Univerzitet u Beogradu

Matematički fakultet

Heuristički pristup rešavanju problema raspoređivanja i preraspoređivanja vozila hitne pomoći po baznim stanicama

Master rad

Student
Stefan Janković
1054/2011

Mentor
prof. dr Zorica Stanimirović
Matematički fakultet, Beograd

Beograd, septembar 2015.

Mentor:	prof. dr Zorica Stanimirović Matematički fakultet, Univerzitet u Beogradu
Članovi komisije:	doc. dr Miroslav Marić Matematički fakultet, Univerzitet u Beogradu
	doc. dr Filip Marić Matematički fakultet, Univerzitet u Beogradu

Datum odbrane: _____

Heuristički pristup rešavanju problema raspoređivanja i preraspoređivanja vozila hitne pomoći po baznim stanicama

Apstrakt:

U radu je prikazana primena metode promenljivog spusta za rešavanje problema optimizacije rada službe hitne pomoći. Imajući u vidu da je problem NP-težak, velika pažnja se posvećuje heurističkim metodama. Ovde je za rešavanje problema korišćen dinamički model, u kome se dopušta premeštanje vozila između različitih baznih stanica tokom različitih perioda dana da bi se maksimizovala efikasnost na lokacijama u kojima se očekuje veliki broj incidenata. Po prvi put je korišćena metoda promenljivih okolina na ovom problemu kojim su se po prvi put rešile instance većih dimenzija. Za instance manjih dimenzija, implementiran je do sada najbolji poznati algoritam u CPLEX rešavaču, i rezultati dva algoritma su upoređeni. Predložen metod je dao optimalna rešenja za sve upoređene instance.

Heuristic approach to solving problem of location and relocation of ambulance vehicles in base stations

Abstract:

An application of Variable Neighborhood Search method was demonstrated on solving problem of optimizating the work of ambulance service. As this problem is NP-hard, great attention was given to heuristic methods. A dynamic model which allows for relocation of ambulance vechiles between different base locations during different day intervals was used. Purpose of dynamic model is to allow for ambulance to respond on location where incidents are expected, when they are expected. For the first time, Variable Neighborhood Search method was used on this problem, and solutions for large instances were obtained for the first time. For instances of smaller dimensions, currently best known Linear Programming algorithm was implemented in CPLEX solver and results were compared with solution presented in this paper. Novel method was giving optimal solutions for all small test instances.

Zahvalnica

Zahvaljujem se Gradskom zavodu za hitnu medicinsku pomoć Beograda za obezbeđivanje podataka korišćenih tokom pisanja ovog rada. Posebno se zahvaljujem, dr. Goranu Čolakoviću, Svetlani Bogdanović, Zagi Maksimović, Nadi Emiš i Mirku Jariću, zaposlenima u zavodu. Želim još da se zahvalim i mentoru, prof. dr Zorici Stanimirović i članovima komisije doc. dr Miroslavu Mariću i doc. dr Filipu Mariću na korisnim sugestijama i primedbama prilikom pisanja ovog rada, kao i Stefanu Miškoviću za pomoć tokom pisanja istog.

SADRŽAJ

Apstrakt/Abstract…… 3
Zahvalnica…… 4
SADRŽAJ…… 5
1. UVOD…… 6
2. PROBLEMI KOMBINATORNE OPTIMIZACIJE…… 7
2.1 Složenost algoritma…… 9
2.2 Linearno programiranje…… 14
3. METAHEURISTIČKE METODE…… 16
3.1 Populacione metode…… 16
3.1.1 Genetski algoritmi…… 17
3.1.2 Optimizacija rojem čestica…… 18
3.1.3 Optimizacija mravljim kolonijama…… 19
3.1.4 Optimizacija rojem pčela…… 20
3.2 Metode zasnovane na lokalnom pretraživanju…… 21
3.2.1 Simulirano kaljenje…… 23
3.2.2 Tabu pretraga…… 24
3.2.3 Iterativna lokalna pretraga…… 24
4. METODA PROMENLJIVIH OKOLINA…… 26
4.1 Metoda promenljivog spusta…… 26
4.2 Redukovana metoda promenljivih okoline…… 27
4.3 Osnovna metoda promenljivih okolina…… 28
5. FORMULACIJA PROBLEMA…… 29
5.1 Planiranje i rad hitne pomoći…… 29
5.2 Dosadašnji rezultati…… 29
5.3 Matematička formulacija problema…… 30
6. PRIMENA METODA OKOLINA NA REŠAVANJE PROBLEMA…… 33
6.1 Lokalna pretraga…… 33
6.2 Korak razmrdavanja…… 34
6.2.1 Slučajno premeštanje…… 34
6.2.2 Premeštanje iz popunjene stanice…… 35
6.2.3 Premeštanje iz praznih u popunjene stanice…… 35
6.2.4 Zamena prazne i popunjene stanice…… 36
6.3 Struktura primenjene osnovne metode promenljivih okolina…… 37
7. ANALIZA REZULTATA…… 39
7.1 Test instance…… 39
7.2 Rezultati dobijeni za realnu test instancu…… 41
7.3 Rezultati na generisanim instancama manjih dimenzija…… 43
7.4 Rezultati na velikim generisanim primerima…… 44
8. Zaključak…… 46
LITERATURA…… 47

1. UVOD

Poslednjih decenija, kao posledica naglog tehnološkog napretka, sve veći broj problema se uspešno rešava uz pomoć računara. Međutim, i dalje postoje teški problemi koji se ne mogu u potpunosti rešiti grubom upotrebom procesorske moći.

Problemi pokrivanja su problemi u kojima se odgovara na pitanje da li izvesna kombinatorna struktura „pokriva“ drugu strukturu, odnosno da li podprostor određen jednim skupom tačaka u nekom prostoru sadrži sve tačke iz drugog skupa. U kontekstu optimizacionih problema, traži se da se odredi optimalna kombinacija tačaka, tako da sve ostale tačke pripadaju prostoru koje te tačke pokrivaju. Najistaknutiji teorijski primeri su minimalni pokrivajući podskup, problem pokrivajućih čvorova kao i problem pokrivajućih ivica [27]. U praksi, primena se nalazi u određivanju centara mobilne telefonije, planiranje lokacija supermarketa, zdravstvenih ustanova, škola, i tako dalje [17]. Mnogi od ovih problema su NP-teški, pa su algoritmi koji nalaze optimalno rešenje u praksi, zbog njihove neefikasnosti, neprimenljivi. Zbog toga se pribegava heurističkim metodama.

U radu će biti razmatran problem optimizacije Sistema službe hitne pomoći. Cilj problema je da se odredi optimalan raspored vozila službe hitne pomoći po zadatim baznim stanicama, kao i preraspoređivanja vozila iz jedne bazne stanice u drugu tokom različitih razmatranih perioda, tako da se maksimizuje sposobnost službe da reaguje na očekivane hitne slučajeve sa što manjim troškovima. Biće korišćen višeperiodni probabilistički model pokrivanja koji je predložen u radu [58]. Problem će biti rešavan metodom promenljivih okolina [25].

Rad je organizovan na sledeći način. U glavi 2 se uvode NP-kompletni problemi i specijalno, problemi kombinatorne optimizacije i neke metode za njihovo efikasno rešavanje, pre svega egzaktne metode i primeri njihove primene, kao i heurističke metode. U glavi 3 se govori o klasi heurističkih metoda, metaheurističkim metodama, i nekim njihovim primerima. U glavi 4 se definiše metoda promenljivih okolina, specifična metaheuristička metoda, dok se u glavi 5 formalno definiše problem. U glavi 6 opisuje se način na koji se metoda promenljivih okolina primenjuje na rešavanje problema. Konačno, u glavama 7 i 8 su izneti eksperimentalni rezultati i njihova analiza, doprinos ovog rada i pravci za dalje istraživanje.

2. PROBLEMI KOMBINATORNE OPTIMIZACIJE

Kombinatorna optimizacija spada u polje matematičke optimizacije koje je povezano sa operacionim istraživanjima, teorijom algoritama i računske kompleksnosti. Cilj kombinatorne optimizacije je da pronađe optimalan objekat, ili kombinacija objekata iz nekog konačnog ili prebrojivo beskonačnog skupa [52].

Veliki broj realnih problema se mogu formulisati kao problemi kombinatorne optimizacije. Važna oblast primene je upravljanje resursima u cilju povećanja produktivnosti. Drugi primeri su problemi upravljanja proizvodnjom, raspoređivanja poslova i mašina, optimalnog transporta, optimalnog dizajna i sl. U poslednje vreme sve češće su i primene u molekularnoj biologiji, fizici i kristalografiji [9]. U matematici se pomoću kombinatorne optimizacije mogu rešavati problemi u kombinatorici, teoriji grafova i logici [9].

Jedan način da se definiše problem kombinatorne optimizacije je sledeći:

Definicija 1. Neka je dat diskretan skup *S*. Neka je funkcija $f : S \rightarrow \mathbb{R}$. Naći minimum funkcije f na skupu *S*.

Skup *S* se naziva *dopustivi skup*, a funkcija *f funkcija cilja*. Za tačku $x \in S$ se kaže da je *dopustivo rešenje problema*. Definicija 1 podrazumeva da je potrebno naći jedno ili sva dopustiva rešenja *x** takva da je $f(x^*) = \min_{x \in S} f(x)$. Takva rešenja se nazivaju *optimalna rešenja*.

Kako je $\min_{x \in S} f(x) = -\max_{x \in S}\big(-f(x)\big)$, problem minimizacije je ekvivalentan problemu maksimizacije, te je dovoljno analizirati samo jedan od ova dva problema. Skup *S* je diskretan, i kao takav može biti konačan ili prebrojiv. Ukoliko je skup *S* konačan, problem kombinatorne optimizacije se još naziva i *problem određivanja ekstremne vrednosti funkcije na konačnom dopustivom skupu*.

Primer 1. Dat je skup od n gradova koje trgovački putnik treba da poseti po jedanput takvim redosledom da troškovi puta budu minimalni. Ovaj problem se naziva *problemom trgovačkog putnika* [37]. Dopustivi skup rešenja *S* je ovde skup svih $n!$ permutacija skupa $\{1, 2, \dots, n\}$. Vrednost funkcije cilja *f(x)* za neku permutaciju $x = j_1, j_2, \dots, j_n$ se računa kao zbir troškova putovanja između svaka dva uzastopna grada.

Jasno je da se za primer problema trgovačkog putnika može pretražiti svih $n!$ rešenja. Potpuna pretraga se može izvršiti i za bilo koji drugi problem gde je skup *S* konačan. Međutim, očigledno je da ako skup *S* ima mnogo elemenata ili je prebrojiv, potpuna pretraga bi zahtevala nepraktično mnogo vremena. Na primer, ako bi elementi skupa *S* bili nizovi od 100 binarnih promenljivih, potpuna pretraga bi zahtevala pretragu 2^{100} rešenja. Ako pretpostavimo da procena vrednosti jednog rešenja zahteva 10^{-10} sekundi, potpuna

pretraga bi trajala preko 3×10^{12} godina. Osnovni cilj kombinatorne optimizacije je konstrukcija efikasnih algoritama za pronalaženje rešenja. Ukoliko je moguće, korisno je konstruisati algoritam za pronalaženje egzaktnog rešenja. Ukoliko takav algoritam ne postoji, pristupa se približnom rešavanju pomoću odgovarajućih heurističkih metoda [9].

Postoje različiti tipovi problema kombinatorne optimizacije u zavisnosti od tipa ograničenja i funkcije cilja. *Linearno programiranje* je problem koji pripada oblasti matematičke optimizacije, u kome je cilj pronaći najbolju (minimalnu ili maksimalnu) vrednost unutar matematičkog modela koji se može predstaviti linearnim relacijama. On pripada oblasti kombinatorne optimizacije ukoliko je model ograničen na celobrojne vrednosti.

Primer 2. *Problem linearnog celobrojnog programiranja* se definiše na sledeći način:

$$\min_{x \in S} c^T x, \qquad S = \{x \in Z^n \mid Ax \leq b\}$$

u kome je *A* celobrojna matrica dimenzija $m \times n$, a *c* i *b* celobrojni vektori odgovarajućih dimenzija. Ovakav zapis se naziva *simetrični oblik*. Postoje još i *opšti* i *standardni oblik*, o čemu će više reči biti u poglavlju 2.2.

Primer 3. *Problem linearnog programiranja* se definiše na sledeći način:

$$\min_{x \in S} c^T x, \qquad S = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

Razlika između prethodna dva primera je da je pri linearnom programiranju relaksiran uslov celobrojnosti. Elementi matrice *A* i vektora *c* i *b* mogu imati realne vrednosti. Problem celobrojnog linearnog programiranja se često rešava uzastopnim relaksacijama uslova celobrojnosti. Važan specijalan slučaj celobrojnog linearnog programiranja je:

Primer 4. *Problem 0-1 linearnog programiranja* se definiše na sledeći način:

$$\min_{x \in S} c^T x, \qquad S = \{x \in B^n \mid Ax \leq b\}$$

gde je $B = \{0, 1\}$. Matrica *A* i vektori *c* i *b* su celobrojni. Predstavimo konkretan primer problema linearnog programiranja.

Primer 5. Neka na farmi površine $L\ km^2$ treba posaditi kombinaciju dve vrste useva, koji daju profit od S_1 i S_2 po kvadratnom kilometru, i neka svaka od vrsta zahteva F_1 kilograma đubriva i P_1 kilograma insekticida, odnosno F_2 kilograma đubriva i P_2 kilograma insekticida po kvadratnom kilometru. Ukoliko je moguće koristiti samo ograničenu količinu đubriva F i insekticida P, odrediti optimalne površine pod kojima treba posaditi dve vrste da bi se maksimizovao profit.

Ovaj problem pripada problemu linearnog programiranja, i može se predstaviti formulama:

Maksimizovati:	$S_1x_1 + S_2x_2,$	maksimizovati profit
Pri ograničenjima:	$x_1 + x_2 \leq L$	ograničiti na dozvoljenu površinu
	$F_1x_1 + F_2x_2 \leq F$	ograničenje đubriva
	$P_1x_1 + P_2x_2 \leq P$	ograničenje insekticida
	$x_1 \geq 0, x_2 \geq 0$	nije moguće zasaditi negativnu površinu

Osim problema linearnog programiranja, poseban tip problema kombinatorne optimizacije predstavljaju optimizacioni problemi na grafovima.

Primer 6. Za dati graf $G = (V, E)$ pronaći put *p* koji prolazi kroz sve tačke grafa tačno jednom. Ovo je problem pronalaženja *Hamiltonovog puta* [1]. Može se primetiti da je ovaj problem sličan problemu trgovačkog putnika. Problem pronalaženja Hamiltonovog puta se može rešavati nad usmerenim ili neusmerenim grafom. Takođe, moguće je tražiti zatvoren ili otvoren Hamiltonov put.

2.1 Složenost algoritama

Neformalno, algoritam za rešavanje unapred fiksirane klase partikularnih slučajeva nekog problema, je konačan spisak pravila pomoću kojih se, ako postupamo po njima, dolazi do rešenja bilo kojeg partikularnog slučaja iz zadate klase, u konačno mnogo koraka [9].

Problemi čije rešenje pripada binarnom skupu, najčešće skupu {„da", „ne"}, se nazivaju *problemima odlučivanja*. Ukoliko je poznat skup mogućih ulaznih podataka, problem odlučivanja se naziva *određenim*.

Može se primetiti da nisu svi problemi problemi odlučivanja. Međutim, razne vrste problema se mogu svesti na probleme odlučivanja, i obrnuto. U primeru 1 je naveden problem trgovačkog putnika kao optimizacioni problem: „Odrediti najkraći put trgovačkog putnika". Varijanta ovog problema u obliku problema odlučivanja glasi: „Da li postoji put trgovačkog putnika čija dužina nije veća od zadatog broja *D*". Može se primetiti da se koristeći algoritam za rešavanje jednog problema može rešiti i drugi problem. Jasno je da ako je poznat odgovor na pitanje „Koja je najmanja dužina puta?" može se odgovoriti i na svako pitanje oblika „Da li postoji put dužine *D*?". Slično, ako se može ustanoviti da li postoji put dužine *D*, rešavanjem datog problema za više različitih vrednosti *D* može se ustanoviti i minimalna dužinu puta.

Ako se, jednostavnosti radi, pretpostavi da su ulazni podaci problema kodirani na neki pogodan način koristeći konačan broj simbola iz nekog konačnog skupa, analiza složenosti algoritma se zasniva na analizi vremena izvršavanja algoritma, relativno u odnosu na veličinu ulaznih podataka. U teorijskom razmatranju, uzima se da za svaki algoritam postoje elementarni koraci koji se mogu obaviti za fiksno vreme, bez obzira na ulazne podatke i

trenutno stanje mašine koja izvršava algoritam. Ovime se obezbeđuje da vreme rada algoritma bude srazmerno broju elementarnih koraka obavljenih tokom izvršavanja algoritma.

