelj na temelju dobrote. Ovaj pristup osigurava ravnotežu između kvalitete i raznolikosti rješenja.

S druge strane, **elitizam** osigurava da se najbolja jedinka iz trenutne generacije automatski prenosi u sljedeću generaciju, čime se zadržava kvaliteta najboljih rješenja. Elitizam je ostvaren u funkciji **create_new_population**, gdje se pomoću funkcije **find_elites** najbolji članovi populacije prenose u novu generaciju prije provođenja selekcije, križanja i mutacije. Funkcija za selekciju roditelja pomoću turnirske selekcije prikazana je na slici 3.10.

```python
def select_parents(population, fitness, tournament_size=3):
    """
    Selekcija dvoje roditelja iz populacije na temelju turnirske selekcije.
    Biraju se nasumično 'tournament_size' jedinki, te se bira najbolja po fitnessu.
    """
    tournament = random.sample(list(zip(population, fitness)), tournament_size)
    # Odabir najboljih roditelja na temelju fitnessa
    parent1 = max(tournament, key=lambda x: x[1])[0]

    tournament = random.sample(list(zip(population, fitness)), tournament_size)
    parent2 = max(tournament, key=lambda x: x[1])[0]

    return parent1, parent2
```

Slika 3.10. Funkcija za selekciju roditelja koristeći turnirsku selekciju

Križanje

Križanje u ovom slučaju kombinira rješenja dvaju roditelja kako bi se stvorili novi potomci. Svaka jedinka predstavlja skup ruta radnika, a križanje se koristi za razmjenu dijelova tih ruta između roditelja. Postupak križanja je sljedeći:

1. Nasumično se odaberu dva roditelja.
2. Bira se nasumična točka križanja unutar rješenja.
3. Prvi dio ruta prvog roditelja kombinira se s drugim dijelom ruta drugog roditelja i obratno, stvarajući dva nova potomka.

Ako roditelji nemaju dovoljno ruta za križanje, potomci su jednostavne kopije roditelja. U ovom postupku križanja, ne mijenjaju se poslovi unutar jedne rute, već se gotove rute poslova raspoređuju na različite radnike tj. rute. Na slici 3.11. prikazan je kod za križanje implementiran u ovom problemu.

```python
def crossover(parent1, parent2):
    """
    Izvršava križanje između dva roditelja kako bi se generirali potomci.

    Parameters:
    - parent1: Prvi roditelj (lista).
    - parent2: Drugi roditelj (lista).

    Returns:
    - child1, child2: Dva nova potomka (liste) dobivena križanjem roditelja.
    """
    # Provjera duljine roditelja kako bi se osiguralo da križanje ima smisla
    if len(parent1) > 2 and len(parent2) > 2:
        # Odabir točke križanja između 1 i len(parent1)-2
        crossover_point = random.randint(1, len(parent1) - 2)

        # Generiranje potomaka kombiniranjem segmenata roditelja
        child1 = parent1[:crossover_point] + parent2[crossover_point:]
        child2 = parent2[:crossover_point] + parent1[crossover_point:]
    else:
        # Ako su roditelji prekratki, potomci su samo kopije roditelja
        child1 = parent1.copy()
        child2 = parent2.copy()

    return child1, child2
```

Slika 3.11. Funkcija za križanje

Mutacija

Mutacija u ovom problemu odnosi se na nasumičnu izmjenu ruta unutar rješenja kako bi se očuvala ili povećala raznolikost populacije. Cilj mutacije je spriječiti preuranjenu konvergenciju algoritma prema lokalnom optimumu te omogućiti istraživanje novih potencijalno boljih rješenja. Mutacija se odvija sljedećim postupkom:

1. nasumično se odabire ruta iz jedinke;
2. zatim se nasumično zamjenjuju rute unutar jedinke;
3. ovaj proces uvodi nove kombinacije ruta, što povećava raznolikost populacije.

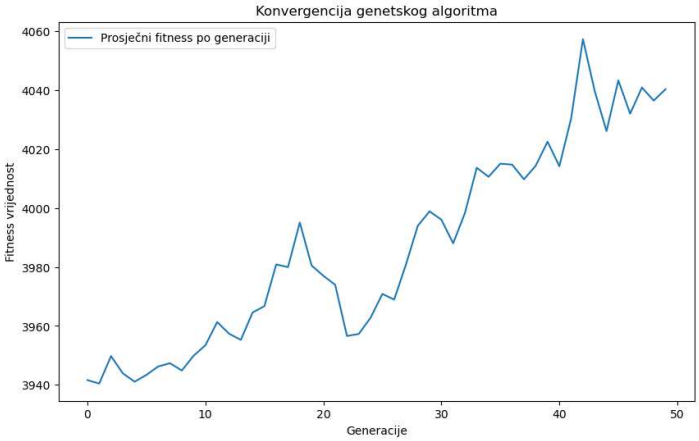
Mutacija se provodi s malom vjerojatnošću (mutation_rate = 0.01) kako bi se izbjeglo nasumično ponašanje algoritma, ali osigurava istraživanje šireg prostora rješenja. Na slici 3.12. prikazan je kod funkcije za mutaciju ostvarene u ovom problemu.

```python
def mutate(genom, mutation_rate=0.01):
    """
    Izvršava mutaciju na danom pojedincu promjenom slučajnih gena.
    Parametri:
    - genom: Pojedinac (lista ili rječnik) koji će biti mutiran.
    - mutation_rate: Vjerojatnost mutacije svakog gena.
    Povratna vrijednost:
    - Mutirani pojedinac.
    """
    if isinstance(genom, dict) and 'routes' in genom:   # Provjera je li genom rječnik s ključem 'routes'
        routes = genom['routes']  # Dohvat rute iz genoma

        # Provjera ima li dovoljno ruta za zamjenu
        if len(routes) < 2:
            return genom  # Premalo ruta za mutaciju, nema smisla mijenjati

        # Prolazak kroz svaku rutu i primjena mutacije
        for i in range(len(routes)):
            if random.random() < mutation_rate: # Nasumična provjera hoće li se mutacija dogoditi
                swap_with = random.randint(0, len(routes) - 1) # Nasumična ruta za zamjenu
                routes[i], routes[swap_with] = routes[swap_with], routes[i] # Zamjena ruta

        # Ažuriranje 'routes' u genomu nakon mutacije
        genom['routes'] = routes

    else:
        raise TypeError("Pogrešna struktura genoma. Očekivao se rječnik s ključem 'routes'.")

    return genom
```

Slika 3.12. Funkcija za mutaciju

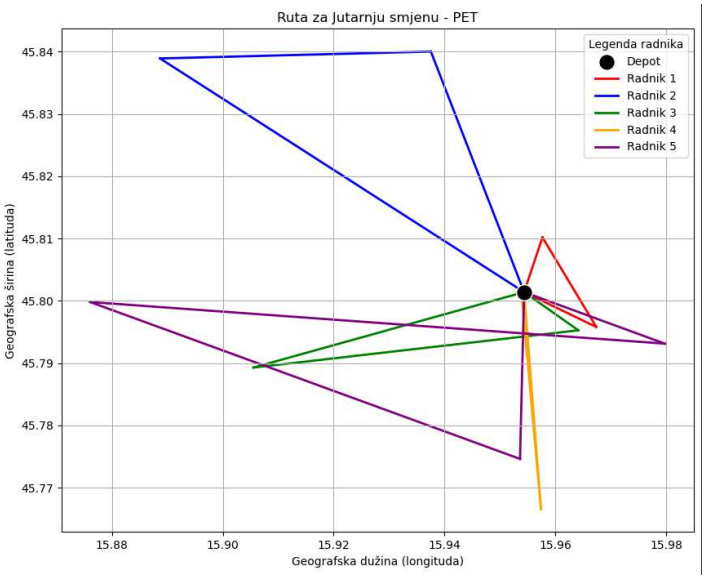
3.1.4. Rezultati

Tijekom optimizacije genetskog algoritma, broj generacija postavljen je na 50. Male prilagodbe hiperparametara, poput veličine populacije te stopa križanja i mutacije, omogućile su pronalazak optimalne točke konvergencije. Nakon višestrukih pokretanja, broj 50 pokazao se kao optimalan, jer se dobrota stabilizirala oko pedesete generacije. Na slici 3.13. prikazana je konvergencija algoritma dobivena pokretanjem programa.



