UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

MASTER RAD

Rešavanje nekih problema kombinatorne optimizacije algoritmom tabu pretraživanja

---

Student:
Miloš STANKOVIĆ

Mentor:
Doc. dr Miroslav MARIĆ

Beograd, 2013

Sadržaj

Rezime

U ovom radu razmatra se metaheuristika tabu pretraživanja za rešavanje nekih problema kombinatorne optimizacije. U radu su predstavljeni istorijski podaci o nastanku i razvoju ovog algoritma. Dat je detaljan opis osnovnih komponenti algoritma kao što su kratkoročna memorija (tabu lista) i zabranjeni (tabu) potezi. Ukazano je na razlike u odnosu na heurisitku lokalnog pretraživanja. Tu se pre svega misli na kratkoročnu memoriju i tabue. Takođe, razmatrana su i proširenja osnovnih koncepata algoritma od kojih su najznačajniji mehanizmi diversifikacije i intenzifikacije rešenja za čije potrebe se uvodi nova vrsta memorije – dugoročna memorija. U cilju efikasnije pretrage razmatrana je modifikacija funkcije cilja problema, a u vezi sa tim i metoda strateškog oscilovanja koja dopušta pretraživanje nedopustivih rešenja.

U radu su dva poznata problema rešavana metodom tabu pretraživanja. Prvi od njih jeste problem generalizovanog pridruživanja čiji je cilj pronaći optimalno pridruživanje skupa poslova skupu agenata pri određenim uslovima ograničenja koji su navedeni u radu. Drugi rešavani problem je problem rutiranja vozila sa vremenskim ograničenjima. Zadatak je pronaći skup ruta koje polaze iz skladišta i obilaze sve korisnike tako da ukupno pređeno rastojanje bude minimalno, pri čemu je neophodno da svaki korisnik bude posećen u određenom vremenskom intervalu. Ostali uslovi ograničenja navedeni su u radu. Za oba problema predstavljeni su eksperimentalni rezultati, kao i analiza dobijenih rezultata.

Ključne reči: Tabu pretraživanje, Generalizovani problem pridruživanja, Problem rutiranja vozila sa vremenskim ograničenjima, Metaheuristika, Kombinatorna optimizacija

Abstract

This paper considers tabu search metaheuristics for solving same problems of combinatoral optimisation. First, historical information about this algorithm are presented. Differences between tabu search metaheuristics and local search heuristic are shown, especially short term memory and tabus. Extensions of basic concepts are presented including the most important mechanisms intensification and diversification. These mechanisms needs introduction new sort of adaptive memory – long term memory. In order to make search more efficient it is applied modification of objective function. Related to that, it is discussed about strategic oscilation procedure, which allows considering infesible solutions.

Two well-known problems are solved using proposed tabu search algorithm. First problem is generalized assignment problem which aims to find optimal assignment between jobs and agents. Additional constraints are presented in this paper. Second problem is vehicle routing problem with time constraints. The task is finding optimal set of vehicle routes, which start and end in depot and visit each customer location only once. Other constraints are presented in following text. Computational experiments have been performed and analysis of obtained results is given.

Keywords: Tabu search, Generalized assignment problem, Vehicle routing problem with time constraints, Metaheuristics, Combinatoral optimisation

1. Uvod

Tehnološki razvoj čovečanstva susretao se sa raznim problemima optimizacije, a u većini slučajeva radilo se o problemima kombinatorne optimizacije. Sa razvojem teorije složenosti došlo se do saznanja da veliki broj problema kombinatorne optimizacije pripada klasi NP-teških problema, tj. da je njihovo rešavanje egzaktnim metodama praktično nemoguće. Naspram egzaktnih metoda, heurističke metode mogle su da daju u razumnom vremenu veoma kvalitetno rešenje za koje ne postoji dokaz da je i optimalno. Zbog toga one su privukle veliku pažnju akademske zajednice, a posledica toga jeste ekspanzija heurističkih i metaheurističkih metoda.