Neka se veličina ulaza može opisati preko niza brojeva $\{n_1,\ n_2, \dots, n_k\}$, i neka funkcija $f(n_1,\ n_2, \dots, n_k)$ izražava vreme rada algoritma. Data funkcija je jednaka najvećem broju koraka potrebnim za izvršavanje algoritma za sve ulazne podatke date veličine. Radi jednostavnosti, u daljem tekstu će se veličina ulaznih podataka opisivati kroz samo jedan broj *n*. Na primer, ako su ulazni podaci kodirani u obliku matrice dimenzija $m \times n$, u zavisnosti od problema može postojati potreba posmatrati dimenzije *m* i *n* odvojeno.

Definicija 2. Neka su funkcije $f(n)$ i $g(n)$ dve pozitivne funkcije definisane na skupu prirodnih brojeva. Ako postoji pozitivna konstanta *c* takva da je $f(n) \leq cg(n)$ za svaki prirodan broj n, tada se kaže $f(n) = O\big(g(n)\big)\ kad\ n \to \infty$.

Za polinom $p_k(n)$ stepena k važi relacija $p_k(n) = O(n^k)\ kad\ n \to \infty$. Za dati algoritam se kaže da je *polinomski* ukoliko je $f(n) = O(n^k)\ kad\ n \to \infty$, gde je $f(n)$ funkcija koja daje vreme rada algoritma za najnepovoljniji ulazni problem dužine *n* [8].

Ako se problem i njegovo algoritamsko rešavanje razmatra izvan nekog formalnog sistema ili jezika, onda se prethodno moraju odrediti definicija dimenzija problema i definicija elementarnog koraka u algoritmu. Pokazuje se da, sve dok su ove definicije smislene, izvesna proizvoljnost u njihovom izboru ne utiče značajno na zaključke razmatranja. Problem i njegovo algoritamsko rešavanje se mogu razmatrati kao deo nekog formalnog sistema, na primer Tjuringove mašine [27]. Za Tjuringovu mašinu važi da je dimenzija problema broj simbola u nizu ulaznih simbola kojim je kodiran ulazni podatak, a elementarni korak algoritma je korak Tjuringove mašine.

Definicija 3. Problem odlučivanja pripada klasi *P* polinomskih problema, ako postoji algoritam *A* za rešavanje tog problema i polinom $p(n)$ tako da algoritam *A* završava rad za ne više od $p(n)$ koraka za svaki od partikularnih slučajeva problema dimenzije n. Kaže se takođe da algoritam *A* ima *polinomsku kompleksnost*, da se izvršava u *polinomskom vremenu* i da ima osobinu *polinomstva* [27].

Ako je ulaz za neki problem neki ceo broj *n*, ili niz celih brojeva, tada se taj broj može kodirati u $\log n$ simbola.

Definicija 4. Algoritam je *slabe polinomske složenosti*, ako je vreme izvršavanja polinomski zavisno u odnosu na *n*. Ukoliko je vreme izvršavanja ograničeno polinomom u odnosu na $\log n$, tada se kaže da je algoritam *jake polinomske složenosti*.

Polinomski algoritmi se smatraju *dobrim* algoritmima, a problemi koji se mogu rešiti polinomskim algoritmima *laki* problemi. Algoritmi čije je vreme rada ograničeno funkcijom oblika $c^n, (c > 1)$, se nazivaju *eksponencijalnim* algoritmima. Eksponencijalni algoritmi se

smatraju *lošim*, a problemi za čije rešavanje nije poznat polinomski algoritam *teškim*. Čak i najbržim računarima je potrebno nepraktično mnogo vremena za njihovo izvršavanje već za male vrednosti *n*.

Postoje problemi za koje je dokazano da svi algoritmi za njihovo rešavanje imaju eksponencijalnu složenost. Međutim, postoje i problemi za koje je poznat eksponencijalni algoritam ali se ne zna da li postoji i polinomski algoritam. Jedna takva klasa problema su *nedeterministički polinomski problemi*. Problemi koji su prethodno razmatrani se nazivaju *determinističkim problemima*.

Neformalno, *nedeterministički algoritam* se definiše kao algoritam koji sadrži korake u kojima dolazi do grananja rada algoritma na dva (ili više) nezavisna dela koji se (kao deo algoritma) obavljaju istovremeno. Broj ovakvih grananja nije fiksan, tj. raste sa veličinom ulaza. Rad nedeterminističkog algoritma najpogodnije se predstavlja grafom oblika stabla. Početak rada algoritma je predstavljen korenom stabla. Rešenje problema odlučivanja, izraženo jednim „da" može da se dobije u više grana algoritma i u različitim vremenima (tj. posle različitog broja elementarnih koraka algoritma), ali rešenje ne mora da se dobije u svakoj grani. Broj elementarnih koraka potrebnih za dobijanje rešenja određuje se zbirom brojeva elementarnih koraka koji realizuju grane algoritma duž najkraćeg puta u korenskom stablu od korena do mesta rešenja.

Kaže se da nedeterministički algoritam rešava zadati problem u polinomskom vremenu ako je za svaki ulazni problem odlučivanja sa pozitivnim rešenjem broj elementarnih koraka od početka rada algoritma do najbližeg mesta gde se dobija odgovor „da" ograničen polinomskom funkcijom promenljive *n*. Ovakav algoritam se naziva *nedeterministički polinomski algoritam* [27].

Definicija 5. Klasa problema odlučivanja koji se mogu rešiti u polinomskom vremenu pomoću nedeterminističkog algoritma označava se sa *NP*. Problem koji pripada klasi *NP* naziva se *NP-problem.*

Oznaka *NP* proizilazi iz „nedeterministički polinomski". NP-problemi se mogu prepoznati i na sledeći način. Za ove probleme je karakteristično da se potvrdan odgovor može verifikovati u polinomskom vremenu. Na primer, ako se posmatra problem trgovačkog putnika u obliku problema odlučivanja – „Da li postoji put dužine manje ili jednake *D*", ako se zna da postoji put kraći od datog broja *D*, u linearnom vremenu se može izračunati njegova dužina i uporediti se sa brojem *D.*

Ako postoji nedeterministički polinomski algoritam za rešavanje nekog problema, onda sigurno postoji deterministički eksponencijalni algoritam [27]. Deterministički polinomski algoritam je specijalan slučaj nedeterminističkog polinomskog algoritma, a klasa problema *P* je podklasa klase problema *NP*. Problemi koji pripadaju klasi *NP* a ne pripadaju klasi *P* se odlikuju time što su poznati algoritmi za njihovo rešavanje eksponencijalne kompleksnosti, pri čemu nije poznato da li postoji algoritam polinomske kompleksnosti.

Može se još primetiti da postoje algoritmi koji ne pripadaju ni jednoj od klasa *NP* i *P*. Na primer, problem za koji se može konstruisati eksponencijalni algoritam, za koje se može dokazeti da polinomski algoritam ne postoji, ali istovremeno je potreban eksponencijalan algoritam za verifikaciju rešenja.

Neka su *A* i *B* problemi odlučivanja. Neka se za problem *A* može konstruisati polinomski algoritam u kome se kao jedan od koraka pojavljuje algoritam za rešavanje problema *B*. Tada se kaže da se *problem A svodi u polinimskom vremenu na problem B*. Ako u ovoj situaciji za problem *B* postoji polinomski algoritam, onda se očigledno i za problem *A* može konstruisati polinomski algoritam.

Problem A se polinomski transformiše u problem B ako se za svaki specijalan slučaj *X* problema *A* može u polinomskom vremenu pronaći specijalan slučaj *Y* problema *B* tako da je za slučaj *X* odgovor potvrdan ako i samo ako je i za slučaj *Y* odgovor potvrdan.

Definicija 6. NP-problem je *NP-kompletan* ako za svođenje bilo kojeg NP-problema na posmatran problem postoji polinomski algoritam, tj. ako se bilo koji NP-problem polinomski transformiše u posmatrani problem.

Postojanje NP-kompletnih problema je dokazano. Neki primeri su već pominjani problemi trgovačkog putnika, problem klika, problem linearnog celobrojnog programiranja, kao i SAT problem, itd. [27]. Bitno je naglasiti da ako se za bilo koji NP-kompletan problem pronađe polinomski algoritam, time se dokazuje postojanje polinomskog algoritma za svaki NP-kompletan problem. Konstruktivni dokaz prethodne tvrdnje bi imao nemerljive praktične posledice. Međutim, mnogi empirijski i teorijski razlozi navode na hipotezu da važi $P \neq NP$ [7, 27].

Dokazivanje da je neki problem odlučivanja NP-kompletan obično se izvodi tako što se prvo dokaže da posmatrani problem pripada klasi *NP*, a zatim da sadrži kao specijalan slučaj neki poznati NP-kompletan problem [27]. Nijedan poznati algoritmi za rešavanje NP-problema nema polinomsku kompleksnost [9, 27]. Problem koji nije problem odlučivanja a čije rešavanje nije jednostavnije od rešavanja nekog NP-kompletnog problema se naziva *NP-težak problem*. Na primer, optimizaciona varijanta problema trgovačkog putnika je NP-teška.

Eksponencijalni algoritmi se ne mogu primeniti na problem čak i vrlo skromnih dimenzija zbog toga što je za njihovo izvršavanje potrebno nepraktično mnogo vremena. U oblasti kombinatorne optimizacije nailazi se na probleme čije rešavanje uz pomoć egzaktnih algoritama (algoritama za tačno rešavanje problema) nije adekvatno zbog velikog utroška računarskog vremena. Zbog toga se odustaje od egzaktnih algoritama i pribegava *heuristikama* [50]. Heuristike ne garantuju nalaženje rešenja problema, ali dobre heuristike rešavaju problem u velikom broju slučajeva. Kod optimizacionih problema algoritam nalazi optimalno rešenje, a heuristike daju suboptimalna rešenja. Dobre heuristike daju suboptimalna rešenja koja su bliska optimalnom rešenju. Heuristike se ponekad nazivaju *heurističkim algoritmima* ili *približnim algoritmima*.

Definicija 7. Kompletan graf $G = (V, E)$ je graf u kome su svaka dva čvora povezana ivicom. Graf $G' = (V', E')$ je podgraf grafa G, ako svi čvorovi i sve ivice grafa *G'* formiraju podskupove čvorova i ivica grafa *G*. *Klika* u grafu *G* je kompletan podgraf grafa *G*.

Kompletni podgrafi se prvi put pominju u [16], kao deo teorijske reformulacije Ramzijeve teorije. Termin „klika“ se prvi put pojavljuje u [41].

Primer 7. Za dati graf $G = (V, E)$ naći maksimalnu kliku. Ovo je primer opšte poznatog i proučavanog grafovskog NP-kompletnog problema. Trenutno najbolji aproksimativni algoritam sa zagarantovanom preciznošću pronalazi kliku sa brojem čvorova unutar faktora $O\left(n\frac{(\log\log n)^2}{\log^3 n}\right)$ u odnosu na maksimum. Stepen aproksimacije ovog algoritma je slab, ali dosad najbolji poznat, i ovo predstavlja primer problema za čije se rešavanje koriste heurističke metode bez zagarantovane preciznosti.

Definicija 8. *Bojenje grafa* $G = (V, E)$ skupom $X = \{x_1, x_2, \dots, x_k\}$ predstavlja pridruživanje elemenata skupa *X* svakom čvoru iz skupa *V*. Pridruživanje je *propisno* ako i samo ako su za svaka dva čvora iz skupa *V*, za koje postoji ivica iz skupa *E*, njima odgovarajući pridruženi elementi x_i i x_j različiti.

Po dogovoru, elementi skupa *X* se nazivaju *bojama*, odakle i potiče naziv „bojenje grafa”.

Definicija 9. Propisno bojenje koje koristi najviše *k* boja se naziva *k-bojenjem*. Najmanji broj *k'* za koje postoji k'-bojenje se *naziva hromatskim brojem grafa*.

Više o bojenju grafova i osobinama hromatskog broja se može naći u [2].

Primer 8. Za dati graf $G = (V, E)$ pronaći njegov hromatski broj.

Varijanta ovog primera koji spada u problem odlučivanja, utvrditi da li dati graf ima hromatski broj manji ili jednak *k* je NP-kompletan problem za svako *k* veće od 2. Više detalja o problemima odlučivanja i NP-kompletnosti se može naći u narednoj glavi.

Jedna primena problema bojenja grafa je problem zakazivanja. Neka postoji izvestan broj zadataka koji moraju biti obavljeni, i neka neki od tih zadataka ne mogu biti obavljeni u isto vreme. Na primer, zadaci mogu zahtevati isti resurs. Hromatski broj grafa čiji svi čvorovi odgovaraju zadacima, a sve ivice odgovaraju konfliktima između zadataka odgovara minimalnom vremenu potrebnom da se ti zadaci obave.

2.2 Linearno programiranje

Linearno programiranje je skup metoda za rešavanje optimizacionih problema unutar matematičkog modela čiji su uslovi reprezentovani linearnim relacijama. *Linearno celobrojno programiranje* u kome su sve promenljive celi brojevi je NP-težak problem [54]. Celobrojna varijanta ovog problema se može rešiti takozvanim *relaksacijama*, čime se problem svodi na problem linearnog programiranja.

Definicija 10. Neka je $A = \left[a_{ij}\right]_{m\times n}$ data realna matrica sa vrstama $V_1, \dots, V_m$ i neka su $b \in \mathbb{R}^m$ i $c \in \mathbb{R}^n$ dati vektori. *Opšti oblik* problema linearnog programiranja je:

$$min\ c^T x,$$

$$V_i^T x = b_i, \qquad i \in I_1,$$

$$V_i^T x \geq b_i, \qquad i \in I_2,$$

$$V_i^T x \leq b_i, \qquad i \in I_3,$$

$$x_j \geq 0, \qquad j \in J,$$

gde je $I_1 \cup I_2 \cup I_3 = \{1, \dots, m\}$, $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$, $I_2 \cap I_3 = \emptyset$ i $J \in \{1, \dots, n\}$. Ovaj zapis podrazumeva da se traži minimalna vrednost linearne funkcije cilja $f(x) = c^T x$ uz uslov da njeni argumenti zadovoljavaju ograničenja definisana navedenim linearnim jednačinama i nejednačinama.

Ukoliko je $I_1 = \{1, \dots, m\}$, $J = \{1, \dots, n\}$, opšti oblik se može svesti na *standardni oblik* problema linearnog programiranja:

$$min\ c^T x,$$

$$Ax = b,$$

$$x \geq 0,$$

dok se u slučaju $I_2 = \{1, \dots, m\}, J = \{1, \dots, n\}$ dobija *simetrični oblik*:

$$min\ c^T x,$$

$$Ax \geq b,$$

$$x \geq 0,$$

Lako je videti da se problem linearnog programiranja zadat u opštem obliku uvek može transformisati u standardni i simetrični oblik. Da bi se došlo do standardnog oblika, potrebno je eliminisati ograničenja tipa nejednačina, što se čini pomoću takozvanih izravnavajućih promenljivih $s_i, i \in I_2 \cup I_3$. Naime, nejednačine $V_i^T x \geq b_i, i \in I_2$ se zamenjuju jednačinama $V_i^T x - s_i = b_i, i \in I_2$, uz uslov $s_i \geq 0, i \in I_2$, dok se nejednačine $V_i^T x \leq b_i, i \in I_3$ zamenjuju

jednačinama $V_i^T x + s_i = b_i, i \in I_3$ uz uslov $s_i \geq 0, i \in I_3$. Najzad, svaka promenljiva $x_j,\ j \in \{1, \dots, n\} \backslash J$, se može zameniti u funkciji cilja i svim ograničenjima sa $x_j^+ - x_j^-$, pri čemu je $x_j^+ \geq 0,\ x_j^- \geq 0$, i na taj način doći do problema kod koga je na sve promenljive nametnut uslov nenegativnosti.

Da bi se problem iz opšteg oblika sveo na simetrični oblik potrebno je eliminisati ograničenja tipa jednačina. Jasno je da se jednačine $V_i^T x \geq b_i, i \in I_1$ mogu ekvivalentno zameniti nejednačinama $V_i^T x \geq b_i, \left(-V^i\right)^T x \geq -b_i, i \in I_1$. Takođe se svako ograničenje oblika $V_i^T x \leq b_i$ može pomnožiti sa -1, što daje ekvivalentno ograničenje $(-V_i)^T x \geq -b_i$. Ako se najzad promenljive $x_j, j \in \{1, \dots, n\} \backslash J$ na isti način kao ranije zamene sa po dve nenegativne promenljive, dobija se problem simetričnog oblika.