Slika 3.13. Konvergencija genetskog algoritma

Na slici 3.14. prikazan je primjer optimalno raspoređenih ruta radnika tijekom jutarnje smjene u petak. Svaka boja predstavlja rutu pojedinog radnika, dok crna točka označava sjedište poduzeća (depot). Vidljivo je kako su poslovi raspoređeni tako da radnici pokrivaju različite geografske lokacije, čime se postiže učinkovitost u obavljanju zadataka i smanjuje preklapanje ruta. Ovakav način raspodjele omogućuje bolju organizaciju radnog vremena i smanjenje troškova. Konkretan ispis ruta radnika za petak ujutro, gdje



Slika 3.14. Primjer rute radnika jutarnje smjene za jedan dan

svaka ruta detaljno prikazuje redoslijed poslova za svakog radnika vidljiva je na slici **??** Ova optimizacija osigurava da su zadaci ravnomjerno raspoređeni, uz minimalno preklapanje i maksimalnu učinkovitost prilikom obavljanja dnevnih zadataka.

```
Ruta za radnika 1: PET (08:00 - 16:00)
  Posao 105: Čišćenje i dezinfekcija skladišta (08:00)
  Posao 116: Čišćenje hotelskog predvorja (13:00)
  Zarada: 100.00 €, Trošak: 1.81 €, Profit: 49.19 €
  Ukupno trajanje rute: 7.00 sati
```

```
Ruta za radnika 2: PET (08:00 - 16:00)
  Posao 142: Dubinsko čišćenje galerije (08:00)
  Posao 25: Dvodnevno čišćenje stana (12:00)
  Zarada: 97.00 €, Trošak: 0.44 €, Profit: 51.06 €
  Ukupno trajanje rute: 6.50 sati
```

```
Ruta za radnika 3: PET (08:00 - 16:00)
  Posao 108: Dubinsko čišćenje teretane (08:00)
  Posao 72: Čišćenje vanjskog dijela kuće (12:00)
  Zarada: 95.00 €, Trošak: 0.84 €, Profit: 45.16 €
  Ukupno trajanje rute: 7.00 sati
```

```
Ruta za radnika 4: PET (08:00 - 16:00)
  Posao 40: Čišćenje luksuzne kuće (08:00)
  Posao 54: Čišćenje i dezinfekcija kupaonice (14:00)
  Zarada: 122.00 €, Trošak: 1.07 €, Profit: 68.43 €
  Ukupno trajanje rute: 7.50 sati
```

```
Ruta za radnika 5: PON (08:00 - 16:00)
  Posao 148: Dubinsko stana (08:00)
  Posao 84: Čišćenje i dezinfekcija hladnjaka (14:00)
  Zarada: 118.00 €, Trošak: 1.68 €, Profit: 63.82 €
  Ukupno trajanje rute: 7.50 sati
```

Slika 3.15. Ispis ruta za jutarnju smjenu u petak

3.2. Optimizacija rute: različiti tipovi vremenskih zahtjeva

Drugi problem "**Optimizacija rute: različiti tipovi vremenskih zahtjeva**" nadograđuje prethodni problem uvođenjem različitih tipova rezervacija za poslove. Dok se u prvom problemu pretpostavljalo da se svaki posao može obaviti u bilo kojem trenutku, ovdje su poslovi kategorizirani prema četiri tipa vremenskih zahtjeva:

1. **Fleksibilni poslovi (bilo kada)**: Ovaj tip posla nema striktno određeno vrijeme obavljanja, što znači da se može izvršiti bilo kada unutar radnog dana. Radnici sami odlučuju kada će obaviti posao na temelju optimizacije svoje rute.

2. **Zakazani poslovi**: Ovi poslovi imaju točno određeno vrijeme izvršenja, primjerice, posao koji mora biti obavljen u 16:00 u petak, 23.8.2024. Radnici moraju biti na lokaciji točno u zadanom vremenu, što zahtijeva precizno planiranje unutar rute kako bi se osigurala pravovremena izvedba.

3. **Poslovi s vremenskim intervalima**: Ovi poslovi mogu se obaviti unutar određenog vremenskog intervala, primjerice između 12:00 i 20:00 28.7.2024., ili između 08:00 i 13:00 29.7.2024. Radnici imaju fleksibilnost u odabiru točnog trenutka obavljanja posla, ali moraju se pridržavati danog ili danih vremenskih intervala.

4. **Hitni poslovi (odmah)**: Ovi poslovi zahtijevaju hitno izvršenje, odnosno moraju biti obavljeni što je prije moguće. U ovom slučaju, cilj je pronaći radnika koji će najbrže doći na lokaciju i odmah izvršiti zadatak, što zahtijeva vrlo brzu reakciju i prilagodbu rute.

U ovom problemu **hitni poslovi** nisu uzeti u obzir jer je optimizacija i dalje **statička**, fokusirana na unaprijed poznate zadatke. Hitni poslovi, koji zahtijevaju dinamičko raspoređivanje u stvarnom vremenu, bit će obrađeni u sljedećem poglavlju, koje se bavi dinamičkom dodjelom poslova kako oni pristižu. U tom kontekstu, algoritam će morati prilagoditi trenutne rute radnika kako bi obuhvatio hitne zadatke.

Uvođenje ovih različitih tipova rezervacija povećava kompleksnost optimizacije, jer svaki tip posla zahtijeva drugačiji pristup pri planiranju ruta. Fleksibilni poslovi dopu-

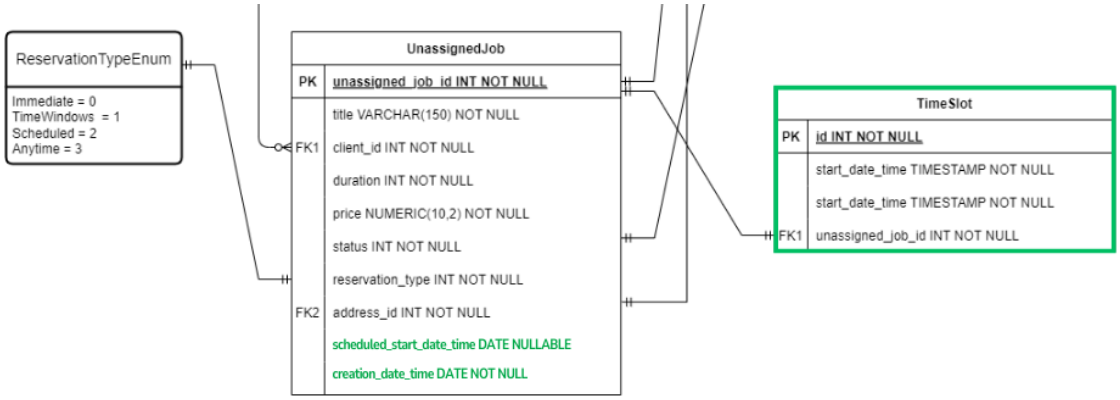
štaju veću slobodu u organizaciji, dok hitni i planirani poslovi zahtijevaju preciznije planiranje kako bi se zadovoljili specifični vremenski zahtjevi.