U ovom radu razmatra se metaheuristika tabu pretraživanja za rešavanje nekih problema kombinatorne optimizacije. Na početku rada dat je istorijski pregled razvoja ovog algoritma. Dat je detaljan opis osnovnih komponenti algoritma kao što su kratkoročna memorija (tabu lista) i zabranjeni (tabu) potezi. Razmatrane su različite mogućnosti za realizaciju kratkoročne memorije i tabu poteza, kao i njihove prednosti i mane. Kao bitan parametar tabu liste je izdvaja se i njena dužina. Ukazano je na razlike u odnosu na heurisitku lokalnog pretraživanja. Osnovni algoritam u često daje zadovoljavajuće rezultate. Ipak, u cilju poboljšanja algoritma razmatrana su i proširenja osnovnih koncepata tabu pretraživanja. Najznačajniji su mehanizmi diversifikacije i intenzifikacije rešenja za čije potrebe se uvodi nova vrsta memorije – dugoročna memorija. U cilju efikasnije pretrage razmatrana je modifikacija funkcije cilja problema, a u vezi sa tim i metoda strateškog oscilovanja koja dopušta pretraživanje nedopustivih rešenja. Kod algoritma tabu pretraživanja, zbog poboljšanja vremenskih performansi, često je potrebno suziti skup mogućih rešenja za izbor novog tekućeg rešenja. Tehnike za smanjenje okoline razmatrane su u sekciji 2.3.5.

U radu su dva poznata problema rešavana metodom tabu pretraživanja. Prvi od njih je generalizovani problem pridruživanja čiji je cilj pronaći optimalno pridruživanje skupa poslova skupu agenata pri određenim uslovima ograničenja koji su navedeni u radu. Pomenute praktične primene ovog problema i ukazano je na njegov značaj. Pored razmatranja implementacije osnovnih komponenti tabu pretraživanja, u radu se diskutuje o načinu predstavljanja rešenja, okoline, o generisanju početnog rešenja, kao i o relaksaciji problema. Na kraju poglavlja o ovom problemu prikazani su eksperimentalni rezutati.

Drugi rešavani problem je problem rutiranja vozila sa vremenskim ograničenjima. Zadatak je pronaći skup ruta koje polaze iz skladišta i obilaze sve korisnike tako da ukupno pređeno rastojanje bude minimalno, pri čemu je neophodno da svaki korisnik bude posećen u određenom vremenskom intervalu. Ostala ograničenja kao i matematički model predstavljeni su u odeljku posvećenom ovom problemu. Ukazano je na značaj rešavanja ovog problema, pre svega ekonomski. Predložen je algoritam za rešavanje ovog problema, a takođe razmatrani su detalji vezani za njegovu implementaciju. Na kraju su diskutovano je o eksperimentalnim rezultatima.

Na samom kraju rada razmatra se mogućnost hibridizacije tabu pretraživanja sa drugim metaheuristikama, kao što su: simulirano kaljenje, genetski algoritmi i metoda promenljivih okolina. Svaki od ovih algoritama ima svoje prednosti i nedostatke i na osnovu toga data je diskusija o mogućoj hibridizaciji kao i o prednostima koje ona donosi.

2. Tabu pretraživanje

2.1. Istorija

Pri rešavanju teških optimizacijskih problema iz realnog života, egzaktne metode veoma često susreću se sa velikim poteškoćama. Razlog za to je što je najčešće reč o NP-teškim problemima. Rešavanje problema koji su od životnog značaja u različitim oblastima kao što su ekonomija, poslovanje, nauka, medicina, ne bi bilo moguće egzaktnim metodama kojima se pridavao veliki značaj u prošlosti (i još uvek se pridaje u pojedinim tekstovima).

Heurističke metode, koje se još nazivaju tehnike približnog rešenja, koriste se za rešavanje teških kombinatornih problema od početka operacionih istraživanja. Sa razvojem teorije složenosti ranih 70-tih godina XX veka, postalo je jasno da većina tih problema spada u klasu NP-teških problema i da je malo nade da će se ikad naći efikasna egzaktna metoda za njihovo rešavanje. Zbog toga u rešavanju kombinatornih problema koji su uključeni u aplikacije iz realnog života glavnu ulogu zauzele su heurističke metode, bez obzira na to da li su ti problemi NP-teški ili ne.