Uzimajući u obzir prethodno navedene transformacije, lako se vidi da se ne gubi na opštosti bez obzira da li se posmatra opšti, standardni ili simetrični oblik problema linearnog programiranja.

Do sada poznati algoritmi za rešavanje problema linearnog programiranja su slabe polinomske vremenske kompleksnosti. Da li postoji jako polinomski algoritam za rešavanje problema linearnog programiranja je otvoren problem. U slučaju problema linearnog programiranja, od interesa je posmatrati veličinu ulaza kao broj simbola kojim je ulaz kodiran. Dodatno, može se dokazati da je problem linearnog celobrojnog programiranja ekvivalentan problemu 0-1 linearnog programiranja, pri čemu je pri svođenju veličina celobrojnih promenljivih eksponencijalno (nepolinomski) ograničena broju 0-1 promenljivih. Dalje, 0-1 linearno programiranje je NP-kompletan problem [30]. Stoga se slabo polinomski algoritmi, a time i metode za rešavanje problema linearnog programiranja, i dalje smatraju eksponencijalnim algoritmima. Više o algoritmima za rešavanje linearnog programiranje se može naći u [10].

U praksi se često koriste različiti komercijalni programi za rešavanje problema linearnog programiranja. U akademskim krugovima, vrlo je česta upotreba CPLEX rešavača. IBM ILOG CPLEX studio za optimizacije [57] je softver za rešavanje različitih optimizacionih problema. U dosadašnjoj literaturi se za rešavanje problema raspoređivanja i preraspoređivanja vozila hitne pomoći koristi linearno programiranje. U ovom radu je korišćen CPLEX 12.1 rešavač da bi smo dobili optimalno rešenje ili donju granicu optimalnog rešenja i dobijeni rezultati su upoređeni sa najboljim rezultatima predložene metaheurističke metode.

3. METAHEURISTIČKE METODE

Kao što je pokazano u prethodnoj glavi, velik broj standardnih problema kombinatorne optimizacije je NP-težak čime je njihovo rešavanje egzaktnim metodama nepraktično sporo. Sa druge strane, problemi kombinatorne optimizacije koji potiču iz prakse su obično i slabo strukturirani i nelinearni, što ih sve čini veoma komplikovanim i za modeliranje i za rešavanje. Zato je do sada razvijen niz specijalizovanih heurističkih metoda koje su prilagođene karakteristikama i strukturama konkretnih problema.

Međutim, poslednjih decenija se naglo razvijaju opšti heuristički pristupi rešavanju problema kombinatorne optimizacije [9]. Veliki uspeh ovakvih opštih heuristika u praksi je doveo do ubrzanog usavršavanja njihovih osnovnih koncepata, kao i do daljih teorijskih istraživanja koja treba da objasne njihovu efikasnost. Ove heuristike predstavljaju heurističke metodologije koje polaze od najopštijeg oblika problema kombinatorne optimizacije

$$\min_{x \in X} f(x)$$

gde dopustivi skup, tj. prostor dopustivih rešenja X ima prebrojivo mnogo elemenata, a funkcija cilja *f(x)* može biti bilo kakva funkcija oblika $f: X \to \mathbb{R}$. Ovakve metodologije daju opšta uputstva kako treba rešiti problem, pri čemu njihova konkretna interpretacija zavisi od specifičnosti oblika i strukture problema, i nazivaju se *metaheurističke metode*.

3.1 Populacione metode

Populacione metode (Population-Based Methods) su stohastičke metaheurističke tehnike koje su do sada uspešno primenjene na brojne realne probleme. Populacione metode iterativno primenjuju *stohastičke operatore* na grupu jedinki (populaciju). Stohastički operatori sadrže izvesnu slučajnost kao deo svoje logike. Svaka jedinka u populaciji je kodirana verzija kandidata rešenja. Rešenja se najčešće kodiraju kao niz bitova, odnosno niz celih ili realnih brojeva, tako da skup svih rešenja predstavlja hiperkocku, odnosno višedimenzionalni prostor. *Funkcija vrednosti rešenja* dodeljuje neku vrednost svakom od rešenja. Ta vrednost reflektuje potencijal (prilagođenost) jedinke da bude konačno rešenje problema. U opštem slučaju funkcija vrednosti rešenja ne mora da bude ekvivalentna funkciji čiji se optimum traži. Probabilistička primena nekog od operatora nad svakom jedinkom u populaciji postepeno dovodi populaciju do rešenja većeg kvaliteta.

Kod populacionih metoda koje spadaju u klasu iterativnih metoda se u svakom koraku trenutno rešenje, odnosno skup rešenja problema postepeno menja. Osnovna prednost populacionih metoda je da ako u okolini nekog kandidata ne postoji rešenje bolje od trenutno najboljeg, takvo rešenje može postojati u okolini druge jedinke. Sa druge strane,

izvršavanje svakog od iterativnih koraka zahteva vreme srazmerno veličini populacije. Algoritmom 1 se opisuje uopšteni algoritam populacionih metoda.

Algoritam 1. Algoritam populacionih metoda

Inicijalizacija

Definisati prostor kodiranih rešenja $\bar{X}$ i funkciju vrednosti rešenja $f(x)$

Izabrati početnu populaciju $P_1 = \{x_1^1, x_2^1, \dots, x_N^1\} \subseteq \bar{X}$

$f^* = min\{f(x_i^1),\ i = 1, \dots, N\},\ x^* = arg f^*$

$n = 1$

Iterativni korak

Odrediti vrednost $f(x_i^n)$ za svaku tačku $x_i^n,\ i = 1, \dots, N$ iz trenutne populacije P_n

Generisati $P_{n+1} = \{x_1^{n+1}, \dots, x_N^{n+1}\} \subseteq \bar{X}$ delovanjem operatora algoritma na elemente populacije P_n

$f_{min} = \min\{f(x_i^{n+1}),\ i = 1, \dots, N\}$

Ako $f_{min} < f^*$, tada $f^* = f_{min}$ i $x^* = arg f_{min}$

$n = n + 1$

Kraj

Ako je ispunjen kriterijum zaustavljanja, uzeti x^* za aproksimaciju opšteg rešenja i staje se za izvršavanjem

Kriterijum zaustavljanja može uzeti različite forme, npr. kada ukupno vreme izvršavanja algoritma pređe unapred zadatu granicu, ili ako se u unapred zadatom broju koraka nije napredovalo u trenutno najboljem rešenju. Uopšteni algoritam odgovara većini metaheurističkih metoda optimizacije, ali izvesne razlike mogu da postoje. Navedimo nekoliko primera popularnih metaheurističkih metoda i njihove osnovne karakteristike.

3.1.1. Genetski algoritmi

Prve ideje o *genetskim algoritmima* (Genetic Algorithms) su izložene 1975. u radu Holanda [26], i javile su se u okviru teorije adaptivnih sistema. Iako genetski algoritmi nisu originalno predviđeni za rešavanje optimizacionih problema, pokazano je da se oni mogu koristiti za optimizaciju nelinearnih funkcija, a u skorije vreme i kao opšta heuristička metoda [24]. Algoritam odgovara uopštenom algoritmu populacionih metoda, pri čemu operatori imitiraju neke genetske procese u biološkim sistemima – operatori selekcije, ukrštanja i mutacije, dok se za kriterijum zaustavljanja mogu uzeti različiti uslovi.

Operator selekcije se sastoji u odabiru rešenja koje će učestvovati u kreiranju nove populacije rešenja. *Operator ukrštanja* se sastoji u postupku u kome se stohastičkim procesom kombinuju delovi kodova dva slučajno odabrana rešenja iz populacije i tako dobijaju nova rešenja. Detalji operatora se prilagođavaju razmatranim problemima, uzimajući u obzir reprezentaciju jedinku, dopustivost rešenja i karakteristike problema.

Operator mutacije u opštem slučaju vrši, kao i odgovarajući genetski process, promenu sadržaja koda nekog rešenja iz *X* slučajnom zamenom pojedinih simbola ovog koda sa nekim drugim simbolima. Algoritam 2 predstavlja uopšteni genetski algoritam.

Algoritam 2. Genetski algoritam

Inicijalizacija

Definisati prostor kodiranih rešenja $\bar{X}$ i funkciju vrednosti rešenja $f(x)$

Izabrati početnu populaciju $P_1 = \{x_1^1, x_2^1, \dots, x_N^1\} \subseteq \bar{X}$

$f^* = min\{f(x_i^1),\ i = 1, \dots, N\},\ x^* = arg f^*$

$n = 1$

Iterativni korak

Odrediti $\overline{P_n} \subseteq P_n$, podskup rešenja iz trenutne populacije sa najvećom vrednošću funkcije rešenja

Generisati $P_{n+1} = \{x_1^{n+1}, \dots, x_N^{n+1}\} \subseteq \bar{X}$ delovanjem operatora selekcije, ukrštanja i mutacije na elemente skupa $\overline{P_n}$

$f_{min} = \min\{f(x_i^{n+1}),\ i = 1, \dots, N\}$

Ako $f_{min} < f^*$, tada $f^* = f_{min}$ i $x^* = arg f_{min}$

$n = n + 1$

Kraj

Ako je ispunjen kriterijum zaustavljanja, uzeti x^* za aproksimaciju opšteg rešenja i staje se za izvršavanjem

Genetske algoritme je jednostavno implementirati, ali njihovo ponašanje je teško razumeti. Specifično, teško je razumeti zašto ovi algoritmi često uspevaju u generisanju rešenja visoke finoće, tj. vrlo bliska optimalnom rešenju. Neka od ograničenja genetskih algoritama su nemogućnost operisanja na dinamički promenljivim podacima, loše skaliranje pri velikom broju dopustivih rešenja, i nemogućnost rada sa problemima gde funkcija evaluacije uzima mali broj različitih vrednosti, posebno ako su jedine vrednosti „da" i „ne".

Genetski algoritmi su uspešno primenjeni na problem ranca [23], raspoređivanje poslova po mašinama, raspored časova i druge probleme, o čemu se više može naći u radu [24].

3.1.2. Optimizacija rojem čestica

Algoritam *optimizacije rojem čestica* (Particle Swarm Optimization) je prvi put objavljen u [32, 53]. Originalno je bila namenjen simulaciji socijalnog ponašanja [31]. Algoritam radi nad populacijom (rojem) potencijalnih rešenja (čestica). Nova populacija rešenja se određuje tako što se svaka čestica iz prethodne populacije „pomeri". Kretanje čestica je određeno kao $x_i^{n+1} = x_i^n + v_i^n$, gde je v_i^n brzina kretanja čestice *i* u koraku *n*. Brzina čestice se menja tokom optimizacije prema formuli $v_i^{n+1} = \omega v_i^n + \psi(p_i^n - x_i^n) + \varphi(s^n - x_i^n)$, gde su $\omega, \psi\ i\ \varphi$ parametri optimizacije, a $p_i^n\ i\ s^n$ do sada najbolja poznata rešenja čestice *i* i celog

roja, respektivno. Pri računanju brzine čestice za sledeći korak, parametar ψ koliko će se brzina čestice promeniti tako da se čestica kreće ka najboljem rešenju čestice, dok parametar φ koliko će se brzina čestice promeniti tako da se kreće ka najboljem rešenju celog roja. Sa druge strane, parametar ω označava koliko brzine će čestica zadržati.

Odabir parametara može imati velikog uticaja na performanse algoritma. Parametri mogu biti određeni dinamički, koristeći takozvani koncept meta-optimizacije. Postoji nekoliko objašnjenja rada algoritma. Najrasprostranjenije je da velika vrednost parametra φ označava visok stepen „lokalnosti" pretrage, tj. algoritam ima veliku verovatnoću da zapadne u lokalni optimum, ali će se pretraga vrlo precizno približiti datom lokalnom optimumu, dok velika vrednost parametra ψ označava visok stepen „istraživanja", tj. verovatnoća da će algoritam da zapadne u lokalni optimum će biti mala, ali će u okolini konačnog rešenja verovatno postojati značajno bolje rešenje.

Teorijska i empirijska analiza optimizacije rojem čestica se može naći u [33]. Pregled uspešnih primena algoritma, kao što su planiranje razvoja nalazišta nafte i gasa, akustično poništavanje eha i dizajn adaptivnih filtera se mogu naći u [47, 48].

3.1.3. Optimizacija mravljim kolonijama

Algoritam *optimizacije mravljim kolonijama* (Ant Colony Optimization) je konstruisan po uzoru na ponašanje mrava i prvi put je predložen u [6, 13]. Prednost ovog algoritma u odnosu na ostale je sposobnost da radi sa podacima koji se dinamički menjaju tokom vremena [12].

Mravi se kreću slučajnim putanjama, i po nalaženju hrane se vraćaju u koloniju ispuštajući tragove feromona. Ako drugi mravi nađu takve tragove, onda verovatno neće nastaviti sa slučajnim kretanjem, već će nastaviti da prate taj trag, i ako nađu hranu pojačavaće trag feromona u povratku. Međutim, tokom vremena trag feromona isparava, i tako se smanjuje njegova atraktivnost mravima. Što je mravu potrebno više vremena da dosegne hranu i da se vrati nazad, feromoni imaju više vremena da ispare. Sa druge strane, mravi će češće prolaziti kroz kratke puteve, i time će gustina feromona biti veća na kratkim nego na dugim putevima. Dodatno, isparavanje feromona ima za prednost da omogućava algoritmu da zaobilazi konvergenciju ka lokalnom optimumu. Ako isparavanja uopšte ne bi bilo, prvi pronađeni put bi imao izrazitu atraktivnost i velik broj mrava bi pratio prvi pronađeni put.

Iz prethodno navedenog sledi da će mravi imati veću verovatnoću da prate kratak umesto dugačak put, i princip pozitivne sprege će eventualno dovesti do toga da svi mravi prate jedan put. Algoritam 3 je uopšteni algoritam optimizacije mravljim kolonijama.

Algoritam 3. Algoritam optimizacije mravljim kolonijama

Inicijalizacija

Definisati prostor kodiranih rešenja $\bar{X}$ i funkciju vrednosti rešenja $f(x)$

Izabrati početnu populaciju $P_1 = \{x_1^1, x_2^1, \dots, x_N^1\} \subseteq \bar{X}$

$f^* = min\{f(x_i^1),\ i = 1, \dots, N\},\ x^* = arg f^*$

Odrediti tragove feromona za svaku tačku $x_i^1,\ i = 1, \dots, N$ na osnovu funkcije $f(x)$

$n = 1$

Iterativni korak

Generisati $P_{n+1} = \{x_1^{n+1}, \dots, x_N^{n+1}\} \subseteq \bar{X}$. Svaka tačka $x_i^{n+1},\ i = 1, \dots, N$ se određuje probabilistički, na osnovu tačke x_i^n i tragova feromona u nekoj njenoj okolini

Ažurirati tragove feromona. Generisati nove i smanjiti intezitet postojećih tragova feromona (isparavanje)

$f_{min} = \min\{f(x_i^{n+1}),\ i = 1, \dots, N\}$

Ako $f_{min} < f^*$, tada $f^* = f_{min}$ i $x^* = arg f_{min}$

$n = n + 1$

Kraj

Ako je ispunjen kriterijum zaustavljanja, uzeti x^* za aproksimaciju opšteg rešenja i staje se za izvršavanjem

Optimizacija mravljim kolonijama je uspešno primenjena na velik broj problema kombinatorne optimizacije, na primer problem trgovačkog putnika [14], problem ranca [18], pronalaženje najkraćeg puta između dve tačke u grafu [13] i druge.

3.1.4. Optimizacija rojem pčela

Algoritam *optimizacije rojem pčela* (Artificial Bee Colony) je inspirisan načinom na koje pčele sakupljaju polen [43, 46]. Prvi put je predložen u [45]. Algoritam se sastoji u inicijalizaciji tokom koje se potencijalna rešenja (pčele) slučajno odrede, i pet koraka pri svakoj od iteracija: *regrutacija, lokalna pretraga, sužavanje okoline, napuštanje lokacije* i *globalna pretraga.*

Korak regrutacije se sastoji u tome da se sva rešenja sortiraju, i rešenja (pčele) koja su bolja „regrutuju" druge jedinke u populaciji, koje se nazivaju *sakupljačima*. Bolja rešenja regrutuju više jedinki u populaciji. Detalji zavise od implementacije do implementacije. Regrutovane jedinke pretražuju okolinu za koju su regrutovane. Ukoliko je bolje rešenje nađeno, jedinka koja je pronašla novo najbolje rešenje postaje novi *izviđač*. Ukoliko bolje rešenje nije pronađeno, okolina koja se pretražuje se sužava. Ukoliko se posle dovoljno vremena bolje rešenje ne pronađe u datoj okolini, lokacija se napušta. U svakom trenutku, jedinke izviđači vrše globalnu pretragu. Algoritam 4 je uopšteni algoritam optimizacije rojem pčela.