3.2.1. Ulazni podatci

U prethodnom problemu već je bila stvorena baza podataka s poljem *reservation_type* za svaki nepridijeljeni posao unutar tablice *UnassignedJob*. U tom slučaju, svi poslovi imali su vrijednost *anytime* (3), što je značilo da se posao može obaviti bilo kada unutar radnog dana. U novoj verziji problema, poslovi mogu pripadati jednoj od četiri kategorije: *immediate* (0), *timewindow* (1), *scheduled* (2) i *anytime* (3). Stoga potrebna izmjena baze podataka ukjučuje dodavanje nekoliko novih polja i tablice **TimeSlot**:

- Polje *scheduled_start_date_time* u tablici *UnassignedJob*, koje se koristi za poslove s tipom rezervacije *scheduled* (vrijednost 2). Ovo polje pohranjuje točno vrijeme kada posao treba biti izvršen.
- Polje *creation_date_time*, također u tablici *UnassignedJob*, koje omogućava praćenje koliko je vremena prošlo od kada je zahtjev za posao tipa *immediate* (0) poslan, čime se optimizira hitnost obrade takvih poslova.
- Nova tablica *TimeSlot*, koja je povezana s tablicom *UnassignedJob*, služi za pohranu vremenskih intervala (*time window* poslovi).

Ove izmjene omogućuju podršku za nove tipove poslova i njihovu efikasnu obradu unutar genetskog algoritma. Struktura ažurirane baze podataka prikazana je na slici 3.16.



Slika 3.16. Ažurirana baza podataka za ovaj problem

3.2.2. Formulacija problema

U ovome proširenom problemu, ključna novost je uvođenje različitih vremenskih ograničenja za obavljanje poslova. Poslovi više nisu obavljeni bilo kada, već ovisno o njihovom tipu rezervacije. Ovi tipovi rezervacija uključuju poslove s točno zakazanim vremenom, poslove s fleksibilnim vremenskim okvirima, te standardne poslove koji se mogu obaviti bilo kada unutar radnog dana.

Glavna izmjena u formulaciji problema odnosi se na poštivanje **vremenskih okvira** i **prioriteta** poslova. Zakazani poslovi moraju se izvršiti u točno definiranom trenutku, dok poslovi s vremenskim okvirima omogućuju fleksibilnost unutar danog intervala. Poslovi koji se mogu obaviti bilo kada daju veću slobodu pri planiranju ruta, ali je važno da optimizacija uzme u obzir ukupnu učinkovitost rasporeda radnika. Hitni poslovi ostaju za sljedeće poglavlje: dinamičko raspoređivanje.

Funkcija cilja ostaje nepromijenjena, usmjerena na **maksimizaciju profita**, koji se definira kao razlika između ukupnih prihoda i troškova. Troškovi uključuju troškove putovanja, radnog vremena i prekovremenih sati. Dodatno, uvedena su vremenska ograničenja za zakazane poslove i poslove s vremenskim okvirima. Ako posao nije završen unutar zadanog vremenskog okvira, to negativno utječe na profit u obliku kazni ili smanjenja prihoda.

$$\max \quad \text{profit} = \left(\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} r_{ij} \cdot x_{ij}^k \right) - \left(\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij}^k + \sum_{j \in V} \lambda_j \cdot \text{kašnjenje}_j \right) \tag{3.3}$$

gdje:

- r_{ij} predstavlja prihod ostvaren obavljanjem posla između lokacija i i j.
- c_{ij} predstavlja troškove putovanja između lokacija i i j.
- x_{ij}^k je binarna varijabla koja je 1 ako radnik k obavlja posao između lokacija i i j, a 0 inače.
- λ_j predstavlja kaznu za kašnjenje u izvršavanju posla na lokaciji j.

- kašnjenje$j$ predstavlja vremensko kašnjenje za posao na lokaciji j, definirano kao razlika između planiranog i stvarnog vremena završetka posla.

3.2.3. Rješavanje problema genetskim algoritmom

Rješavanje ovog problema predstavlja poseban izazov zbog prisutnosti različitih tipova zadataka: zadataka s vremenskim prozorima, zakazanih zadataka te fleksibilnih zadataka koji se mogu izvršiti u bilo kojem trenutku unutar radnog dana. Ova raznolikost otežava optimalnu raspodjelu jer svaki tip zadatka nosi svoja vremenska ograničenja i specifičnosti, što zahtijeva pažljivu strategiju prilikom dodjele poslova u rute. U ovom problemu ispitano je raspoređivanje korištenjem prioriteta, dok je u trćem problemu isprobano nasumično raspoređivanje s kažnjivanjima koja tjeraju poslove u odgovarajuće vremenske okvire.

Jedan od prvih pristupa koji pada na pamet prilikom rješavanja problema s različitim tipovima zadataka jest taj da se prvo rasporede zadaci s vremenskim ograničenjima, kao što su zakazani zadaci (*scheduled*) i zadaci s vremenskim prozorima (*time windows*). Pri tome se tipovima zadataka mogu dodijeliti **prioriteti**, gdje bi *scheduled* zadaci imali najviši prioritet (**P1**), zadaci s vremenskim prozorima srednji (**P2**), a fleksibilni zadaci koji se mogu obaviti bilo kada (*anytime*) najniži prioritet (**P3**).

Međutim, ovaj pristup ima značajan **nedostatak** – davanjem prednosti zadacima s fiksnim vremenskim ograničenjima, fleksibilni zadaci često bivaju zanemareni, što može rezultirati suboptimalnim rješenjima. Takva stroga strategija može također ograničiti pretragu prostora rješenja i dovesti algoritam do zaglavljivanja u lokalnim optimumima.

Zbog toga je važno uvesti određenu razinu **nasumičnosti**. Nasumičnost omogućava algoritmu da istraži širi prostor mogućih rješenja, dok istovremeno ne gubi iz vida ključne vremenske zahtjeve pojedinih zadataka. Na taj način, genetski algoritam može kombinirati prioritizaciju i nasumičnost kako bi stvorio balans između raspoređivanja zadataka prema važnosti i održavanja raznolikosti u populaciji rješenja.

Inicijalizacija populacije

Funkcija za inicijalizaciju populacije koristi kombinaciju prioriteta i nasumičnosti kako bi osigurala da su zadaci raspoređeni na optimalan način. Budući da se poslovi razlikuju po vrsti vremenskih ograničenja, važno je prilagoditi raspodjelu svakom tipu zadatka, dok se zadržava raznolikost u početnoj populaciji. Tipovi zadataka se dijele prema tri prioriteta:

- **P1: Zakazani zadaci** – zadaci koji imaju fiksno vrijeme početka i koje je potrebno dodijeliti na točno određeno vrijeme.
- **P2: Zadaci s vremenskim rasponima** – zadaci koji imaju definirane vremenske raspone unutar kojih moraju biti završeni.
- **P3: Fleksibilni zadaci** – zadaci koji nemaju vremenskih ograničenja i mogu se obaviti u bilo kojem trenutku unutar radnog dana.