Među mnogim predstavljenim metodama najpopularnije su bile one koje su se bazirale na lokalnom pretraživanju. One se mogu grubo opisati kao iterativne metode pretraživanja koje polaze od početnog dopustivog rešenja unapređujući ga primenom serija lokalnih modifikacija ili pokreta. Kvalitet rešenja i vreme izvršavanja značajno zavise od raznovrsnosti skupa transformacija koje se primenjuju u svakoj iteraciji kao i od početnog rešenja.

Godine 1983. predstavljena je nova heuristička metoda nazvana simulirano kaljenje [32]. Zasnovano na analogiji sa procesom kaljenja čelika, simulirano kaljenje se može opisati kao kontrolisano slučajno kretanje kroz prostor dopustivih rešenja. Pojava algoritma simuliranog kaljenja prouzrokovala je novi pogled na rešavanje problema kombinatorne optimizacije i podstakla razvoj heurističkih algoritama. U kasnijim godinama predstavljeno je mnogo novih algoritama, najviše baziranih na prirodnim fenomenima, kao što su metoda zasnovana na ponašanju mrava (*Ant Colony Optimization* – ACO), metoda zasnvana na ponašanju roja čestica (*Particle Swarm Optimization* – PCO), metoda veštačkog imuniteta (*Artificial Immune System* – AIS), tabu pretraživanje (*Tabu Search* – TS). Zajedno sa nekim starijim algoritmima kao što su genetski algoritmi (*Genetic Algorithms* – GA), dostigli su veliku popularnost. Poznate pod nazivom metaheuristike, ove metode u poslednjih 20 godina imaju vodeću ulogu u rešavanju problema kombinatorne optimizacije.

Godine 1986. Fred Glover, profesor računarskih nauka Univerziteta Kolorado, predstavio je novi heuristički pristup nazvan tabu pretraživanje [18]. Zanimljivo je da je u paralelnom radu slični pristup nazvan najstrmiji uspon / najblaži pad (s*teepest ascent / mildest descent*) predložio je P. Hansen iste godine. Koreni tabu pretraživanja mogu se sresti u ranijim radovima Freda Glovera gde je većina elemenata ovog algoritma razmatrana [19]. Među njima je npr. kratkoročna memorija o kojoj će biti reči u nastavku. Važno je napomenuti da Glover nije posmatrao tabu pretraživanje kao klasičnu heuristiku već radije kao metaheuristiku, tj. kao generalnu strategiju za vođenje i kontrolisanje unutrašnje heuristike skrojene prema specifičnom problemu koji se rešava.

Poslednjih 20 godina predstavljeno je na stotine radova koji razmatraju primenu tabu pretraživanja na različite probleme kombinatorne optimizacije. Neki od njih su [20], [21], [43], [44], [46], [59]. U većini slučajeva, opisane metode su dostizale optimalna rešenja ili rešenja veoma blizu optimalnih, i pokazale su se kao veoma efikasne metode za rešavanje gotovo svih vrsta problema kombinastorne optimizacije. Ovi uspesi učinili su da tabu pretraživanje postane ekstremno popularna metoda među onima koji su zainteresovani za pronalaženje kvalitetnih rešenja velikih kombinatornih problema sa praktičnim primenama.

Moguće je uočiti vezu između pojma tabu i tabu pretraživanja. Reč "tabu" potiče iz Tonganskog jezika, jezika Polinezije, gde su je Aboridžini ostrva Tonga koristili da označe predmete koji ne smeju biti dodirivani zato što su sveti. Danas se ta reč odnosi na "zabranu nametnutu društvenim običajima ili emocionalnim averzijama kao meru zaštite" ili "nešto zabranjeno jer sadrži rizik". Ove definicije daju pragmatičniji smisao reči tabu a takođe ukazuju i na karakteristike tabu pretraživanja. Karakteristika rizika odnosi se na sve heuristike koje u sebi sadrže pretraživanje (tzv. S heuristike), jer kretanje (tj. pretraživanje) u smerovima koji donose poboljšanje sadrži rizik od dobijanja ne tako dobrog rešenja. S druge strane, zabrana kao mera zaštite može biti prevaziđena kada okolnosti to zahtevaju, a u slučaju tabu pretraživanja "tabui" se zanemaruju sa ciljem da se dostigne što bolje rešenje. Najvažnija asocijacija tabu pretraživanja sa pojmom tabua odnosi se na činjenicu da se u realnom svetu tabu menja kroz vreme za šta je odgovorna neka vrsta "društvene memorije". Kod tabu pretraživanja memorija igra ključnu ulogu u određivanju i menjanju (tabu) statusa elemetima tabu pretraživanja kroz vreme i u odnosu na okolnosti.