Neke uspešne primene algoritma su strukturna optimizacija, klasifikacija snimaka magnetnom rezonancom, procena pozicije lica, i druge [43, 44].

Algoritam 4. Algoritam optimizacije rojem pčela

Inicijalizacija

Definisati prostor kodiranih rešenja $\bar{X}$ i funkciju vrednosti rešenja $f(x)$

Izabrati početnu populaciju $P_1 = \{x_1^1, x_2^1, \dots, x_N^1\} \subseteq \bar{X}$

$f^* = min\{f(x_i^1),\ i = 1, \dots, N\},\ x^* = arg f^*$

$n = 1$

Iterativni korak

Odrediti $R_1 = \{x_{i_1}^n, x_{i_2}^n, \dots, x_{i_M}^n\} \subseteq \bar{X}$ skup tačaka sa najvećim vrednostima funkcije rešenja i za svaku tačku skupa definisati neku okolinu (oblast)

Za svaku prethodno definisanu oblast, odrediti jedinke koje će vršiti lokalnu pretragu

Izvršiti lokalnu i globalnu pretragu. Jedinke koje su dodeljene oblastima definisanim skupom R_1 vrše lokalnu pretragu, dok ostale jedinke vrše globalnu pretragu

$f_{min} = \min\{f(x_i^{n+1}),\ i = 1, \dots, N\}$

Ako $f_{min} < f^*$, tada $f^* = f_{min}$ i $x^* = arg f_{min}$

$n = n + 1$

Kraj

Ako je ispunjen kriterijum zaustavljanja, uzeti x^* za aproksimaciju opšteg rešenja i staje se za izvršavanjem

3.2 Metode zasnovane na lokalnom pretraživanju

Metod lokalnog pretraživanja (Local Search Method) se često primenjuje i u neprekidnoj i u diskretnoj optimizaciji. Ovaj metod podrazumeva definisanje neke strukture okolina (topologije) u prostoru dopustivih rešenja *X* na sledeći način: svakom $x \in X$ se pridružuje neki podskup $N(x) \subseteq X,\ x \notin N(x)$, koji se naziva okolina od *x*, a njeni članovi su susedi od *x*. U lokalnom pretraživanju se polazi od proizvoljne tačke iz *X* kao od početnog rešenja, pa su u svakoj iteraciji pretražuje okolina trenutnog rešenja i u njoj nalazi, prema nekom definisanom pravilu, sused koji predstavlja sledeće rešenje. Ako se takav sused može naći, pretraživanje staje i za aproksimaciju optimalnog rešenja uzima se ono među generisanim rešenjima za koje je vrednost funkcije cilja *f(x)* najmanja. Opšti oblik metode koja se bazira na principu lokalnog pretraživanja se može zapisati kao algoritam:

Algoritam 5. Algoritam metoda zasnovanih na lokalnom pretraživanju

Inicijalizacija

Izabrati početno rešenje $x_1 \in X, x^* = x_1, f^* = f(x_1)$

Iterativni korak, n = 1, 2, …

U okolini $N(x_n)$ trenutnog rešenja x_n naći sledeće rešenje x_{n+1}, prema nekom kriterijumu izbora

Ako je $f(x_{n+1}) < f^*$, tada $x^* = x_{n+1}\ i\ f^* = f(x_{n+1})$

Kraj

Ako kriterijum izbora nije zadovoljen ni za jednog suseda iz okoline $N(x_n)$ ili je zadovoljen neki drugi kriterijum zaustavljanja, izvršavanje se prekida

x^* se uzima za aproksimaciju optimalnog rešenja

Efikasnost jedne konkretne metode lokalnog pretraživanja, tj. stepen njene šanse da u razumnom vremenu dobije rešenje dovoljno blisko nekom optimalnom rešenju, zavisi pre svega od strukture okolina koje se koriste, kao i od kriterijuma za izbor sledećeg rešenja u svakom koraku metode.

U kontekstu lokalnog pretraživanja, okolina *N(x)* nekog dopustivog rešenja *x* se može u opštem slučaju definisati kao skup svih onih elemenata *y* iz *X* koji mogu biti dobijeni direktno iz *x* primenom neke, unapred definisane, modifikacije nazvane *pomak* $m(x, y)$ iz *x* u *y*. Na primer, ako je *X* skup binarnih vektora, tada se okolina *N(x)* vektora $x \in X$ može odrediti kao skup svih rešenja iz *X* koja se dobijaju *x* promenom neke njegove koordinate od 0 na 1 ili obrnuto. Drugim rečima, *N(x)* je kugla poluprečnika 1 u metrici indukovanoj Hamingovim rastojanjem [27].

Da bi neka struktura okolina obezbedila efikasnost lokalnog pretraživanja intuitivno je jasno da, pri njenom definisanju, treba težiti da budu zadovoljeni sledeći opšti uslovi:

1. okolina svake tačke iz *X* treba da bude simetrična, tj. $y \in N(x)$ ako i samo ako $x \in N(x)$
2. struktura okolina treba da zadovoljava uslov doseživosti, tj. da se, polazeći od bilo koje tačke prostora *X* može nizom uzastopnih pomaka doći do bilo koje druge tačke ovog prostora
3. pomak treda da omogući što jednostavnije i brže generisanje susednih rešenja, tj. preporučljivo je da ova modifikacija bude polinomske složenosti
4. okolina ne treba da bude ni suviše velika ni suviše mala, da bi se, s jedne strane, mogla lakše odgovarajuće pretražiti, a s druge strane da bi pružila dovoljno mogućnosti da se nađe sledeća tačka pretraživanja

Kriterijumi izbora sledeće tačke x_{n+1} u lokalnom pretraživanju mogu se definisati na različite načine. Tako se kod klasičnih metoda spuštanja dozvoljavaju pomace samo u susede koji ne pogoršavaju funkciju cilja, tj. x_{n+1} se bira tako da $f(x_{n+1}) \leq f(x_n)$. Specijalno, kod dobro poznate metode najstrmijeg spuštanja [9] x_{n+1} treba da zadovoljava uslove $f(x_{n+1}) \leq f(x_n)$ i $f(x_{n+1}) \leq f(x)$ za svako $x \in N(x_n)$. Mađutim, glavni nedostatak pri primeni metoda spuštanja na problem nalaženja globalnih minimuma je što se često u procesu pretraživanja ne mogu izbeći zamke lokalnih minimum. Naime, pretraživanje se može „zaglaviti" u okolini nekog lokalnog minimuma, iz koga se ne može izvući korišćenjem samo „silazećih" pomaka. Pokušaji da se ovaj nedostatak prevaziđe tako što će se neka metoda spuštanja primeniti više puta na jedan isti problem, i to za različita slučajno generisana početana rešenja, u većini slučajeva se nisu pokazali zadovoljavajući.

3.2.1. Simulirano kaljenje

Simulirano kaljenje (Simulated Annealing) je opšta heuristika koja kombinuje princip determinističke metode spuštanja sa probabilističkim Monte Carlo principom. Naime, u svakoj iteraciji ova heuristika generiše ba slučajan način nekog suseda iz okoline trenutne tačke, prihvatajući ga kao sledeću tačku pretraživanja, ne samo u slučaju poboljšanja, nego i u slučaju pogoršanja funkcije cilja, ali sa izvesnom verotanoćom koja se kontrolisano menja tokom iteracija. Na taj način se mogu izbeći zamke lokalnih minimuma i tako povećati mogućnost dobijanja kvalitetnih rešenja. Metod je nezavisno predložen u [35] i [4]. Algoritam 6 predstavlja opšti oblik algoritma simuliranog kaljenja.

Algoritam 6. Algoritam simuliranog kaljenja

Inicijalizacija
Izabrati početno rešenje $x_1 \in X$, $x^* = x_1$, $f^* = f(x_1)$
Iterativni korak, za n=1,2, …
Naći na slučajan način *x* u okolini $N(x_n)$ trenutnog rešenja x_n
Ako $f(x) \leq f(x_n)$, tada $x_{n+1} = x$
Ako $f(x) < f^*$, tada $x^* = x$ i $f^* = f(x)$
Ako $f(x) > f(x_n)$, izabrati slučajan broj *p* uniforman na *[0, 1]*
Ako $p \leq p_n$, tada $x_{n+1} = x$
Ako $p > p_n$, tada $x_{n+1} = x_n$
Kraj
Ako je zadovoljen kriterijumj zaustavljanja, izvršavanje se prekida, a x^* se uzima za aproksimaciju optimalnog rešenja

Vrednost p_n predstavlja verovatnoću prihvatanja pogoršanja funkcije cilja u iteraciji *n*. U osnovnoj verziji ova verovatnoća je jednaka, analogno procesu kaljenja:

$$p_n = e^{\frac{f(x_{n+1}) - f(x_n)}{t_n}}$$

gde je t_n vrednost takozvane temperature u iteraciji *n*. Temperatura je pozitivan kontrolni parametar, čije su vrednosti zadate nizom $t_1, t_2, \ldots$, takvim da je

$$t_1 \geq t_2 \geq \cdots, \qquad \lim_{n \to \infty} t_n = 0$$

Ovakav niz se naziva šema hlađenja i ona definiše način i brzinu smanjivanja temperature tokom iteracija.

Metod je uspešno primenjen na brojne probleme optimizacije, na primer za rešavanje problema trgovačkog putnika [55], maksimalne klike, raspoređivanja vozila po rutama i druge [15].

3.2.2. Tabu pretraga

Osnovni koncept *tabu pretraživanja* (Tabu Search) kao heurističke metodologije za rešavanje problema optimizacije predloženo je u radovima [19, 20, 22]. Tabu pretraga se bazira na principu lokalne pretrage i koristi, kao svoju najvažniju komponentu, takozvanu adaptivnu memoriju.

Adaptivna memorija predstavlja podatke o prethodnim fazama procesa pretraživanja koji utiču na izbor sledećih tačaka u ovom procesu. U osnovnoj verziji algoritam koristi samo jedan tip memorije, takozvanu kratkoročnu memoriju. Adaptivna memorija pamti samo *p* prethodni koraka pretrage, koji predstavljaju tabu – dalja pretraga se neće vršiti u tim tačkama. Drugi tipovi memorije koji se često upotrebljavaju su srednjeročna memorija, koja služi da bi se pretraga koncentrisala na regije koje imaju veliku verovatnoću da poseduju dobro rešenje, i dugoročna memorija, koja služi da bi se pretraga diverzifikovala. Algoritam 7 predstavlja opšti oblik algoritma tabu pretrage.

Algoritam 7. Algoritam tabu pretrage

Inicijalizacija
Izabrati početno rešenje $x_1 \in X$, $x^* = x_1$, $f^* = f(x_1)$
Istorija *H* je prazna
Iterativni korak, za n=1,2, …
Definisati modifikovanu okolinu $O(x_n, H)$ i funkciju vrednosti rešenja $f(x, H)$
Generisati podskup $O'(x_n) \subseteq O(x_n, H)$
Odrediti x_{n+1} minimizacijom $f(x, H)$ na $O'(x_n)$
Ako $f(x_{n+1}) < f^*$, tada $x^* = x_{n+1}$ i $f^* = f(x_{n+1})$
Ažurirati istoriju *H*
Kraj
Ako je zadovoljen kriterijumj zaustavljanja, izvršavanje se prekida, a x^* se uzima za aproksimaciju optimalnog rešenja

Opšti koncept tabu pretrage je izuzetno širok i omogućava različite strategije memorisanja sa različitim ciljevima. Više o dugoročnoj memoriji se može pronaći u [49]. Neki primeri uspešne primene tabu pretrage su problem trgovačkog putnika i spajanje puteva [3].

3.2.3. Iterativna lokalna pretraga

Iterativna lokalna pretraga (Iterated Local Search) se zasniva na ponavljanju obične lokalne pretrage, a prvi put je opisana u [39]. Lokalna pretraga može dovesti do rešenja koje je lokalno, ali ne i globalno optimalno. Takva rešenja ujedno postaju i konačna rešenja metode, ukoliko se ne uvede neka strategija da se pretraga nastavi izvan okoline trenutnog lokalnog

minimuma. Jedan od načina da se pretraga nastavi je da se lokalna pretraga ponovi više puta.

Najjednostavniji način da se lokalna pretraga ponovi više puta jeste da se svaki put započne od slučajno odabranog rešenja. Međutim, kada se bira početno rešenje za novu iteraciju lokalne pretrage, moguće je iskoristiti informacije dobijene tokom prethodnih iteracija. Ideja je da se na osnovu prethodnih iteracija, za sledeću iteraciju lokalne pretrage odabere početno rešenje koje je već u startu blizu lokalnog (globalnog) optimuma. Algoritam 8 predstavlja uopšteni algoritam iterativne lokalne pretrage.

Algoritam 8. Algoritam iterativne lokalne pretrage

Inicijalizacija
Izabrati početno rešenje $x_1 \in X$, $x^* = x_1$, $f^* = f(x_1)$
Istorija *H* je prazna
Iterativni korak, za n=1,2, …
Odrediti x_{n+1}^{start} na osnovu *H* i x_n
Odrediti x_{n+1} lokalnom pretragom počev od tačke x_{n+1}^{start}
Ažurirati istoriju *H*
Kraj
Ako je zadovoljen kriterijumj zaustavljanja, izvršavanje se prekida, a x^* se uzima za aproksimaciju optimalnog rešenja

Takav algoritam, koji se sastoji od višestrukog ponavljanja lokalne pretrage, pri čemu se svaka sledeća pretraga započinje od rešenja čiji se izbor bazira na osnovu prethodnih rešenja se naziva iterativna lokalna pretraga. Detalji implementacije zavise od problema do problema. Algoritam je uspešno primenjen na problem sekvenciranja poslova [38], problem protoka kroz prodavnice [29], kao i problem kretanja vozila [42]. Više se može pronaći u [40].

Metoda promenljivih okolina, koja je korišćena u ovom radu će biti opisana detaljnije u narednoj glavi.

4. METODA PROMENLJIVIH OKOLINA

Metaheuristika *metoda promenljivih okolina* (Variable Neighborhood Search, VNS) se zasniva na višestrukom ponavljanju lokalne pretrage. Struktura okoline se menja pri svakom uzastopnom ponavljanju pretrage, čime se pokušava izbeći lokalni optimum bilo koje od pojedinačnih okolina. Ukoliko pretraga pri nekoj okolini dosegne lokalni optimum, definiše se naredna okolina i pretraga se nastavlja. Metoda je predložena 1995. godine, od strane Mladenovića i Hensena [25].

Prilikom pretraživanja dopustivog prostora rešenja, primećene su tri činjenice:

1. Lokalni optimum u odnosu na jednu okolinu ne mora biti lokalni optimum u odnosu na neku drugu okolinu.
2. Globalni optimum je lokalni optimum u odnosu na bilo koju okolinu.
3. Lokalni optimumi u odnosu na različite okoline su međusobno bliski.

Pretraga po više okolina ima smisla uzimajući u obzir činjenice (1) i (2). Ove dve činjenice se lako dokazuju razmatrajući proizvoljnu neprekidnu funkciju nad jednom promenljivom. Ako je funkcija lokalni optimum u odnosu na sve definisane okoline, ona ne mora ujedno biti i globalni optimum. Činjenica (2) isključivo ukazuje na to da ako rešenje nije lokalni optimum u nekoj tački, onda nije ni globalni optimum. Činjenica (3) je empirijska, i na osnovu nje se može zaključiti da se pretragom okoline lokalnog optimuma, promenom definicije okoline može očekivati poboljšanje rešenja. Metoda promenljivih okolina se zasniva na navedene 3 činjenice. Postoje 3 glavne varijacije algoritma: *deterministički, stohastički* i *kombinovani*.

4.1 Metoda promenljivog spusta

Metoda promenljivog spusta (Variable Neighborhood Descent, VND) je varijacija metode promenljivih okolina prilikom koje se koriste isključivo determinističke metode. Neka je $O = \{O_1, O_2, \dots, O_n\}$ skup odabranih okolina. Za svaku od odabranih okolina se sekvencijalno vrši lokalna pretraga.

Počev od okoline O_1, metoda promenljivog spusta vrši lokalnu pretragu sve dok se ne dosegne lokalni optimum za tu okolinu. Počev od lokalnog optimuma za okolinu O_1, vrši se pretraga za okolinu O_2, zatim za okolinu O_3, i tako dalje - pretraga se nastavlja do okoline $O_{k_{max}}$ gde je $k_{max} \leq n$, kada se pretraga ponovo vrši za okolinu O_1. Algoritam se zaustavlja kada se nađe rešenje koje je lokalni optimum za sve okoline $\{O_1, O_2, \dots, O_n\}$.

Ukoliko postoji samo jedna okolina, tj. za *n = 1*, metoda promenljivog spusta je ekvivalentna klasičnoj lokalnoj pretrazi. Algoritam 9 predstavlja metodu promenljivog spusta.