U funkciji inicijalizacije *initialize_population* prvo se poslovi podijele s obzirom na vrstu zadataka, s ciljem određivanja prioriteta (slika 3.17.).

```
# Podijeli zadatke po tipovima zadataka (različiti prioriteti)
scheduled_jobs = [job for job in jobs if job.reservation_type == ReservationTypeEnum.SCHEDULED]     # P1
timewindow_jobs = [job for job in jobs if job.reservation_type == ReservationTypeEnum.TIMEWINDOWS]  # P2
anytime_jobs = [job for job in jobs if job.reservation_type == ReservationTypeEnum.ANYTIME]         # P3
```

Slika 3.17. Podjela poslova prema prioritetima

Inicijalizacija populacije koristi kombinaciju prioritetnog odabira i nasumičnog biranja poslova. Ovdje se koristi parametar *priority_weight* (npr. 0.7) koji određuje koliko često se biraju zadaci prema prioritetu u odnosu na nasumično biranje. Ako je nasumični broj manji od *priority_weight*, tada se zadatak bira prema prioritetu (najprije *scheduled*, zatim *time windows*). Ako je nasumični broj veći od *priority_weight*, tada se zadatak bira nasumično iz bilo koje kategorije. (vidi sliku 3.18.).

Provjerava se mogu li se zadaci s vremenskim ograničenjima (*Scheduled* i *Time Windows*) uspješno smjestiti u rutu na temelju trenutnog rasporeda. Za *Scheduled* zadatke (P1) provjerava se jesu li dodijeljeni točno u predviđeno vrijeme. Za *Time Windows* zadatke (P2) provjerava se nalaze li se unutar zadanog vremenskog prozora (slika 3.19.).

```python
if random.random() < priority_weight:
    if scheduled_jobs:
        selected_job = scheduled_jobs.pop(0)
    elif timewindow_jobs:
        selected_job = timewindow_jobs.pop(0)
    else:
        selected_job = anytime_jobs.pop(0)
else:
    job_pool = scheduled_jobs + timewindow_jobs + anytime_jobs
    selected_job = random.choice(job_pool)
```

Slika 3.18. Nasumičnost uz prioritete

```python
if selected_job.reservation_type == ReservationTypeEnum.SCHEDULED:
    scheduled_start_time = selected_job.scheduled_start_date_time
    if not (current_route_time_as_datetime == scheduled_start_time):
        can_assign = False

elif selected_job.reservation_type == ReservationTypeEnum.TIMEWINDOWS:
    can_assign = False
    for time_slot in selected_job.time_slots:
        time_window_start = time_slot.start_date_time
        time_window_end = time_slot.end_date_time
        if time_window_start <= current_route_time_as_datetime <= time_window_end and finish_time <= time_window_end:
            can_assign = True
            break
```

Slika 3.19. Odabir zadataka s vremenskim ograničenjima

Fleksibilni zadaci koji nemaju vremenskih ograničenja mogu se smjestiti bilo gdje u rasporedu, ako ostali uvjeti (npr. dostupnost vremena) dopuštaju. Kada zadatak zadovoljava sve uvjete, dodjeljuje se ruti, a trenutni radni sati i putovanja za tu rutu se ažuriraju. Ako zadatak ne može biti dodijeljen nijednoj ruti, stavlja se na popis nepridruženih zadataka.

Procjena dobrote

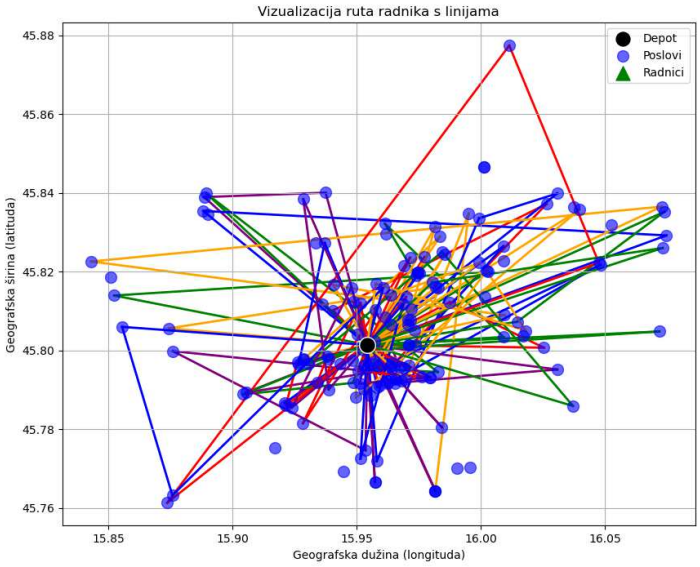
U funkciji procjene dobrote provjeravaju se vremenska ograničenja za različite tipove zadataka. Zakazani zadaci moraju se obaviti točno u predviđeno vrijeme, dok zadaci s vremenskim prozorima trebaju biti završeni unutar zadanog raspona. Nepoštivanje tih ograničenja rezultira visokom penalizacijom, čime se obeshrabruju loša rješenja.

```python
# Glavna funkcija za evaluaciju fitnessa
def evaluate_fitness(genome: List[int], jobs: List[UnassignedJob], distance_matrix: np.ndarray, time_matrix: np.ndarray, current_time,
                     fuel_cost_per_km: float, penalty_weight_per_minute_in_eur: float = 1.0) -> float:
    """
    Izračunava fitness funkciju - cilj je maksimizacija profita
    """

    total_revenue = 0.0  # Ukupna zarada
    total_cost = 0.0     # Ukupni troškovi putovanja - gorivo
    total_penalty = 0.0  # Penalizacije za kašnjenje i neobavljene poslove

    routes = genome["routes"]

    # Prolazimo kroz sve rute
    for route in routes:
        if not route:
            continue

        jobs_in_route = [next((job for job in jobs if job.unassigned_job_id == gene.job_id), None) for gene in route]   # Pronađi poslove u ruti
        jobs_in_route = [job for job in jobs_in_route if job is not None]                                  # Ukloni None vrijednosti iz liste
        worker_index = route[0].worker_id    # Indeks radnika u ruti

        # 1. Izračunaj PROFIT za sve poslove u ruti
        total_revenue += calculate_profit(jobs_in_route)

        # 2. Izračunaj TROŠKOVE putovanja za sve poslove u ruti
        distance_cost = calculate_distance_cost(jobs_in_route, worker_index, fuel_cost_per_km)
        total_cost += distance_cost

        # 3. Izračunaj PENALIZACIJU za kašnjenje i uranjene dolaske za sve poslove u ruti
        total_penalty += calculate_timing_penalty(route, jobs_in_route, current_time, penalty_weight_per_minute_in_eur)

    # 4. Izračunaj PENALIZACIJU za neobavljene poslove
    unassigned_jobs = [job for job in jobs if job.unassigned_job_id not in [gene.job_id for route in routes for gene in route]]
    total_penalty += calculate_unassigned_job_penalty(unassigned_jobs, penalty_weight_per_minute_in_eur)

    # Ukupna dobrota: prihod - trošak - penalizacije
    return total_revenue - total_cost - total_penalty
```

Slika 3.20. Procjena dobrote u slučaju različitih tipova zadataka

3.2.4. Rezultati

U ovom potproblemu izmijenjene su funkcije izračuna dobrote, prioriteti kod inicijalizacije te struktura podataka, dok su mutacija, križanje i ostali dijelovi genetskog algoritma ostali isti. Ovaj problem radi sporije od prijašnjeg, ali i dalje uspjeva dobiti optimalno rješenje. Rješenje ima malo manji profit zbog različitih ograničenja koja smanjuju fleksibilnost raspoređivanja poslova.