2.2. Tabu pretraživanje

Tabu pretraživanje dramatično povećava sposobnosti za rešavanje problema od praktičnog značaja. Primena ovog algoritma je velika i susreće se u raznim oblastima kao sto su telekomunikacije, finansijske analize, planiranje prostora i resursa, problemi raspoređivanja, problemi rutiranja, distribucija energije, biomedicinska analiza, molekularno inžinjerstvo, dizajn mikročipova sa visokim stepenom integracije (*Very La*rge *Scale Integration* – VLSI), i mnoge druge (tabela 2.1.). Poslednjih godina različiti časopisi objavili si veliki broj članaka i rezultata istraživanja koji govore o uspehu i pomeranju granica u primenama tabu pretraživanja na različitim poljima.

Tabu pretraživanje je iterativni postupak koji polazi od datog početnog rešenja i pokušava da ga unapredi u svakoj iteraciji. Osnovni princip tabu pretraživanja je da nastavi sa pretragom kad god naiđe na lokalni optimum primenom poteza koji ne donose poboljšanje sa ciljem da se lokalni optimum prevaziđe. Da se pretraga ne bi vraćala u već posećena rešenja potezi koji vode do njih postaju zabranjeni, tj. tabu potezi. Informacije o pretrazi se čuvaju u memoriji a vraćanje u već posećena rešenja sprečeno je na osnovu tih informacija. Ključna ideja da se za upravljanje pretragom koriste informacije o prethodnom periodu pretrage može se povezati sa metodama informisanog pretraživanja u polju veštačke inteligencije.

Tabela 2.1. Primeri primene tabu pretraživanja

Raspoređivanje Raspoređivanje heterogenih procesora Raspoređivanje radne snage Raspoređivanje mašina Raspoređivanje poslova **Dizajn** Dizajn transportnih mreža VLSI dizajn Dizajn mreža sa fiksnim obračunom **Lociranje i alociranje** Problemi kvadratnog pridruživanja Polukvadratno pridruživanje Generalizovano pridruživanje Lokacijski problemi (sa ograničenim kapacitetima) Prioblasko istraživanje nalazišta nafte **Logika i veštačka inteligencija** Prepoznavanje obrazaca Problemi grupisanja Integritet podataka Neuronske mreže **Tehnologija** Seizmička inverzija Distribucija električne energije Konstrukcija svemirskih stanica	**Telekomunikacije** Rutiranje poziva Sinhronizacija optičkih mreža Planiranje popusta za korisnike **Proizvodnja, investiranje** Fleksibilna proizvodnja Ograničeno planiranje zahteva za materijalima (Capacitated MRP) Fiksno mešovito investiranje **Rutiranje** Rutiranje vozila Rutiranje vozila sa kapacitetima Rutiranje vozila sa vremenskim ograničenjima Rutiranje vozila mešovite flote Problem trgovačkog putnika **Grafovi** Bojenje grafova Deljenje grafova Problemi p-medijane **Kombinatorna optimizacija** Celobrojno programiranje 0-1 programiranje Mešovito celobrojno programiranje Nekonveksno nelinearno programiranje

2.2.1. Pretraživački prostor i okoline

Algoritam tabu pretraživanja jeste proširenje metode lokalnog pretraživanja. Osnovni TS algoritam je kompozicija lokalne pretrage i adaptivne memorije. Iz toga sledi da se svaki TS algoritam mora pozabaviti definicijom pretraživačkog prostora i okolina. Izbor pretraživačkog prostora i okoline rešenja je najkritičniji korak u dizajnu svakog tabu pretraživanja, stoga je neophodno posvetiti više pažnje ovim komponentama TS algoritma.