Algoritam 9. Algoritam metode promenljivog spusta [25]

Funkcija *VND*(x, k_{max}):
Repeat
 $k \leftarrow 1$
 Repeat
  $x' \leftarrow arg\ \min_{y \in O_k\ (x)} f(x)$ // Pronaći najbolje rešenje u okolini trenutnog kandidata rešenja
  $PromeniOkolinu(x, x', k)$ // $x \leftarrow x'$, $k \leftarrow k + 1$
 Until $k = k_{max}$
Until dalja poboljšanja nisu moguća

4.2 Redukovana metoda promenljivih okolina

Redukovana metoda promenljivih okolina (Reduced Variable Neighborhood Search, RVNS) je varijacija metode promenljivog spusta prilikom koje se koriste stohastičke metode. Neka je $O = \{O_1, O_2, \dots, O_n\}$ skup odabranih okolina. Za svaku od odabranih okolina se redom vrši stohastička pretraga: za trenutno rešenje i posmatranu okolinu O_k, slučajno se bira jedna tačka, i ukoliko novoizabrana tačka daje bolje rešenje, pretraga se nastavlja od te tačke.

Prilikom redukovane metode promenljivih okolina, pretraga se započinje od okoline O_1 i neke početne tačke *x*, i bira se jedno rešenje *x'* iz skupa $O_1(x)$. Ukoliko *x'* predstavlja poboljšano rešenje u odnosu na rešenje *x*, rešenje *x'* postaje tekuće rešenje i pretraga se nastavlja od tog rešenja. U suprotnom se prelazi na okolinu O_2, zatim na okolinu O_3 i, slično metodi promenljivog spusta, sa okoline $O_{k_{max}}$ se prelazi na okolinu O_1. Algoritam se može zaustaviti nakon unapred definisanog vremena t_{max} ili nakon što određen broj iteracija I_{max} ne dovede do poboljšanja rezultata. Redukovana metoda promenljivih okolina se može predstaviti sledećim algoritmom:

Algoritam 10. Algoritam redukovane metode promenljivih okolina [25]

Funkcija *RVNS*($x, k_{max}, t_{max}, I_{max}$)
Repeat
 $k \leftarrow 1$
 Repeat
  $x' \leftarrow Razmrdaj(x, k)$ // primeniti funkciju razmrdavanja
  $PromeniOkolinu(x, x', k)$ // $x \leftarrow x'$, $k \leftarrow k + 1$
 Until $k = k_{max}$
 $t \leftarrow VremeIzvršavanja()$
Until $t > t_{max}$ ili poboljšanja nisu u pronađenja u I_{max} iteracija

Redukovana metoda promenljivih okolina je korisna za dobijanje dobrog početnog rešenja, ili kod optimizacionih problema velikih dimenzija. Okoline se trebaju birati tako da svaka sledeća okolina bude veće kardinalnosti od prethodne.

4.3 Osnovna metoda promenljivih okolina

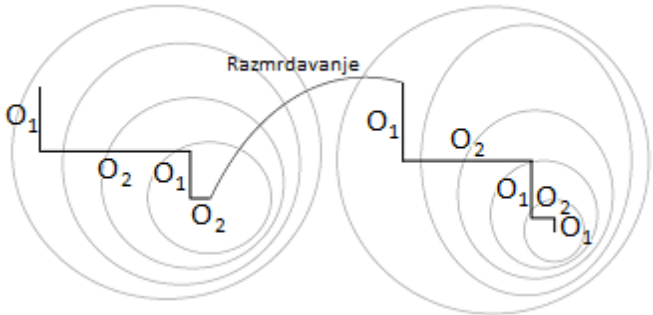
Osnovna metoda promenljivih okolina (Basic Variable Neighborhood Search, BVNS) kombinuje prethodne dve metode. Pri ovom pristupu algoritam vrši lokalnu pretragu kroz više različitih okolina (slično metodi promenljivog spusta). Kada se pretraga izvrši za sve okoline, vrši se korak razmrdavanja.

Neka je $O = \{O_1, O_2, \dots, O_n\}$ skup odabranih okolina. Pri Osnovnoj metodi promenljivih okolina, u prvom koraku se vrši lokalna pretraga u odnosu na okolinu O_1, zatim okolinu O_2, sve do okoline $O_{k_{max}}$. Pre nego što se pređe na okolinu O_1, vrši se korak razmrdavanja (eng. *Shake*)Ovaj korak je stohastički da bi se izbegla ciklična pretraga. Često su okoline $\{O_1, O_2, \dots, O_n\}$ ugnježdene, i cilj koraka razmrdavanja je da se izbegne lokalni optimum. Algoritam se može zaustaviti nakon unapred definisanog vremena t_{max} ili nakon što određen broj iteracija I_{max} ne dovede do poboljšanja rezultata. Osnovna metoda promenljivih okolina se može predstaviti algoritmom:

Algoritam 11. Algoritam osnovne metode promenljivih okolina [25]

Funkcija *VNS*$(x, k_{max}, k'_{max}, t_{max}, I_{max})$
Repeat
 $k \leftarrow 1$
 Repeat
  $x' \leftarrow Razmrdaj(x, k)$
  $x'' \leftarrow VND(x', k'_{max})$
  $PromeniOkolinu(x, x', k)$ // $x \leftarrow x'$, $k \leftarrow k + 1$
 Repeat $k = k_{max}$
 $t \leftarrow VremeIzvršavanja()$
Until $t > t_{max}$ ili poboljšanja nisu u pronađenja u I_{max} iteracija

Na Slici 1 je predstavljen primer rada osnovne metode promenljivih okolina. Za neko rešenje r, $O_1(r)$ je skup svih rešenja koje imaju istu vrednost *X* koordinate, dok je $O_2(r)$ skup svih rešenja koje imaju istu vrednost *Y* koordinate. Na slici, O_1 i O_2 predstavljaju korake pretrage u odgovarajućim okolinama.



Slika 1. Primer rada osnovne metode promenljivih okoline

5. FORMULACIJA PROBLEMA

Rad hitne pomoći sadrži niz specifičnosti koje se moraju uzeti u obzir prilikom definisanja matematičkog modela.

5.1 Planiranje i rad hitne pomoći

Hitna pomoć se bavi pružanjem medicinskih usluga pacijentima sa urgentnim potrebama. Kada poziv stigne u pozivni centar, ambulantna vozila se zadužuju za poziv. Verovatnoća preživljavanja pacijenta je direktno zavisna od vremena koje je potrebno kolima hitne pomoći da stignu na mesto gde je neophodno ukazati pomoć [36]. Zato je od izuzetne važnosti da vreme od trenutka poziva do prihvatanja pacijenta bude što kraće. U beogradskoj hitnoj pomoći, za najurgentnije slučajeve postavljen je zahtev da od trenutka poziva do dolaska na mesto poziva vozila hitne pomoći stignu za najviše 15 minuta.

Da bi se smanjilo vreme od poziva do dolaska vozila na mesto poziva, vozila hitne pomoći su raspoređena po baznim stanicama po gradu. U Beogradskoj hitnoj pomoći, svaka stanica ima jedna ili više vozila, i zadužena je za zasebnu opštinu. U specijalnim slučajevima, vozila mogu biti poslata do dela grada za koja nisu zadužena.

Kada se vozila pošalju na poziv, ona više nisu slobodna za druge pozive. Ovo dovodi do smanjene pokrivenosti. Jedan način da se poboljša pokrivenost jeste da se poveća broj vozila po stanicama. Međutim, takvo rešenje nije praktično primenljivo, jer većina službi hitne pomoći ima ograničen budžet. Druga opcija je da se uvede očekivano pokrivanje, koje se računa uzimajući u obzir očekivan broj poziva, brzinu kretanja vozila, broj slobodnih vozila i očekivani procenat vremena tokom kojeg su vozila slobodna.

Ovakva postavka generiše graf, sa ivicama od baznih stanica do svih delova grada do kojih vozila mogu stići za manje od zadatog vremena. Cilj ovog rada je da, koristeći metaheurističke metode kombinatorne optimizacije, minimizuje vreme izračunavanja optimalnog raspodela vozila hitne pomoći u različitim periodima dana tako da se maksimizuje očekivana pokrivenost uzimajući u obzir gore navedene parametre.

5.2 Dosadašnji rezultati

Među prvim modelima za pozicioniranje baznih stanica hitne pomoći je lokacijski problem skupova koji se razmatra u radu [56]. Ovaj model minimizuje potreban broj ambulantnih vozila sa ciljem da se pokrije svaka tačka sa koje se očekuje poziv sa najmanje jednim

vozilom. U radu [5] se koristi model maksimalnog pokrivanja lokacija. Ovaj model maksimizuje broj pokrivenih tačaka pri ograničenom broju ambulantnih vozila. Oba ova modela ignorišu činjenicu da su vozila nedostupna tokom intervencije. U radu [21] se koristi dupli standardni model, koji maksimizuje broj tačaka pokrivenih sa najmanje dva vozila. Međutim, ni pri ovom modelu ne postoji garancija da će postojati dostupno vozilo pri svakom od poziva. Da bi se prevazišao ovaj problem, u radu [11] se koristi probabilistički model maksimalnog očekivanog pokrivanja lokacija. Ovaj model ne uzima u obzir da su pri različitim vremenskim intervalima parametri modela (pre svega očekivani broj poziva i vreme putovanja) različiti. Ova činjenica je prvi put iskorišćena u radu [58], u kome se koristi model iznet u prethodnoj glavi. Autori rada koriste metod linearnog programiranja, koji ne omogućava pronalaženje rešenja za velike instance. U radu [28] se koristi heuristički algoritam koji se može skalirati na veliki broj ambulantnih vozila, ali ne i na veliki broj baznih stanica. U radu [34] se predlaže iterativni heuristički algoritam koji uzima u obzir dinamičnost sistema, ali se ne omogućava rad algoritma na velikim instancama. Do sada u literaturi za rešavanje ovog problema nije korišćen metod promenljivih okolina.

5.3 Matematička formulacija problema

U ovom radu se koristi matematički model predložen u radu [58]. Neka je q verovatnoća da su neka ambulantna vozila zauzeta u bilo kom trenutku vremena. Vrednost q je jednaka ukupnom vremenu koja sva vozila provedu na zadatku tokom jednog vremenskog intervala, podeljeno sa ukupnim trajanjem intervala.

Definicija 11. Očekivana pokrivenost neke tačke je očekivana vrednost broja poziva sa te tačke na koje služba hitne pomoći može reagovati. Marginalna pokrivenost tačke je očekivana pokrivenost tačke pri nekom ograničenju. Marginalna pokrivenost tačke k-tim vozilom je očekivana pokrivenost tačke ako se posmatra samo k-to vozilo.

Ako se pretpostavi da do neke tačke u gradu može doći najviše k vozila hitne pomoći u zadatom vremenu (15 minuta za Beograd), tada je očekivano pokrivanje date tačke jednako

$$E_k = 1 - q^k$$

Marginalna pokrivenost date tačke k-tim vozilom je jednaka

$$E_k - E_{k-1} = (1 - q^k) - (1 - q^{k-1}) = (q^{k-1} - q^k) = (1 - q)q^{k-1}$$

Neka su dati sledeći ulazni podaci:

T – skup svih vremenskih intervala unutar jednog dana $1, \dots, t_{max}$

V – skup svih regiona do kojih kola moraju doći

W – skup svih potencijalnih baznih lokacija za kola hitne pomoći

W_{it} – skup svih potencijalnih baznih lokacija za vozila hitne pomoći koje mogu reagovati na pozive koji dolaze iz regiona $i \in V$ tokom intervala $t \in T$ u zadatom roku

P – maksimalan broj kola hitne pomoći

d_{it} – očekivan broj poziva iz regiona $i \in V$ u vremenskom intervalu $t \in T$

β – parametar koji predstavlja penal za svaku novu iskorišćenu stanicu. U praksi su službe hitne pomoći ograničene budžetom i korisno je upotrebiti što manjui broj stanica

γ – slično parametru β, ovaj parametar predstavlja penal za svako premeštanje vozila između dve stanice

Na osnovu prethodnih ulaznih podataka, cilj je odrediti sledeće promenljive:

x_{jt} – broj vozila lociran na poziciji *j* tokom intervala t

y_{ikt} – je jednako 1 akko je tačka i pokrivena sa najmanje k vozila tokom intervala t, inače y_{ikt} je 0

z_j – je jednako 1 akko je bazna lokacija korišćena najmanje jednom, inače z_j je jednako 0

r_{ijt} – je jednako broju vozila koja su premeštena iz stanice i u stanicu j između vremenskih intervala t i $t+1$

Tada je unkcija koju treba optimizovati:

$$\sum_{t\in T}\sum_{i\in V}\sum_{k=1}^{p_t} d_{it}(1-q_t)q_t^{k-1}y_{ikt} - \beta\sum_{j\in W} z_j - \gamma\sum_{i\in W}\sum_{j\in W}\sum_{t\in T} r_{ijt}$$

Prvi član predstavlja očekivanu pokrivenost u odnosu na sve regione tokom svih vremenskih intervala. Traži se da se maksimizuje očekivani broj poziva na koji vozila mogu odgovoriti, zbog čega se očekivani broj poziva množi sa marginalnom verovatnoćom da će vozila odgovoriti na poziv. Drugi član se odnosi na kaznu za svaku iskorišćenu baznu stanicu, dok je treći član kazna za svako premeštanje vozila između dve stanice. Drugi i treći član imaju negativni predznak jer predstavljaju kazneni deo funkcije cilja. Skup ograničenja modela je sledeći:

$$\sum_{j\in W_{it}} \geq \sum_{k=1}^{p_t} y_{ikt}\,, \qquad i \in V, t \in T \tag{1}$$

$$\sum_{j\in W} x_{jt} \leq p_t\,, \qquad t \in T \tag{2}$$

$$\sum_{t\in T} x_{jt} \leq M z_j\,, \qquad j \in W \tag{3}$$

$$x_{jt} + \sum_{j \in W} r_{ijt} - \sum_{j \in W} r_{ijt} = x_{j(t+1)}, \qquad t \in T \backslash t_{max}, j \in W \tag{4}$$

$$x_{jt_{max}} + \sum_{i \in W} r_{ijt_{max}} - \sum_{i \in W} r_{ijt_{max}} = x_{j1} \tag{5}$$

$$x_{jt}, r_{ijt} \in \mathbb{N} \tag{6}$$

$$y_{ikt}, z_j \in \{0,1\} \tag{7}$$

Ograničenje (1) osigurava da, u svakom vremenskom intervalu, iz baznih stanica do svakog regiona može stići najmanje *k* vozila, ukoliko *k* vozila može odgovoriti na pozive. Ograničenje (2) garantuje da neće biti korišćeno više of *p* vozila u bilo kom vremenskom intervalu. Ograničenje (3) garantuje da je broj aktivnih vozila u nekoj stanici za neki vremenski interval veći od nule, samo ako se ta stanica koristi. Ovo ograničenje koristi veliko-*M* ograničenje. Ograničenja (4) i (5) osiguravaju da r_{ijt} (broj vozila koja su premeštena iz stanice *i* u stanicu *j* između intervala *t* i *t+1*) odgovara vrednostima x_{jt} (broj vozila koja su stacionirana u stanici *j* tokom intervala *t*). Ograničenja (6) i (7) označavaju dozvoljene vrednosti za promenljive.

Ovaj model se može naći u [58] u kome je, za male instance, rešen algoritmom linearnog programiranja.

6. PRIMENA METODA OKOLINA NA REŠAVANJE PROBLEMA

Za rešavanje problema korišćena je osnovna metoda promenljivih okolina (Basic VNS) čije su osnovne karakteristike opisane u glavi 4 ovog rada. Za lokalnu pretragu, korišćeno je 6 ugnježdenih okolina. Korišćena su četiri različita koraka razmrdavanja (Shake), pri čemu svaki ima različitu verovatnoću primene. Parametri su podešeni tokom preliminarnih eksperimenata u cilju što boljih performansi algoritma. Interpretacija odabira tih parametara je data zajedno sa njihovom odabranom vrednošću.

6.1 Lokalna pretraga

Za lokalnu pretragu, korišćeno je $|T| = t_{max}$ ugnježdenih okolina. Za sve test instance, t_{max} = 6, da bi se održala konzistentnost sa radom [58]. Za datu raspodelu vozila po baznim stanicama *X*, tačka *X'* je u okolini $O_t(X)$, $1 \leq t \leq t_{max}$, ako i samo ako postoje stanice S_1 i S_2, takve da se premeštanjem jednog vozila iz stanice S_1 u stanicu S_2, počevši od raspodele *X* dobija raspodela *X'*, pri čemu se premeštanje vozila vrši za sve vremenske intervale iz skupa

$$T_{j,t} = \begin{cases} \{j, j+1, \dots, j+t-1\} & , \quad ako\ j+t-1 \leq t_{max} \\ \{j, j+1, \dots, t_{max}, 1, 2, \dots, t-t_{max}+j-1\} & , \quad inače \end{cases}$$

Prvo se vrši pretraga tako što se vozila premeštaju iz jedne stanice u drugu za ceo dan, zatim se premeštaju za $t_{max} - 1$ vremenskih intervala, itd.