Slika 3.21. Pridruženi poslovi u rute

3.3. Dinamičko dodjeljivanje zadataka u stvarnom vremenu

Dinamičko dodjeljivanje zadataka omogućuje sustavu prilagodbu promjenjivim uvjetima na terenu, poput novih zahtjeva klijenata ili promjena u rasporedu radnika. Zadaci se ne dodjeljuju unaprijed, već se radnicima dodjeljuju "u hodu", uzimajući u obzir njihovu trenutnu lokaciju, dostupnost i optimalnu rutu. Pravovremena reakcija na ove promjene ključno je kako bi se smanjila kašnjenja i optimizirali troškovi [9].

U ovoj implementaciji optimizacija se provodi za jedan dan, 1. srpnja 2024., kada je 10 radnika aktivno unutar sustava. Zadaci dolaze dinamički i sustav ih raspoređuje radnicima na temelju trenutnih uvjeta. Simulacija se odvija u intervalima od pola sata, pri čemu se svakih 30 minuta generiraju novi zadaci. Genetski algoritam se svakih pola sata ponovno pokreće, uzimajući u obzir trenutne lokacije i dostupnost radnika te zakazane zadatke iz prethodnih koraka. Cilj je optimalno dodijeliti zadatke radnicima, poštujući sva postavljena ograničenja.

Zadaci se dijele na nekoliko tipova: fleksibilni zadatci (anytime), zakazani zadaci (scheduled), zadatci s vremenskim prozorima (time windows) i hitni zadaci (immediate). Ovisno o tipu zadatka, genetski algoritam prilagođava dodjelu kako bi optimizirao rješavanje zadataka, što se postiže kroz prilagodbu **funkcije dobrote**.

Ovakav pristup omogućuje efikasno upravljanje radnim resursima, prilagodbu u realnom vremenu i poboljšano korisničko iskustvo. Klijenti mogu biti sigurni da će njihovi zadaci biti izvršeni na vrijeme, uz minimalna kašnjenja.

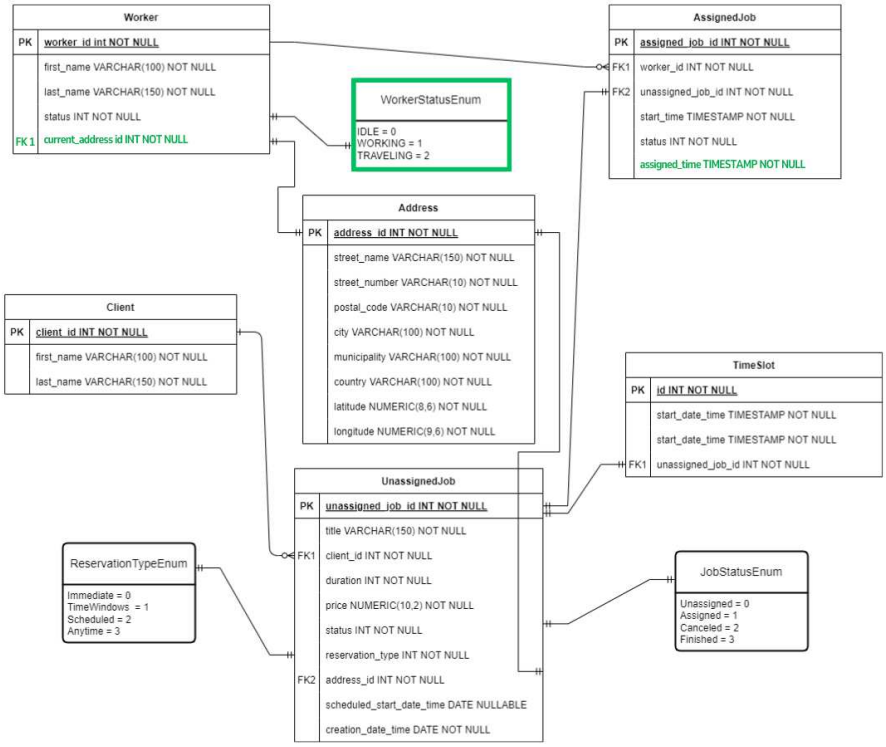
3.3.1. Ulazni podatci

U novom modelu baze (slika 3.22.) uveden je **WorkerStatusEnum**, koji omogućuje praćenje statusa radnika u stvarnom vremenu, poput toga je li radnik slobodan (`IDLE`), radi na zadatku (`WORKING`) ili putuje prema lokaciji zadatka (`TRAVELING`). Ovaj status pomaže pri dinamičkom dodjeljivanju zadataka i prilagođavanju uvjetima na terenu.

Za razliku od prijašnjeg modela (slika 3.16.), gdje je postojao koncept smjena s pomoću `ShiftEnum`, smjenski rad je sada uklonjen. Svaki radnik djeluje kao "svoj vlastiti

zaposlenik", što omogućuje veću fleksibilnost. Zbog tog pristupa, tablica `Company` je također uklonjena jer nema potrebe za vezom radnika s kompanijom.

Dodatno, radnici sada imaju polje `current_address_id`, koje prati njihovu trenutnu lokaciju. To olakšava sustavu da odabere najboljeg radnika za određeni zadatak na temelju udaljenosti. U prijašnjem modelu ta informacija nije bila uključena, što je ograničavalo mogućnost optimizacije rasporeda u realnom vremenu.



Slika 3.22. ER dijagram baze podataka *"vrptw_dynamic_db"*

3.3.2. Formulacija problema

U ovom problemu uvodimo dinamičko dodjeljivanje zadataka u stvarnom vremenu. Sustav mora dodijeliti nove zadatke radnicima, uzimajući u obzir njihovu **trenutnu lokaciju**, **dostupnost** i **optimalnu rutu**. Cilj je maksimizirati **profit**, a istovremeno minimizirati **kašnjenja**. Funkcija cilja je sljedeća:

$$\max \quad \text{profit} = \text{prihodi} - \text{troškovi} - \text{kazne} \tag{3.4}$$

gdje su:

- **Prihodi:**

$$\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} r_{ij} \cdot x_{ij}^{k}$$

Prihod od izvršenih poslova između lokacija i i j.

- **Troškovi:**

$$\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij}^{k}$$

Trošak putovanja između lokacija i i j.

- **Kazne:**

$$\sum_{j \in V} \lambda_j \cdot \text{kašnjenje}_j$$

Kazne zbog kašnjenja u izvršavanju poslova na lokaciji j u odnosu na predviđeno vrijeme obavljanja posla.

3.3.3. Rješavanje problema genetskim algoritmom

Problem dinamičkog raspoređivanja zadataka rješava se pomoću genetskog algoritma, pri čemu svako rješenje (jedinka) predstavlja **rute** radnika i **nepridružene poslove**. Algoritam prilagođava zadatke radnicima na temelju njihove **trenutne lokacije**, **dostupnosti** i **tipova zadataka**. Na kraju svake iteracije, cilj je optimalno dodijeliti preostale nepridružene poslove i minimizirati kašnjenja te kazne.

Budući da se zadaci dodjeljuju dinamički, potrebno je simulirati dinamičko okruženje. To je ostvareno tako da simulacija započinje u **07:00** i napreduje u koracima od **30 minuta**. U svakoj iteraciji generiraju se novi poslovi, uz uzimanje u obzir prethodno nepridijeljenih zadataka, nakon čega se ponovno pokreće genetski algoritam za njihovo raspoređivanje. Tako se dobivaju nove optimalne rute (najbolji genom) prije prelaska na sljedeći korak simulacije, koja traje do **23:00**. Opisani postupak vidljiv je na slici 3.23.