Slika 2. Primeri lokalne pretrage za okoline *t* jednako od 1 do 3

Na Slici 2, za *t = 1*, vozilo se premešta iz stanice 1 u stanicu 2 za interval 3. Za *t = 2*, vozilo se premešta iz stanice 1 u stanice 2 za intervale 3 i 4. Slično, za *t = 3*, vozilo se premešta iz stanice 1 u stanicu 2 za intervale 3, 4 i 5.

6.2 Korak razmrdavanja

Korak razmrdavanja je u osnovi stohastički i koristi se radi izbegavanja lokalnih optimuma. Tokom preliminarnih eksperimenata, primećeno je da sva globalno optimalna rešenja imaju relativno mali broj baznih stanica. Preliminarni testovi su pokazali da je algoritam koji se sastoji isključivo od lokalne pretrage nalazio lokalno optimalna rešenja koja dobro raspoređuju vozila u različitim vremenskim intervalima, ali su odabrane bazne stanice u kojima su ta vozila bila stacionirana često bile pogrešne. Ovakva rešenja predstavljaju lokalne optimume koje treba izbeći koracima razmrdavanja. Specifično, ako bi se odabrali i fiksirali mali broj dozvoljenih baznih stanica, algoritam bi nalazio rešenje optimalno na smanjenom dopustivom skupu.

Može se primetiti da, kako se pretraga približava lokalnom optimumu, u izvesnim stanicama će biti stacionirano sve više, a u ostalim stanicama sve manje vozila. U Definicijama 12 i 13 se formalno definišu ove dve grupe stanica kao *popunjene* i *prazne*.

Definicija 12. *Popunjenost* bazne stanice je prosečan broj vozila hitne pomoći stacioniranih u toj stanici tokom celog dana. Računa se po formuli:

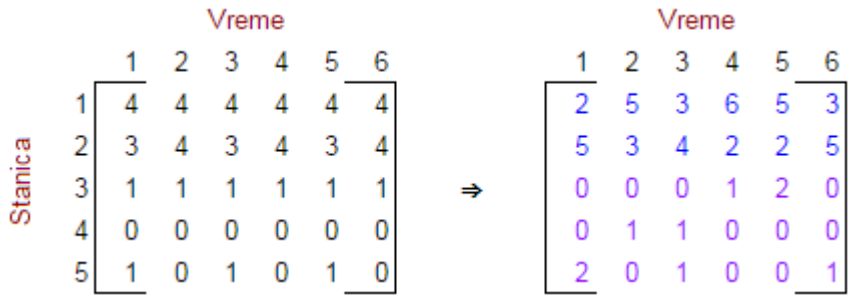
$$F(s) = \sum_{t=1}^{t_{max}} \frac{w_{st}}{t_{max}}$$

gde je w_{st} broj vozila hitne pomoći stacioniranih u baznoj stanici *s* tokom interval *t*, a t_{max} broj vremenskih interval na koje je dan podeljen.

Definicija 13. Neka je $F = (F_1, F_2, \dots, F_{|W|})$, i neka je $F' = (F'_1, F'_2, \dots, F'_{|W|})$, gde svaki element niza *F'* odgovara tačno jednom elementu niza *F*, i $F'_{i+1} \leq F'_i,\ i = 1, \dots, |W|$. Neka je *k* takvo da je $F'_{k+1} - F'_k = \max_{i=1..|W|-1}(F'_{i+1} - F'_i)$. Za stanicu S_i kaže se da je *prazna* ako je $i \leq k$, a u suprotnom, za datu stanicu se kaže da je *popunjena*.

6.2.1 Slučajno premeštanje

Slučajno premeštanje predstavlja premeštanje izvesnog broja vozila hitne pomoći iz jedne slučajno odabrane bazne stanice u drugu, za slučajno odabrani vremenski trenutak. Premeštanja se vrše isključivo iz popunjene u drugu popunjenu stanicu, i iz jedne prazne u drugu praznu stanicu. Odabir stanica iz koje, i u koje se vrši premeštanje vozila se vrši po uniformnoj raspodeli na skupu svih stanica koje omogućavaju premeštanja. Na ovaj način se obezbeđuje da broj praznih i popunjenih stanica ostaje konstantan.

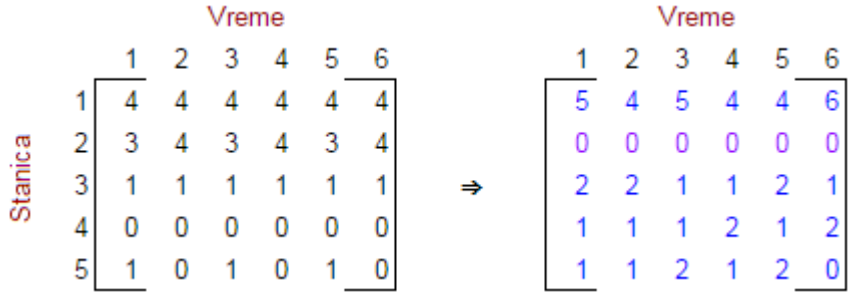


Slika 3. Primer slučajnog premeštanja

Na slici 3 se vidi primer slučajnog premeštanja. Mali broj vozila je premešten, i to tako da je ukupan broj vozila koje su stacionirane u praznim stanicama i ukupan broj vozila koje su stacionirane u popunjenim stanicama zadržan.

6.2.2 Uklanjanje popunjene stanice

Od svih baznih stanica koje su popunjene, bira se jedna i sva vozila se slučajno premeštaju u druge stanice. Cilj ovog koraka je da se izbegne lokalni optimum koji nije ujedno i globalni optimum. Ovaj korak smanjuje broj popunjenih, i povećava broj praznih stanica.



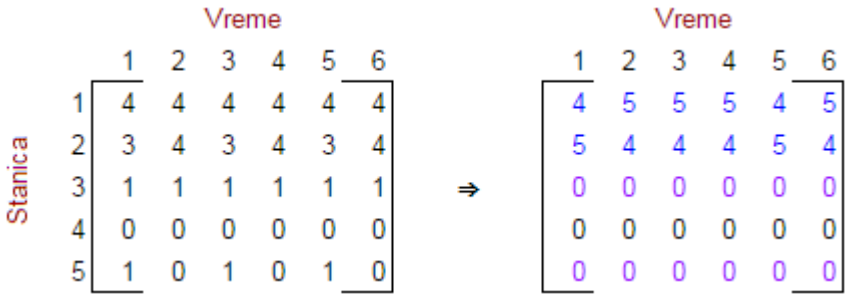
Slika 4. Primer uklanjanja popunjene stanice

Na slici 4 predstavljen je primer *uklanjanja popunjene stanice*. Sva vozila locirana u jednoj popunjenoj stanici se premeštaju u druge stanice slučajnom raspodelom.

6.2.3 Uklanjanje iz praznih stanica

Imajući u vidu kazneni parametar β, koji reflektuje ograničenja budžeta, u optimalnom rešenju samo ograničen broj baznih stanica sadrži vozila hitne pomoći, dok su ostale stanice u potpunosti prazne. Da bi se smanjilo vreme potrebno algoritmu da dosegne optimum, uvodi se korak razmrdavanja koji se sastoji u tome da sva vozila u svim stanicama koje su

prazne premeštamo u sve stanice koje su popunjene. Ovim korakom se povećava broj praznih stanica.

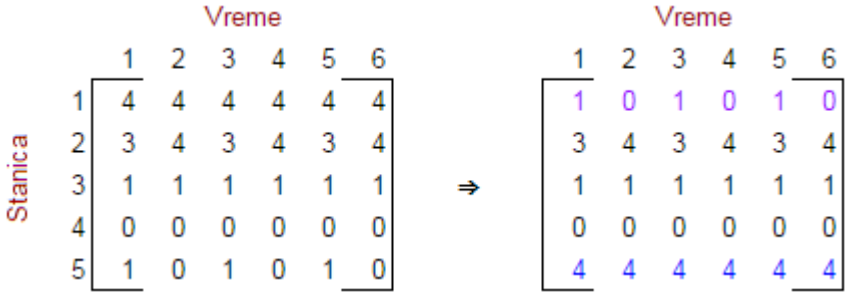


Slika 5. Primer uklanjanja iz praznih u popunjene stanice

Primer uklanjanja iz praznih stanica se može videti na Slici 5. Sva rešenja dobijena tokom eksperimenata imaju ovakav oblik. Sva vozila se nalaze u samo nekoliko stanica, ostale stanice su prazne.

6.2.4 Zamena prazne i popunjene stanice

Slično kao i korak „Premeštanje iz popunjene stanice", korak *zamena prazne i popunjene stanice* ima za cilj da izbegne lokalni minimum koji nije globalni. Razlika je u tome što se vozila iz slučajno odabrane popunjene stanice premeštaju u jednu praznu stanicu, umesto da se premeštaju u različite slučajno odabrane stanice. Ovaj korak zadržava broj praznih i popunjenih stanica. Za razliku od ostalih koraka razmrdavanja, ovaj korak se ne izvršava ukoliko promena ne dovodi do poboljšanja.



Slika 6. Primer zamene prazne i popunjene stanice

Slika 6 predstavlja primer zamene prazne i popunjene stanice. Za svaki vremenski interval, broj vozila u stanicama 1 i 6 je zamenjen.

6.3 Struktura primenjene osnovne metode promenljivih okolina

U svim primerima, dan je izdeljen na 6 jednakih vremenskih intervala koji generišu 6 različitih okolina koje se koriste za lokalnu pretragu definisanu u glavi 4.3. Početna rešenja su slučajno generisana. Algoritam prvo vrši pretragu unutar prve okoline trenutnog rešenja, zatim unutar druge okoline, sve do šeste, najgranularnije okoline. Svaki korak pretrage se obavlja tako što se slučajno odabere jedno rešenje iz date okoline, i ukoliko je novo rešenje bolje od trenutnog, trenutno rešenje se zamenjuje sa novim. Iz jedne okoline se prelazi u narednu okolinu ukoliko 10 uzastopnih provera ne da bolje rešenje, izuzev za najgranularniju okolinu, za koju se provera vrši sve do *10+c* uzastopnih neuspeha, gde je *c* jednako ukupnom broju ciklusa. Jedan ciklus se sastoji od svih uzastopnih koraka lokalne pretrage, i jednog koraka razmrdavanja. Korak razmrdavanja je bilo koji od koraka navedenih u glavama 6.2.1 – 6.2.4. Parametri koraka razmrdavanja su navedeni u Tabeli 1. Pseudokod primenjene osnovne metode promenljivih okolina prikazan je Algoritmom 7.

Tabela 1. Parametri pri izvršavanju koraka razmrdavanja

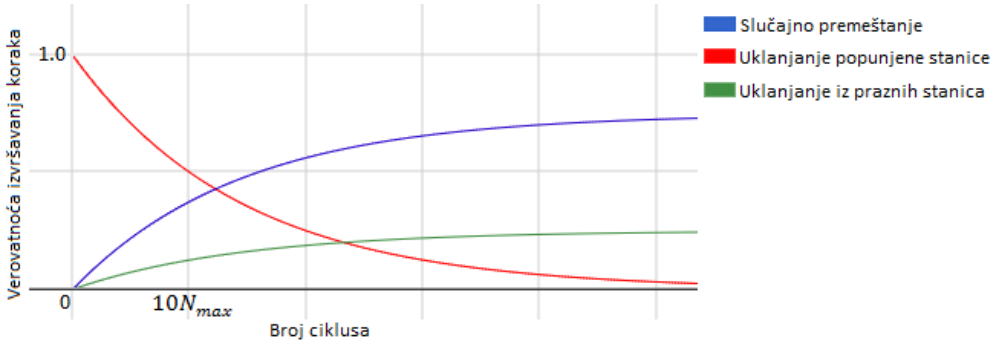
	Učestalost	**Parametri**
Zamena prazne i popunjene stanice	100%	Za svaku popunjenu stanicu, isprobava se jedna slučajno odabrana prazna stanica. Zamena se vrši samo ako je novo rešenje bolje.
Uklanjanje iz praznih stanica	$0.5^{\frac{n-1}{10N_{max}-1}}$	
Uklanjanje popunjene stanice	$\frac{1}{4}\left(1-0.5^{\frac{n-1}{10N_{max}-1}}\right)$	
Slučajno premeštanje	$\frac{3}{4}\left(1-0.5^{\frac{n-1}{10N_{max}-1}}\right)$	Ukupan broj premeštanja je jednak $1/10$ broja ambulantnih vozila.

U Tabeli 1 broj *n* označava u kom ciklusu se korak izvršava, a $10N_{max}$ broj ciklusa nakon kojeg pretraga staje ukoliko napredak nije napravljen, koji će upravo biti objašnjen.

Preliminarni eksperimenti su pokazali da ovakva postavka daje najbolje rezultate. Potpuna pretraga cele okoline zahteva previše vremena po koraku za velike primere. Eksperimentisano je sa varijantama algoritma gde se nakon 5, 10, 15 i 20 neuspešnih pokušaja prelazi na sledeće rešenje, i najbolji rezultati su postignuti za broj 10. Pri kraju pretrage, kada se trenutno rešenje razlikuje od optimalnog za svega nekoliko promena, algoritam povećava intezitet pretrage unutar okoline maksimalne kardinalnosti.

Pri svakom koraku razmrdavanja, zamena prazne i popunjene stanice se isprobava $|W|$ puta. Ukoliko nijedan od pokušaja ne da bolje rešenje, izvršava se jedan od ostalih tipova koraka razmrdavanja. Tip koraka koji će se obaviti se bira stohastički. Precizne formule za odabir tipa koraka su navedene u Tabeli 1. Verovatnoća da se obavi korak uklanjanje praznih stanica vremenom raste. U prvom ciklusu je 0, dok je u ciklusu *10X* jednaka 0.5, gde je *X* broj ciklusa

nakon kojeg pretraga staje ukoliko napredak nije napravljen. Obrnuto, uklanjanje praznih stanica ima verovatnoću 0 u prvom koraku i postepeno raste. Slično, verovatnoća slučajnog premeštanja je trostruko veća od verovatnoće uklanjanja praznih stanica, tako da verovatnoća odigravanja ova dva koraka konvergira ka $\frac{1}{4}$ i $\frac{3}{4}$. Korak zamene praznih i popunjenih stanica se izvršava samo ako daje bolje rešenje, te je koristan u svakom trenutku.



Slika 7. Grafik verovatnoća da se izvrši neki od koraka razmrdavanja

Na slici 7 se može videti da verovatnoća da se izvrši korak „Uklanjanje popunjene stanice" vremenom opada, dok se verovatnoća da se izvrše koraci „Slučajno premeštanje" ili „Uklanjanje iz praznih stanica" vremenom povećava. Pseudokod se može predstaviti kao:

Algoritam 12. Pseudokod primenjene osnovne metode promenljivih okolina

Funkcija *VNS*(x, t_{max}, I_{max})
$trenutniCiklus \leftarrow 0$
Repeat
 for t = 1 .. $\boldsymbol{t_{max}}$
  $O_t(x) \leftarrow OdaberiOkolinu(x, t)$
  for $\boldsymbol{i = 1..(t = t_{max}\ ?\ 10 + trenutniCiklus\ :\ 10}$
   $x' \leftarrow SlučajnoOdaberiElement(O_t(x))$
   if rešenje x' bolje od rešenja x **then**
    $x \leftarrow x'$
    $O_t(x) \leftarrow OdaberiOkolinu(x, t)$
   end then
  end for
 end for
 $trenutniCiklus \leftarrow trenutniCiklus + 1$
 $x \leftarrow Razmrdaj(trenutniCiklus)$
Until poboljšanja nisu pronađena u I_{max} iteracija

Uklanjanje iz popunjene stanice služi da bi se izbegao lokalni optimum koji nije globalni optimum. Ukoliko algoritam radi kao što je očekivano, što više ciklusa se izvrši, algoritam je bliži globalnom optimumu i udaljavanje od trenutno lokalnog optimuma je ujedno i udaljavanje od globalnog optimuma. U prvim koracima, ukoliko algoritam konvergira ka nekom lokalnom optimumu, verovatnoća da je taj optimum ujedno i globalni je mala.

Sa druge strane, pošto je poznat oblik svakog lokalnog optimuma, uklanjanje iz praznih stanica služi da bi se taj lokalni optimum brže dosegao, i taj korak ima više smisla što izvršavanje traje duže.

Slučajno premeštanje služi da bi se uvela izvesna slučajnost pri „kretanju“ trenutnog rešenja kroz prostor dopustivih rešenja. U slučaju determinističke varijante, algoritam će jednostavno konvergirati ka najbližem lokalnom optimumu. Ukoliko uvedemo izvesnu slučajnost u pretrazi, algoritam će paralelno konvergirati ka različitim lokalnim optimumima i vršiti globalnu pretragu. Pretpostavka, koja je empirijski potvrđena, je da u proizvoljnom trenutku, trenutno rešenje ima veću verovatnoću da konvergira ka boljem lokalnom optimumu nego ka lošijem. Kao posledica prethodno navedenog, može se očekivati da uvođenjem slučajnosti u pretrazi algoritam konvergira sporije, ali ka boljem lokalnom optimumu.

Poređenjem optimalnih rešenja dobijenih CPLEX 12.1 rešavačem, i međurezultatima heurističkog rešenja, utvrđeno je da je prosečan najduži niz ciklusa bez popravke rešenja, a da optimalno rešenje već nije dosegnuto, jednak $1.59W + 22.6$, gde je W broj baznih stanica. Nijedan od testova nije posedovao niz ciklusa bez popravke rešenja, a da optimalno rešenje već nije dosegnuto, veći od $1.59W + 52.4$. Na osnovu prethodnog, postavljeno je da se algoritam zaustavlja kada prođe $1.6W + 60$ ciklusa bez popravke najboljeg rešenja.

Radi bržeg izračunavanja, algoritam je optimizovan na nekoliko načina. Graf povezanosti baznih stanica i regiona je zapisan u nizu listi, umesto u matrici. Za sve vrednosti *k*, član

$$\sum_{t \in T} \sum_{i \in V} \sum_{k=1}^{p_t} d_{it}(1 - q_t) q_t^{k-1} y_{ikt}$$

u funkciji čiji maksimum se traži je unapred izračunat i zapisan u matrici *y'*. U isto vreme, kreira se matricu *y''* čiji je svaki element $y''_{ijt} = y'_{it} - y'_{jt}$, za svako $i, j \in V$. Dok je neke korake sa mnogo promena (uklanjanje popunjene stanice) efikasno izračunati svaki put, druge korake sa malo promena (svaki korak lokalne pretrage) je korisno izračunavati kao razliku u odnosu na prethodno rešenje, koristeći podatke smeštene u keš tabelu.

Algoritam je testiran na 2.1 Ghz Dual-Core procesoru sa 4 GB RAM-a, pri operativnom sistemu Windows 7. Heuristički algoritam je implementiran u C# jeziku. Da bi se iskoristila oba procesora, u svakom koraku lokalne pretrage su paralelno izračunavana 2 rešenja iz trenutne okoline. Ukoliko bi oba rešenja bila lošija od trenutnog rešenja, brojač uzastopnih neuspešnih pokušaja bi bio povećan za 2. Ukoliko bi najmanje jedno novo rešenje bilo bolje od trenutnog, najbolje novo rešenje bi bilo uzimano za trenutno rešenje i brojač uzastopnih neuspešnih pokušaja bi bio postavljen na nulu. U slučaju da je dozvoljen samo jedan korak, samo jedno novo rešenje bi bilo generisano na jednom procesoru, bez paralelizacije. Kao osnova za poređenje, implementiran je program u CPLEX 12.1 rešavaču, prema specifikacijama iz rada [58]. Ovaj program predstavlja deterministično rešenje.

7. ANALIZA EKSPERIMENTALNIH REZULTATA

7.1 Test instance

U ovom radu su korišćene jedna realna test instanca i slučajno generisane test instance. Realna test instanca je kreirana na osnovu podataka iz službe Hitne Pomoći Beograda. Podaci su dobijeni za period jun-jul 2015. godine, za najhitnije, „crvene“ slučajeve. Gradski zavod za hitnu medicinsku pomoć Beograda operiše na području od ukupno 11 opština. 22 ambulantna vozila su raspoređena po baznim stanicama na ukupno 10 opština. Dan je podeljen na 6 jednakih vremenskih intervala od 4 sata. Učestalost poziva i očekivano vreme putovanja tokom različitih perioda su izračunati na osnovu dobijenih podataka. Svaka opština predstavlja potencijalnu lokaciju za bazne stanice.

Za potrebe ocene predloženog algoritma za veće dimenzije problema, generisane su instance većih dimenzija po uzoru na instance koje mogu da se pronađu u radu [58] za slučaj Amsterdama. Za svaku instancu, generisan je izvestan broj tačaka sa slučajno odabranim *x* i *y* koordinatama između 0 i 1. Raspodela tačaka je uniformna. Sve tačke predstavljaju regije sa kojih se poziv može očekivati. Prvih 10% tačaka su potencijalne stanice ambulantnih vozila. Maksimalan dozvoljen broj vozila je dva puta veći od broja baznih stanica. Slično kao i za realni primer, dan je podeljen na 6 jednakih vremenskih intervala od 4 sata.

Tabela 2. Parametri testiranja u različitim vremenskim intervalima

Interval	Učestalost poziva	Vreme putovanja
00:00 – 04:00	0.6	0.8
04:00 – 08:00	0.5	1.1
08:00 – 12:00	1.2	1.1
12:00 – 16:00	1.5	0.9
16:00 – 20:00	1.3	1.2
20:00 – 24:00	0.9	0.9

Tokom dana očekivani broj poziva i brzina kretanja ambulantnih vozila se mogu razlikovati. U radu [51] se može naći da brzina kretanja u različitim periodima može da varira do 25% u odnosu na dnevni prosek. Podaci u Tabeli 2 su uzeti iz rada [58]. Na osnovu tih vrednosti se generišu stepen zauzetosti vozila i do kojih regija ambulantna vozila mogu doći u očekivanom vremenu. Stepen zauzetosti je relativan u odnosu na posmatrani vremenski interval, i računa se preko prosečnog broja poziva za taj interval, broja ambulantnih vozila i očekivanog vremena potrebnog da vozila budu slobodna nakon što dobiju poziv. Za prosečno vreme potrebno kolima da budu slobodna uzeto je 50 minuta, isto kao i u radu [58], što je vreme koje odgovara Holandiji za čije područje je rad [58] i pisan. Ovo se razlikuje od prosečnog trajanja intervencije u Beogradu. Vreme od 50 minuta je uzeto da bi se održala konzistentnost sa postojećom literaturom. Vozila iz bazne stanice mogu da odgovore na

poziv iz neke regije ako mogu da stignu do njih za manje od 15 minuta. Vreme potrebno da se iz jedne tačke (stanice) dospe do druge tačke (koja predstavlja regiju u gradu) se računa kao Euklidska distanca pomnožena sa konstantom, u ovom slučaju 35 (vrednost preuzeta iz rada [58]), pomnožena sa procenom odstupanja vremena putovanja od proseka za posmatrani interval iz tabele 2.

Za parametre β i γ uzete su vrednosti 0.001 i 0.00001. Vrednost β parametra 0.001 može se u ovom slučaju interpretirati da znači da će se nove bazne stanice uvoditi samo ako mogu da pokriju 0.1% do sada nepokrivenih poziva. Slično, γ parametar znači da će se premeštanja vozila tokom dana dozvoliti samo ako to dovede do povećanja očekivane pokrivenosti od najmanje 0.001%. Primetimo da se u radu [58] koriste vrednosti 0.005 i 0.00005. Za velike instance koje su korišćene u ovom radu, a nisu bile moguće u radu [58], ove vrednosti su davale rezultate u kojima je korišćen vrlo mali broj stanica, bez premeštanja vozila.

7.2 Rezultati dobijeni za realnu test instancu

Da bi smo dobili uvid u realnu primenljivost modela, evaluirali smo model dva puta. Prvi put smo fiksirali bazne stanice i vozila hitne pomoći na postojeće lokacije. Drugi put smo izračunali optimalne lokacije na osnovu modela. U oba slučaja korišćen je egzaktni rešavač CPLEX 12.1, a zatim i predložena metoda promenljivih okolina u cilju procene kvaliteta rešenja heuristike. Testiranja su izvršena na istoj platformi pomenutoj u sekciji 6.3.

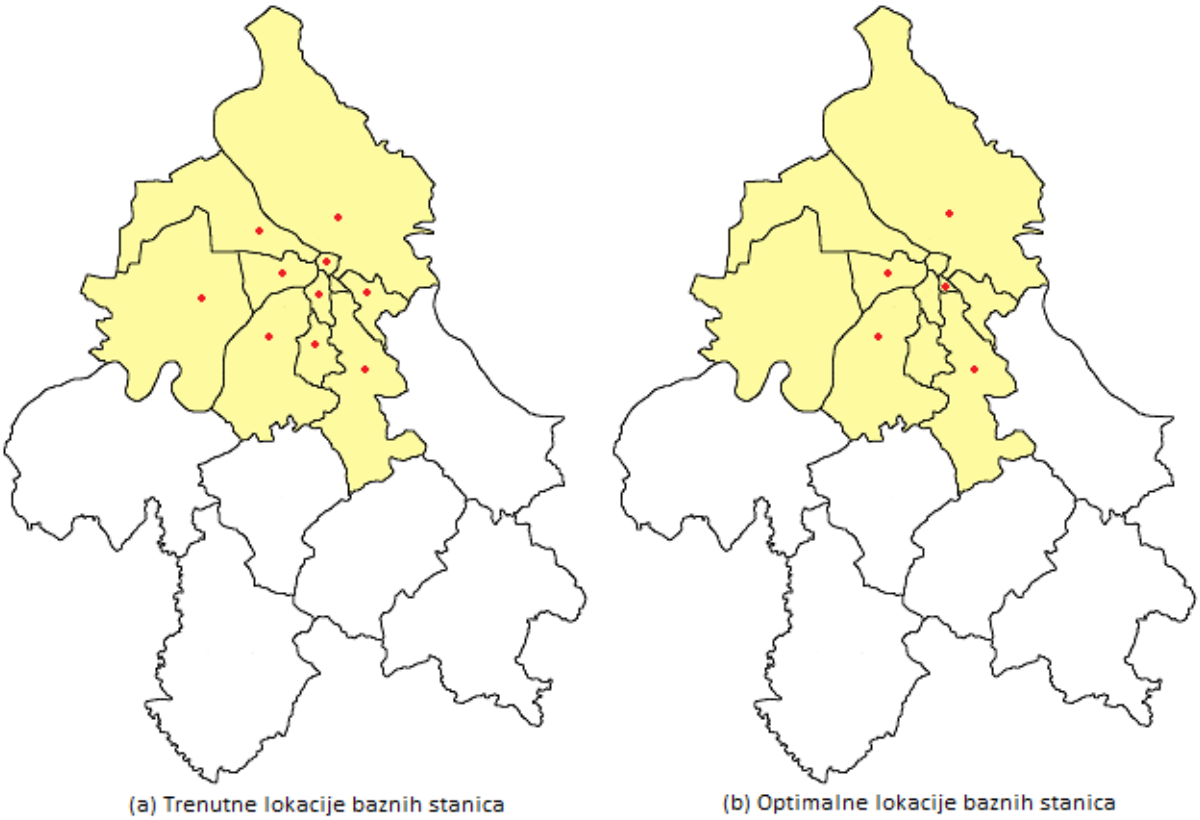
Isti rezultati su dobijeni korišćenjem CPLEX rešavača 12.1 i metodom promenljivih okolina. CPLEX rešavač 12.1 je davao rešenje nakon 0.872 sekundi, dok je heuristika davala rešenje nakon 0.758 sekundi. Učestalost poziva je bila normirana na 100, te je teorijski maksimalna moguća vrednost očekivane pokrivenosti takođe jednaka 100. Za parametre β i γ su uzete vrednosti 0.001 i 0.00001, respektivno. Trenutni raspored vozila i preporučeni raspored na osnovu modela se mogu videti u Tabeli 3 i na Slici 8.

Tabela 3. Raspored vozila hitne pomoći po opštinama Beograda

		Optimalni raspored na osnovu modela					
Opština	**Trenutni raspored**	**00-04**	**04-08**	**08-12**	**12-16**	**16-20**	**20-24**
Stari Grad	*1*						
Vračar	*0*	6	7	7	7	6	6
Savski Venac	*5*						
Rakovica	*1*						
Zvezdara	*1*						
Novi Beograd	*3*	7	6	6	6	7	7
Zemun	*2*						
Palilula	*4*	2	2	2	2	2	2
Voždovac	*2*	5	4	4	4	5	5
Čukarica	*2*	2	3	3	3	2	2
Surčin	*1*						

Očekivana pokrivenost dobijena na osnovu modela je za trenutni raspored jednaka 98.72, dok se rasporedom dobijenim algoritmom dobija vrednost 99.44. Beogradski zavod za hitnu medicinsku pomoć koristi veliki broj stanica, što indikuje da su vrednosti parametara β i γ koje odgovaraju realnoj situaciji značajno manje od onih korišćenih u prethodnoj proceni. Algoritam je davao slične rezultate i za manje vrednosti parametara. Specifično, kada su obe vrednosti 0, rezultat se razlikuje u tome što su samo 2 vozila u po 2 vremenska perioda premeštena iz opštine Vračar u opštinu Stari grad, i 1 vozilo u jednom periodu iz opštine Voždovac u opštinu Čukarica. Pri novim parametrima, očekivana pokrivenost na osnovu realne situacije je 99.72, a na osnovu algoritma 99.95.

Ukoliko se uračuna cena korišćenja baznih stanica, rezultati pokazuju da se vrednost ciljne funkcije može povećati za 0.72%, a da se pri tom koristi samo 5 umesto 10 stanica koje se trenutno koriste na teritoriji Beograda. Slično, ukoliko se pretpostavi da je dozvoljen proizvoljan broj stanica, vrednost ciljne funkcije se može povećati za 0.23%, pri smanjenju broja baznih stanica sa 10 na 6.



Slika 8. Trenutni i optimalni raspored baznih stanica

Na Slici 8 se može videti da je ukupan broj baznih stanica manji pri optimalnom rasporedu. Ovakvi rezultati ukazuju na to da se najveći broj poziva događa iz nekoliko poslovnih i komercijalnih centara.

7.3 Rezultati na generisanim instancama manjih dimenzija

Za sve test instance, broj ambulantnih kola i regiona je srazmeran broju potencijalnih baznih stanica $|W|$. Test instance su generisane za različite vrednosti $|W|$. Najmanja instanca ima 10 potencijalnih baznih stanica, i svaka sledeća instanca ima 10 potencijalnih stanica više. Algoritam za rešavanje problema linearnog programiranja preuzet iz rada [58], implementiran u CPLEX 12.1 rešavaču i rezultati dobijeni njegovim izvršavanjem su uzeti kao osnova za poređenje sa rezultatima dobijenim VNS algoritmom. Dobijeni rezultati su prikazani u Tabeli 4.

Tabela 4. Uporedni rezultati VNS algoritma, i algoritma linearnog programiranja

$\|W\|$	Max_{LP}	Max_{VNS}	T_{LP}	T_{VNS}^{S}	T_{VNS}^{E}	T_{red}
10	535.36	535.36	3.538	0.303	1.531	56.73
20	1161.13	1161.13	82.465	1.159	3.658	95.56
30	1769.33	1769.33	344.388	3.530	5.609	98.37
40	2384.20	2384.20	272.726	4.660	15.923	94.16
50	2976.18	2976.18	524.389	9.322	24.726	95.28
60	3578.55	3578.55	3442.764	121.762	133.480	96.12
70	4177.13	4177.13	2541.610	153.799	156.441	93.84
80	4772.26	4772.26	3393.810	220.890	335.245	90.12
90	5344.39	5348.46	7214.318	231.526	539.423	

Kolona Max_{LP} u Tabeli 4 označava najbolji rezultat dobijen izvršavanjem modela linearnog programiranja koji je implementiran u CPLEX rešavaču 12.1, dok kolona T_{LP} označava vreme koje je algoritmu bilo potrebno da dobije rezultate. CPLEX 12.1 rešavač je izvršavan na instancama počev od najmanje, sve do instance za čije rešavanje je bilo potrebno preko 2 sata (7200 sekundi). Rešavanje instance za $|W| = 90$ je prekinuto posle zadatog ograničenja od 7200 sekundi i zabeleženo je do tada najbolje zagarantovano pronađeno rešenje. U Tabeli 4 osenčene ćelije predstavljaju rešenja dobijena prekidom izvršavanja CPLEX 12.1 rešavača usled prekoračenja vremenskog ograničenja.

Kolone Max_{VNS} i T_{VNS}^{E} se odnose na VNS algoritam, i sadrže najbolje pronađeno rešenje, i vreme potrebno da se to rešenje pronađe. Preciznije, kolona T_{VNS}^{S} označava vreme koje je bilo neophodno VNS algoritmu da dostigne najbolje rešenje, dok je u koloni T_{VNS}^{E} zapisano vreme potrebno algoritmu da ispuni kriterijum zaustavljanja. Za svaku instancu vrednosti predstavljaju prosek u 10 testova sa različitim inicijalnim vrednostima.