Inicijalizacija populacije

U prethodnom pristupu, zadaci su dodjeljivani radnicima samo ako su mogli poštovati sva ograničenja, poput vremenskih prozora ili maksimalnog radnog vremena. Zadatci koji nisu ispunjavali te uvjete ostajali su nedodijeljeni, što je često dovodilo do subop-

```python
def simulate_day():
    # Definiranje parametara simulacije (početak i kraj simulacije)
    start_time = datetime(year=2024, month=7, day=1, hour=7)
    end_time = datetime(year=2024, month=7, day=1, hour=23, minute=0)
    current_time = start_time  # Inicijalno vrijeme postavljeno na početak simulacije

    # Petlja koja prolazi kroz dan u intervalima od 30 minuta
    if True:
        while current_time <= end_time:
            # Simulacija kreiranja novog zadatka (nasumično 4-5 zadataka svakih 30 minuta)
            num_new_jobs = random.randint(4, 5)
            for _ in range(num_new_jobs):
                create_random_unassigned_job(current_time)  # Kreiraj nasumični zadatak

            # Dohvati sve nedodijeljene zadatke iz baze podataka
            unassigned_jobs = fetch_all_unassigned_jobs()

            # Dohvati sve dostupne radnike (status: IDLE) iz baze podataka
            all_workers = fetch_all_workers()
            available_workers = [worker for worker in all_workers if worker.status == WorkerStatusEnum.IDLE]

            # Prikaz trenutnog vremena i broja nedodijeljenih zadataka na grafičkom prikazu
            visualize_jobs_and_workers_on_xy_graph(unassigned_jobs, all_workers, current_time.strftime("%H:%M"))

            # Pokušaj dodijeliti zadatke dostupnim radnicima
            if unassigned_jobs and available_workers:  # Provjera postoje li zadaci i radnici za dodjelu
                assign_jobs_to_workers(unassigned_jobs, available_workers, current_time)

            # Pomakni vrijeme za 30 minuta naprijed
            current_time += timedelta(minutes=30)

        print("Simulacija za dan je završena.")  # Ispis kada simulacija završi
```

Slika 3.23. Simulacija dana i dinamičkog raspoređivanja poslova

timalnih rješenja jer mnogi zadaci nisu bili obavljeni. U ovom novom pristupu, **nasumično** dodjeljujemo zadatke radnicima, bez obzira na to jesu li svi uvjeti ispunjeni. Nakon toga, genetski algoritam evaluira rješenja i dodjeljuje **kazne** za ona koja ne zadovoljavaju sva ograničenja, poput kašnjenja ili preklapanja zadataka. Ovaj način omogućuje fleksibilnije rješavanje problema, jer algoritam može pokušati pronaći bolju konfiguraciju kroz evoluciju rješenja (bez uključivanja apriornog znanja).

Uz to, svi tipovi zadataka (fleksibilni, hitni, zakazani i zadaci s vremenskim prozorima) tretiraju se jednako, s istim prioritetima. To omogućuje sustavu da uravnoteži dodjeljivanje različitih vrsta poslova i postepeno poboljša rješenja kroz iteracije, uzimajući u obzir stvarne uvjete na terenu.

Procjena dobrote

Funkcija dobrote u ovom modelu izračunava se kao i u prijašnjem problemu, samo je još u izračun penalizacije uključen i hitan tip poslova. Dakle, ukupna zarada je suma prihoda obavljenih poslova, troškovi putovanja su suma troška goriva po prijeđenim kilometrima, dok penalizacije trebaju se pobliže definirati s kaznama:

- **Penalizacija za kašnjenje ili uranjeni dolazak** (*penalty for delay*) računa se za svaki zadatak na temelju razlike između stvarnog i planiranog vremena početka zadatka. Kazna iznosi `penalty` × 1 € po minuti kašnjenja. Primjerice, ako je zadatak trebao početi u 16:00, a počinje u 12:00, razlika je 240 minuta, te je kazna 240 € × `penalty`.

- **Penalizacija za neobavljene zadatke** (*penalty for unassigned jobs*) računa se za svaki zadatak koji **nije dodijeljen** radniku. Kazna je jednaka negativnoj vrijednosti zarade tog zadatka × `penalty_unassigned`. Na primjer, ako zadatak ima zaradu od 100 €, kazna za neobavljeni zadatak iznosi -100 € × `penalty_unassigned`.

S obzirom na to da želimo izbjeći obavljanje poslova u različito vrijeme od onog koje su klijenti zatražili, možemo prilagoditi kaznu preko `penalty`, npr. `penalty=3`. Isto tako i za neobavljene poslove - trošak kod previše neobavljenih poslova nije samo izgubljena potencijalna zarada, već i mogući gubitak povjerenja klijenata. Ukupna funkcija dobrote izračunava se kao:

$$\text{fitness} = \text{total revenue} - \text{total cost} - \text{total penalty}$$

Kazne za kašnjenje i neobavljene zadatke zbrajaju se kao *total penalty*, gdje se svaka pojedinačna penalizacija računa prema prethodno opisanim pravilima. Cilj je pronaći rješenje sa što većom funkcijom dobrote, jer to znači da je što veća zarada. Programski kod funkcije `evaluate_fitness` opisan je u prijašnjem poglavlju (slika 3.20.), pa je ovdje opisan samo izračun penala (slika 3.24.).

```python
def calculate_timing_penalty(route: List, jobs_in_route: List, current_time: int, penalty_weight: float) -> float:
    total_timing_penalty = 0.0  # Inicijalizacija ukupne kazne

    for job_in_route in route:
        job = next((x for x in jobs_in_route if x.unassigned_job_id == job_in_route.job_id), None)

        time_difference = 0  # Inicijalna razlika u vremenu

        if job.reservation_type == ReservationTypeEnum.IMMEDIATE:
            # Ako je posao hitan, kazna se temelji na vremenu između nastanka i planiranog početka
            time_difference = abs(job.creation_date_time - job_in_route.scheduled_start_time)

        elif job.reservation_type == ReservationTypeEnum.SCHEDULED:
            # Ako je posao zakazan, kazna se temelji na vremenskom odstupanju od zakazanog početka
            time_difference = abs(job.scheduled_start_date_time - job_in_route.scheduled_start_time)

        elif job.reservation_type == ReservationTypeEnum.TIMEWINDOWS:
            # Ako posao ima vremenske prozore, izračunaj kaznu na temelju odstupanja od vremenskog prozora
            penalties = [
                abs(max(0, job_in_route.scheduled_start_time - time_window.start_date_time, time_window.end_date_time - job_in_route.scheduled_end_time))
                for time_window in job.time_slots  # Prođi kroz sve vremenske prozore
            ]
            # Odaberi minimalnu kaznu od svih prozora
            time_difference = min(penalties) if penalties else 0

        # Pretvori vremensku razliku u minute i dodaj u ukupnu kaznu
        total_timing_penalty += (time_difference.total_seconds() / 60) * penalty_weight if time_difference else 0

    return total_timing_penalty  # Vraća ukupnu kaznu za kašnjenja
```

Slika 3.24. Izračun kazne za kašnjenje ili uranjeni dolazak

Križanje

Križanje se odvija tako da se dvije jedinke (rješenja) međusobno kombiniraju kako bi se dobili potomci. Svaka jedinka predstavlja 10 ruta za 10 radnika. Sve jedinke imaju iste poslove, ali su ti poslovi raspoređeni na različite načine u različitim rutama. Križanje se vrši odabirom nasumične točke križanja, čime se nakon obavljanja križanja stvaraju dvije nove jedinke.