Razlika u vremenu izvršavanja dva algoritma je zapisana u koloni T_{red} i predstavlja ubrzanje dobijeno korišćenjem VNS algoritma u odnosu na LP algoritam. Ubrzanje je predstavljeno u obliku procenata i dobija se formulom $100 \times \left(1 - \frac{T_{VNS}^{E}}{T_{LP}}\right)$.

U Tabeli 5 se mogu pronaći detaljniji podaci o rezultatima izvršavanja predloženog VNS algoritma. Kao osnova za poređenje uzeti su rezultati dobijeni trivijalnim, Greedy algoritmom, zapisani u koloni Max_{Greedy}. Algoritam počinje od svih praznih stanica. Pri svakom koraku proverava se koliko će se unapređenje u očekivanoj pokrivenosti dobiti ukoliko se novo vozilo postavi u neku stanicu u nekom vremenskom intervalu. Takve provere se vrše za svaku stanicu i svaki interval za koje to ograničenja dopuštaju, i novo vozilo se postavlja tako da se unapređenje maksimizuje za dati korak. Ovakav algoritam je sličan algoritmu najstrmijeg spuštanja.

Tabela 5. Detalji rezultata izvršavanja VNS algoritma na instancama manjih dimenzija

$\|W\|$	Max_{Greedy}	Max_{VNS}	$\%_{VNS}$	$AvgErr$
10	509.66	535.36	100	0.000
20	1111.85	1161.13	90	0.525
30	1699.89	1769.33	90	0.680
40	2274.78	2384.20	100	0.000
50	2809.45	2976.18	70	0.051
60	3359.50	3578.55	80	0.120
70	3873.91	4177.13	80	0.152
80	4382.11	4772.26	80	0.168
90	4843.46	5348.46	70	0.273

Kolona $\%_{VNS}$ predstavlja u kom procentu slučajeva je VNS algoritam dosegao optimalno rešenje. Za svaki test tokom koga je VNS algoritam dao sub-optimalno rešenje, zabeleženo je odstupanje od optimalnog rešenja i prosečno odstupanje je zapisano u koloni $AvgErr$. Greška je relativna u odnosu na trivijalno rešenje, tj. smatra se da razlika između vrednosti dobijenih Greedy i VNS algoritmom ima više smisla od razlike između VNS algoritma i 0 vrednosti. Formula za izračunavanje je prosek svih vrednosti $100 \times \frac{Max_{VNS} - R_i}{Max_{VNS} - Max_{Greedy}}$ za testove koji nisu dosegli optimalno rešenje, gde je R_i rezultat dobijen za partikularni test i.

7.4 Rezultati na generisanim instancama većih dimenzija

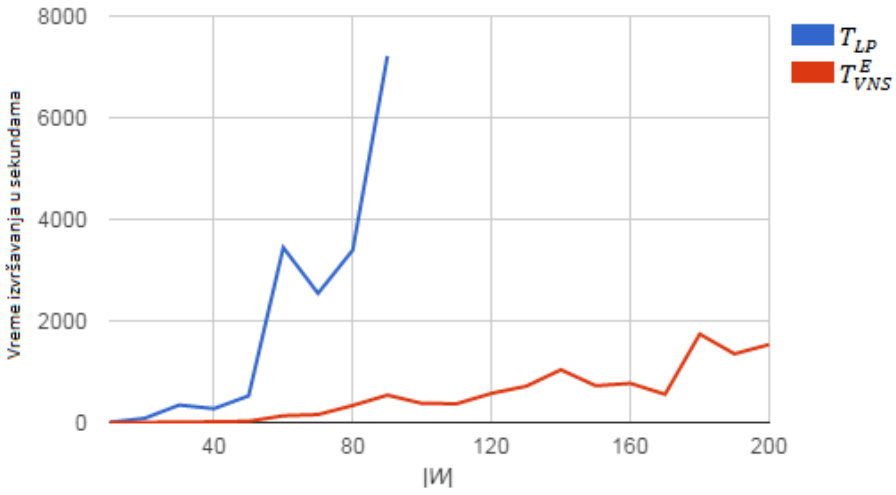
U slučaju generisanih instanci većih dimenzija, implementacija algoritma linearnog programiranja iz rada [58] ne može dati optimalna rešenja u doglednom vremenu izvršavanja. Dopušteno vreme izvršavanja algoritma linearnog programiranja je bilo 2 sata (7200 sekundi). Instanca sa 80 potencijalnih baznih stanica je bila instanca najvećih dimenzija za čije rešavanje je algoritmu linearnog programiranja bilo potrebno manje od 7200 sekundi. Iz tog razloga, na instancama većih dimenzija testiran je samo VNS algoritam koji se relativno brzo izvršava imajući u vidu dimenzije i težinu instanci.

Eksperimentalni rezultati dobijeni na generisanim instancama većih dimenzija su prikazani u Tabeli 6, na isti način kao u Tabelama 4 i 5. Za svaku partikularnu instancu, maksimalne dobijene vrednosti u 10 testova su dobijene u dovoljno velikom procentu testova (30% - 70%) da indikuju da su dobijena rešenja visokokvalitetna.

Tabela 6. Rezultati VNS algoritma na većim instancama

$\|W\|$	Max_{Greedy}	Max_{VNS}	T_{VNS}^{S}	T_{VNS}^{E}	$\%_{VNS}$
100	5347.97	5804.03	198.30	376.02	60
110	5806.88	6306.18	198.45	370.55	50
120	6251.41	7139.13	312.00	573.78	60
130	6678.34	7110.79	394.32	714.40	40
140	7105.73	7866.51	581.72	1038.42	30
150	7525.62	8218.89	411.44	723.81	60
160	7935.67	8772.98	444.61	770.97	50
170	8320.06	9136.97	324.65	555.00	70
180	8684.84	9872.48	1032.49	1740.41	40
190	9018.51	10358.88	812.82	1351.21	60
200	9370.57	10728.39	935.99	1534.77	40

Na Slici 9 se može videti značaj ubrzanja dobijenog korišćenjem heurističkog algoritma. Vremena izvršavanja, pored dimenzija instanci, značajno zavise i od konfiguracije. Međutim, predloženi algoritam promenljivih okolina je stabilno rešavao problem za instance velikih dimenzija ($|W| = 200$) za manje vremena nego što je algoritmu linearnog programiranja bilo potrebno za instance manjih dimenzija ($|W| = 60$).



Slika 9. Poređenje vremena izvršavanja na test instancama

8. ZAKLJUČAK

U ovom radu prikazana je implementacija osnovne metode promenljivih okolina za rešavanje problema raspoređivanja i preraspoređivanja vozila službe hitne pomoći po baznim stanicama koja odgovara karakteristikama problema i garantuje brzo izvršavanje. Dat je opis problema, teorijska osnova i opis implementacije metode, i analiza rezultata testiranja na instancama generisanim u skladu sa postojećom literaturom, kao i na realnoj instanci za teritoriju Beograda.

Naučni doprinos rada se ogleda u činjenici da je prvi put za ovaj problem upotrebljena metoda promenljivih okolina. Za instance manjih dimenzija koje algoritam linearnog programiranja implementiran u CPLEX 12.1 rešavaču uspeva da reši, metod pronalazi optimalne rezultate. Za instance većih dimenzija, algoritam linearnog programiranja ne pronalazi optimalno rešenje u doglednom vremenu. Izvršavanje predloženog algoritma osnovne metode promenljivih okolina se završava za značajno manje vremena, i činjenica da se u velikom broju testova za istu instancu dobija isto rešenje daje razlog da se veruje da su ta rešenja ujedno visokokvalitetna.

Dalji rad se može odvijati u više pravaca, među kojima su:

1. Granularnija paralelizacija koja se može skalirati na veliki broj procesora
2. Hibridizacija metode promenljivih okolina sa drugim heurističkim ili egzaktnim metodama
3. Unapređivanje modela, i to tako da:
   - uzeti u obzir mogućnost velikih incidenata kao što su saobraćajni udesi
   - uračunati lokaciju vozila tokom, i neposredno nakon pružanja usluge
   - uvesti u model mogućnost dodeljivanja različite hitnosti različitim pozivima
   - prilagođavanje modela za rešavanje problema optimizacije nekih drugih sistema za reagovanje u hitnim situacijama (vatrogasci, policija) uzimajući u obzir njihove specifičnosti
4. Poređenje sa drugim heurističkim metodama

LITERATURA

[1] Berge, C., & Ghouila-Houri, A. (1965). *Programmes, jeux et réseaux de transport*. Methuen.

[2] Brooks, R. L. (1941, April). On colouring the nodes of a network. U *Mathematical Proceedings of the Cambridge Philosophical Society* (Vol. 37, No. 02, pp. 194-197). Cambridge University Press.

[3] Čangalović, M. M., Kovačević-Vujčić, V. V., Ivanović, L., Dražić, M., & Ašić, M. D. (1996). Tabu search: A brief survey and some real-life applications. *Yugoslav journal of operations research*, *6*(1), 5-18.

[4] Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, *45*(1), 41-51.

[5] Church, R., & Velle, C. R. (1974). The maximal covering location problem. *Papers in regional science*, *32*(1), 101-118.

[6] Colorni, A., Dorigo, M., & Maniezzo, V. (1991, December). Distributed optimization by ant colonies. U *Proceedings of the first European conference on artificial life* (Vol. 142, pp. 134-142).

[7] Cook, S. A. (1971, May). The complexity of theorem-proving procedures. U *Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151-158). ACM.

[8] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm", year*.

[9] Cvetković D., Čangalović M., Dugošija Đ., Kovačević-Vujčić V., Simić S., Vuleta J. (1996). *Kombinatorna optimizacija: Matematička teorija i algoritmi*. Beograd: Društvo operacionih istraživača Jugoslavije.

[10] Dantzig, G. B., & Thapa, M. N. (2006). *Linear programming 1: introduction*. Springer Science & Business Media.

[11] Daskin, M. S. (1983). A maximum expected covering location model: formulation, properties and heuristic solution. *Transportation Science*, *17*(1), 48-70.

[12] Donati, A. V., Darley, V., & Ramachandran, B. (2008). An Ant-bidding Algorithm for Multistage Flowshop Scheduling Problem: Optimization and Phase Transitions. U *Advances in Metaheuristics for Hard Optimization* (pp. 111-136). Springer Berlin Heidelberg.

[13] Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*.

[14] Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, *1*(1), 53-66.

[15] Dowsland, K. A. (1993). Simulated annealing, modern heuristic techniques for combinatorial problems, Ed. CR Reeves.

[16] Erdös, P., & Szekeres, G. (1935). A combinatorial problem in geometry. *Compositio Mathematica*, *2*, 463-470.

[17] Farahani, R. Z., Asgari, N., Heidari, N., Hosseininia, M., & Goh, M. (2012). Covering problems in facility location: A review. *Computers & Industrial Engineering*, *62*(1), 368-407.

[18] Fidanova, S. (2003). ACO algorithm for MKP using various heuristic information. U *Numerical Methods and Applications* (pp. 438-444). Springer Berlin Heidelberg.

[19] Fred G. (1989). Tabu Search - Part 1. *ORSA Journal on Computing 1*, 190-206.

[20] Fred G. (1990). Tabu Search - Part 1. *ORSA Journal on Computing 2*, 4-32.

[21] Gendreau, M., Laporte, G., & Semet, F. (1997). Solving an ambulance location model by tabu search. *Location science*, *5*(2), 75-88.

[22] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, *13*(5), 533-549.

[23] Glover, F., & Taillard, E. (1993). A user's guide to tabu search. *Annals of operations research*, *41*(1), 1-28.

[24] Grant, K. (1995). An introduction to genetic algorithms. *C/C++ Users Journal*, *13*(3), 45-58.

[25] Hansen, P., & Mladenović, N. (2005). *Variable neighborhood search* (pp. 211-238). Springer US.

[26] Holland, J. (1975). Adaptation in artificial and natural systems. *Ann Arbor: The University of Michigan Press*.

[27] Hopcroft, J.E., Motwani, R., and Ullman, J.D. (2001). *Introduction to Automata Theory, Languages, and Computation,* Massachusetts: Addison Wesley.

[28] Jagtenberg, C. J., Bhulai, S., & van der Mei, R. D. (2015). An efficient heuristic for real-time ambulance redeployment. *Operations Research for Health Care*, *4*, 27-35.

[29] Juan, A. A., Pascual, I., Guimarans, D., & Barrios, B. (2014). Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem. *International Transactions in Operational Research*.

[30] Karp, R. M. (1972). *Reducibility among combinatorial problems* (pp. 85-103). springer US.

[31] Kennedy, J. (1997, April). The particle swarm: social adaptation of knowledge. U *Evolutionary Computation, 1997., IEEE International Conference on* (pp. 303-308). IEEE.

[32] Kennedy, J.; Eberhart, R. (1995). „Particle Swarm Optimization“. *Proceedings of IEEE International Conference on Neural Networks* **IV**. pp. 1942–1948.

[33] Kennedy, J.; Eberhart, R.C. (2001). *Swarm Intelligence*. Morgan Kaufmann.

[34] Kim, S. H., & Lee, Y. H. (2015). Iterative optimization algorithm with parameter estimation for the ambulance location problem. *Health care management science*, 1-21.

[35] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671-680.

[36] Larsen, M. P., Eisenberg, M. S., Cummins, R. O., & Hallstrom, A. P. (1993). Predicting survival from out-of-hospital cardiac arrest: a graphic model. *Annals of emergency medicine*, *22*(11), 1652-1658.

[37] Lawler, E. L. (1985). The traveling salesman problem: a guided tour of combinatorial optimization. *WILEY-INTERSCIENCE SERIES IN DISCRETE MATHEMATICS*.

[38] Lourenço, H. R., & Zwijnenburg, M. (1996). Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In *Meta-Heuristics* (pp. 219-236). Springer US.

[39] Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). *Iterated local search* (pp. 320-353). Springer US.

[40] Lourenço, H. R., Martin, O. C., & Stützle, T. (2010). Iterated local search: Framework and applications. In *Handbook of Metaheuristics* (pp. 363-397). Springer US.

[41] Luce, R. D., & Perry, A. D. (1949). A method of matrix analysis of group structure. *Psychometrika*, *14*(2), 95-116.

[42] Penna, P. H. V., Subramanian, A., & Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, *19*(2), 201-232.

[43] Pham, D. T., & Castellani, M. (2009). The Bees Algorithm: modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, *223*(12), 2919-2938.

[44] Pham, D. T., & Castellani, M. (2014). Benchmarking and comparison of nature-inspired population-based continuous optimisation algorithms. *Soft Computing*,*18*(5), 871-903.

[45] Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2005). The bees algorithm. Technical note. *Manufacturing Engineering Centre, Cardiff University, UK*, 1-57.

[46] Pham, D. T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., & Zaidi, M. (2006). The Bees Algorithm–A Novel Tool for Complex Optimisation Problems. Intelligent Production Machines and Systems. DT Pham, EE Eldukhri and AJ Soroka (eds). Cardiff University. *Manufacturing Engineering Centre, Cardiff, UK. Published by Elsevier Ltd*.

[47] Poli, R. (2007). An analysis of publications on particle swarm optimization applications. *Essex, UK: Department of Computer Science, University of Essex*.

[48] Poli, R. (2008). Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, *2008*, 3.

[49] Reeves, C. R. (1993). *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc..

[50] Russel, S., & Norvig, P. Artificial Intelligence: A Modern Approach, 2003. *EUA: Prentice Hall*.

[51] Schmid, V., & Doerner, K. F. (2010). Ambulance location and relocation problems with time-dependent travel times. *European Journal of Operational Research*, *207*(3), 1293-1303.

[52] Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency* (Vol. 24). Springer Science & Business Media.

[53] Shi, Y., & Eberhart, R. (1998, May). A modified particle swarm optimizer. U *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on* (pp. 69-73). IEEE.

[54] Sierksma, G. (2001). *Linear and integer programming: theory and practice*. CRC Press.

[55] Suzuki, S., Okada, M., Das, A., & Chakrabarti, B. K. (2005). Quantum annealing and related optimization methods. *Lecture Notes in Physics*, *679*, 207-238.

[56] Toregas, C., Swain, R., ReVelle, C., & Bergman, L. (1971). The location of emergency service facilities. *Operations Research*, *19*(6), 1363-1373.

[57] *User's Manual for CPLEX - ibm.* Preuzeto 23. septembra 2015, sa ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf

[58] van den Berg, P. L., & Aardal, K. (2015). Time-dependent MEXCLP with start-up and relocation cost. *European Journal of Operational Research*, *242*(2), 383-389.