Na primjer, zamislimo dvije jedinke J_1 i J_2, pri čemu svaka ima zadatke raspoređene u dvije rute. Neka J_1 ima poslove P_1 i P_2 u prvoj ruti, a P_3 i P_4 u drugoj ruti. S druge strane, J_2 ima poslove P_3 i P_4 u prvoj ruti, a P_1 i P_2 u drugoj ruti:

$$J_1 = \{\text{R1: } P_1, P_2;\ \text{R2: } P_3, P_4\}$$

$$J_2 = \{\text{R1: } P_3, P_4;\ \text{R2: } P_1, P_2\}$$

Ako se križanje odvija između prve i druge rute, potomci bi naslijedili kombinirane rute od roditelja. Primjerice, ako je točka križanja između rute 1 i rute 2, potomci bi izgledali ovako:

$$\text{Potomak}_1 = \{\text{R1: } P_1, P_2\,(\text{od } J_2),\ \text{R2: } P_3, P_4\,(\text{od } J_1)\}$$

$$\text{Potomak}_2 = \{\text{R1: } P_3, P_4\,(\text{od } J_1),\ \text{R2: } P_1, P_2\,(\text{od } J_2)\}$$

Međutim, zbog križanja može doći do dupliciranja poslova unutar ruta. U ovom primjeru, poslovi P_1, P_2, P_3 i P_4 pojavljuju se više puta u različitim rutama potomaka. Kako bi se izbjegli duplikati, svi poslovi koji se dupliciraju bit će premješteni u skup *unassigned_jobs* (nepridruženi poslovi) pozivom funkcije `remove_duplicates`. Na ovaj način osigurava se da svaki posao pripada samo jednoj ruti ili bude premješten u skup nepridruženih poslova, čime se izbjegava da zadaci budu dodijeljeni više puta. Programski kod ove funkcije vidljiv je na slici 3.25.

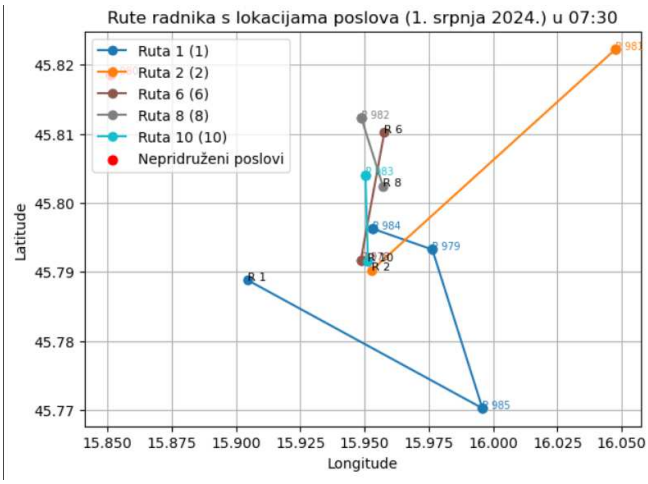
Međutim, ovakav pristup može rezultirati time da velik broj poslova završi u *unassigned_jobs*. Zato je na kraju funkcije dodan pokušaj da se nepridruženi poslovi ponovno dodijele natrag u rute, kako bi se smanjio broj nedodijeljenih zadataka. Dodatno, uveden je i postupak `crossover_within_single_individual`, koji omogućuje križanje dviju ruta unutar iste jedinke, što doprinosi dodatnoj raznolikosti rješenja.

```
def crossover(parent1, parent2):
    # Rute iz roditelja
    routes1 = parent1['routes']
    routes2 = parent2['routes']

    # Nepridruženi poslovi iz roditelja
    unassigned_jobs1 = parent1['unassigned_jobs']
    unassigned_jobs2 = parent2['unassigned_jobs']

    # Prikupljamo sve poslove (kompletne Gene objekte) iz oba roditelja (rute + nepridruženi poslovi)
    all_jobs_parent1 = {gene.job_id: gene for route in routes1 for gene in route}
    all_jobs_parent1.update({job.job_id: job for job in unassigned_jobs1})

    all_jobs_parent2 = {gene.job_id: gene for route in routes2 for gene in route}
    all_jobs_parent2.update({job.job_id: job for job in unassigned_jobs2})

    # Kreiramo potomke kao kopije roditelja
    child1_routes = routes1.copy()
    child2_routes = routes2.copy()

    # Nasumično odabir točke križanja između 1 i najmanje duljine roditelja - 1
    crossover_point = random.randint(1, min(len(routes1), len(routes2)) - 1)

    # Jednostavna zamjena ruta nakon točke križanja
    child1_routes[crossover_point:], child2_routes[crossover_point:] = routes2[crossover_point:], routes1[crossover_point:]

    # Uklanjamo duplikate u potomcima
    remove_duplicates(child1_routes)
    remove_duplicates(child2_routes)

    # Križanje unutar jedinki (za svaki potomak posebno)
    child1_routes = crossover_within_single_individual(child1_routes)
    child2_routes = crossover_within_single_individual(child2_routes)

    # Ažuriramo unassigned_jobs tako da dodamo poslove koji nisu dodjeljeni rutama
    child1_unassigned = update_unassigned_jobs(all_jobs_parent1, child1_routes)
    child2_unassigned = update_unassigned_jobs(all_jobs_parent2, child2_routes)

    # Vraćamo potomke s njihovim rutama i nepridruženim poslovima
    return {'routes': child1_routes, 'unassigned_jobs': child1_unassigned}, {'routes': child2_routes, 'unassigned_jobs': child2_unassigned}
```

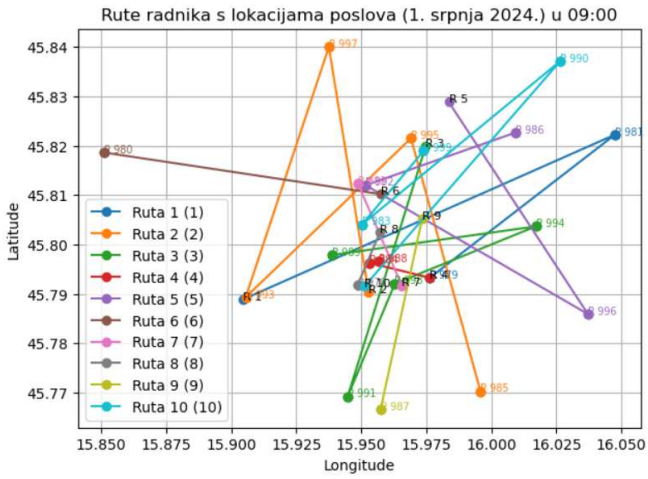
Slika 3.25. Križanje

3.3.4. Rezultati

Rješenje ovog problema implementirano je kroz simulaciju koja svakih pola sata pokreće genetski algoritam kako bi ažurirala rute radnika na temelju novih zadataka. Na početku dana radnicima su dodijeljeni početni zadaci, kao što je prikazano na slici 3.26. Tijekom dana, kako novi poslovi dolaze u sustav, rute se prilagođavaju, te se u svakom intervalu od pola sata ponovno optimiziraju. Primjer takve promjene vidljiv je u 9:00 sati na slici 3.27., gdje su rute prilagođene kako bi se uzeli u obzir novi zadaci i trenutna dostupnost radnika. Ovaj pristup omogućuje sustavu da dinamički reagira na promjene u stvarnom vremenu, kontinuirano optimizirajući rasporede radnika i minimalizirajući kašnjenja.



Slika 3.26. Raspored radnika u 7:30



Slika 3.27. Raspored radnika u 9:00

4. Praktična primjena u aplikacijama na zahtjev

Algoritmi razvijeni u ovom radu pružaju izuzetnu priliku za primjenu u industrijama koje zahtijevaju brzo i učinkovito upravljanje resursima, kao što su dostava hrane, transport, te kućne i profesionalne usluge. U takvim sustavima, gdje korisnici naručuju usluge putem aplikacija, a radnici ili dostavljači odgovaraju na te zahtjeve, **genetski algoritmi** za optimizaciju ruta, poput onih opisanih u ovom radu, omogućuju brzu i učinkovitu raspodjelu resursa. Ovi algoritmi uzimaju u obzir različite parametre poput **vremenskih intervala**, **zakazanih zadataka** i **fleksibilnosti** kako bi sustavi na zahtjev radili što bolje i s manjim troškovima.

Ključne prednosti primjene ovih algoritama uključuju: **bržu i precizniju dodjelu zadataka** te **dinamičku prilagodbu promjenama** u stvarnom vremenu. To omogućuje sustavima da neprestano optimiziraju rute radnika prema promjenama u potražnji, prometu ili dostupnosti resursa, osiguravajući maksimalnu učinkovitost i zadovoljstvo korisnika.

Kao praktičan primjer, algoritmi razvijeni u ovom radu mogli bi se primijeniti u aplikaciji **CleanApp**, koja je zamišljena kao platforma za usluge čišćenja na zahtjev. Primjenom genetskih algoritama, CleanApp bi mogla optimizirati raspoređivanje čistača na različite lokacije u stvarnom vremenu, uzimajući u obzir lokaciju korisnika, raspoloživost radnika i vremenske zahtjeve usluga. Na taj način bi se poboljšala efikasnost platforme, omogućilo brzo izvršavanje zadataka, te optimizirali troškovi i vrijeme putovanja, čime bi se značajno podiglo ukupno korisničko zadovoljstvo.

5. Zaključak

Tijekom ovog rada prikazan je razvoj i primjena **genetskog algoritma** za rješavanje složenog problema optimizacije rasporeda radnika i zadataka. Ovaj pristup pokazao se učinkovitim za **statičko** i **dinamičko** dodjeljivanje zadataka u realnom vremenu, omogućujući sustavu prilagodbu promjenjivim uvjetima poput novih zahtjeva korisnika ili promjena u dostupnosti radnika. Primjena genetskog algoritma pruža fleksibilnost i sposobnost prilagodbe, što je ključno u industrijama gdje **vrijeme** i **resursi** igraju važnu ulogu.

Problemi poput ovih rijetko su jednostavni te zahtijevaju pažljivo balansiranje između optimizacije troškova i efikasnog upravljanja resursima. Iako su rješenja razvijena u ovom radu napredna i primjenjiva, specifičnosti industrije često donose dodatne izazove. U svakom slučaju ispravna prilagodba i primjena algoritma može značajno poboljšati učinkovitost raspoređivanja i minimizirati kašnjenja.

Postoji nekoliko smjerova za daljnje poboljšanje. Jedan od njih je istraživanje **genetskog programiranja** [10] [11], koje bi omogućilo automatsko otkrivanje pravila za raspoređivanje poslova. Također, kombiniranje genetskog algoritma s **lokalnim pretraživanjem** [12], **simuliranim kaljenjem** [13] ili nekim drugim heuristikama [14] može dovesti do boljih rezultata u složenijim okruženjima. To su neke od ideja, ali naravno sve to treba isprobati i ispitati.

Zaključno, rješenja razvijena u ovom radu pružaju čvrstu podlogu za optimizaciju rasporeda radnika i zadataka, ne samo u uslugama na zahtjev, već i u širem spektru aplikacija koje zahtijevaju dinamično upravljanje resursima u stvarnom vremenu.

Sažetak

Primjena problema usmjeravanja vozila s vremenskim ograničenjima u uslugama na zahtjev

Magda Smolić - Ročak

Ovaj diplomski rad bavi se problemom usmjeravanja vozila s vremenskim okvirima (VRPTW) u kontekstu usluga na zahtjev, poput usluga autotaksi prijevoza ili dostave. Razvijen je model koji optimizira rute radnika koristeći genetski algoritam, s ciljem smanjenja troškova i povećanja efikasnosti. U radu su ostvarena različita rješenja za statičke i dinamičke scenarije, uključujući prilagodbu radnog vremena i različite tipove zadataka. Prikazane su i mogućnosti proširenja modela te praktična primjena u stvarnim sustavima.

Ključne riječi: problem usmjeravanja vozila s vremenskim okvirima; VRPTW; VRP; genetski algoritmi; usluge na zahtjev

Abstract

Application of time-constrained vehicle routing problem in on-demand services

Magda Smolić - Ročak

This thesis addresses the Vehicle Routing Problem with Time Windows (VRPTW) in the context of on-demand services, such as taxi services or deliveries. A model was developed to optimize worker routes using a genetic algorithm, with the goal of reducing costs and increasing efficiency. The thesis implements various solutions for both static and dynamic scenarios, including adjustments for working hours and different task types. Moreover, the work discusses possible avenues for improvement and practical applications.

Keywords: Vehicle Routing Problem with Time-Windows; VRPTW; VRP; genetic algorithms; on-demand services

Literatura

[1] K. N. Ayesha Maroof, Berk Ayvaz, "Logistics optimization using hybrid genetic algorithm (hga): A solution to the vehicle routing problem with time windows (vrptw)", 2024.

[2] S. I. M. Y. Hideki Hashimotoa, Toshihide Ibarakib, "The vehicle routing problem with flexible time windows and traveling times", 2006.

[3] N. A. El-Sherbeny, "Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods", *Journal of King Saud University-Science*, sv. 22, br. 3, str. 123–131, 2010.

[4] A. G. Qureshi, E. Taniguchi, i T. Yamada, "Exact solution for the vehicle routing problem with semi-soft time windows and its application", *Procedia - Social and Behavioral Sciences*, sv. 2, br. 3, str. 5931–5943, 2010., the Sixth International Conference on City Logistics. https://doi.org/10.1016/j.sbspro.2010.04.008

[5] "Heuristics for vehicle routing problem - a survey and recent advances", 2010.

[6] R. P. M. W. E. E. S. F. R. K. A. Elinor Jernheden, Carl Lindstrom, "Comparison of exact and approximate methods for the vehicle routing problem with time windows", 2020.

[7] A. M. Roberto Baldacci, "A unified exact method for solving different classes of vehicle routing problems", 2007.

[8] T. V. Matthew, "Genetic algorithm", https://developers.google.com/optimization/routing/vrp, 2012.

[9] S. d. C. S. Ana Madureira, Carlos Ramos, “Using genetic algorithms for dynamic scheduling”, 2003.

[10] L. B. Domagoj Jakobović, “Dynamic scheduling with genetic programming”, 2006.

[11] K. B. J. F. T. C. D. D. Domagoj Jakobović, Marko Đurasević, “Evolving dispatching rules for dynamic vehicle routing with genetic programming”, 2023.

[12] F. H. Beatrice Ombuki, Brian J. Ross, “Multi-objective genetic algorithms for vehicle routing problem with time windows”, 2006.

[13] C.-C. L. Shih-Wei Lin, Vincent F. Yu, “A simulated annealing heuristic for the truck and trailer routing problem with time windows”, 2011.

[14] D. J. Matija Gulić, “Evolution of vehicle routing problem heuristics with genetic programming”, 2011.