Pretraživački prostor je skup svih mogućih rešenja koja mogu biti posećena tokom pretrage. Struktura pretraživačkog prostora zavisi od konkretnog problema. Jedan od najpoznatijih problema je problem raspoređivanja poslova (*Job-Shop Scheduling Problem* – JSSP). Kod JSSP-a dato je n poslova i m mašina. Svaki posao se sastoji od fiksne sekvence m operacija, a svaka operacija se mora izvršavati na određenoj (specifičnoj) mašini i to izvršavanje traje određeno vreme. Svaka mašina kada započne izvršavanje operacije ne sme je prekinuti, tj. započeta operacija se mora izvršiti u celosti. Cilj je napraviti raspored izvršavanja operacija poslova tako da se minimizuje vreme izvršavanja tih poslova. Rešenje se može predstaviti kao skup od m permutacija n poslova. Kod JSSP-a pretraživački prostor se može definisati kao skup svih dopustivih rešenja problema. U ovom slučaju definicija pretraživačkog prostora je prilično prirodna, ali postoje i slučajevi gde nije tako.

Drugi veoma poznati problem je problem lociranja postrojenja (skladišta, fabrika...) sa kapacitetima (*Capacited Plant Location Problem* – CPLP). Kod CPLP-a dat je skup korisnika (potrošača) I, i za svakog korisnika $i \in I$ dat je njegov zahtev d_i, tj. količina proizvoda koju korisnik potražuje. Dat je skup potencijalnih lokacija za postrojenja J i za svaku lokaciju $j \in J$ data je fiksna cena f_j uspostavljanja postrojenja na toj lokaciji. Postrojenja snabdevaju korisnike proizvodima i za svaku lokaciju $j \in J$ dat je kapacitet postrojenja K_j na toj lokaciji. Korisnik se može snabdevati proizvodima iz različitih postrojenja i za svakog korisnika $i \in I$ i za svaku lokaciju $j \in J$ data je cena c_{ij} transporta jedinice proizvoda sa lokacije j do korisnika i. Cilj je minimizovati ukupne troškove otvaranja postrojenja i transporta proizvoda. Neka je sa x_{ij} $(i \in I, j \in J)$ označena količina proizvoda koju postrojenje j dostavlja korisniku i (tzv. promenljive protoka), i neka su y_j $(j \in J)$ binarne promenljive koje imaju vrednost 1 ako je postrojenje otvoreno na lokaciji j, inače imaju vrednost 0. CPLP se matematički može zapisati na sledeći način:

$$min\, z = \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{2.1.}$$

pri uslovima: $\sum_{j \in J} x_{ij} = d_i, \forall i \in I$ (2.2.)

$$\sum_{i \in I} x_{ij} \leq K_j y_j, \forall j \in J \tag{2.3.}$$

$$x_{ij} \in \mathbb{Z}, x_{ij} \geq 0, \forall\, i \in I, \forall j \in J \tag{2.4.}$$

$$y_j \in \{0,1\}, \forall j \in J \tag{2.5.}$$

Ovde pretraživački prostor može biti skup tačaka koje u sebi sadrže binarne lokacijske promenljive kao i promenljive protoka. Ovakvo predstavljanje pretraživačkog prostora je jako komplikovano. Atraktivniji pretraživački prostor može se dobiti restrikcijom prethodnog na skup binarnih vektora (zanemaruju se promenljive protoka). Svakom ovakvom vektoru pridružen je i transportni problem čijim rešavanjem se određuju vrednosti preostalih neprekidnih promenljivih. Ove dve mogućnosti nisu jedine. Pretraživanje se može vršiti na skupu ekstremnih tačaka (tj. temena) poliedra koga određuju uslovi (2.2.), (2.3.), (2.4.). Lokacijske promenljive se mogu dobiti na jednostavan način jer postrojenje mora biti postavljeno na određenoj lokaciji ako se sa te lokacije očekuje dostava proizvoda korisnicima (tj. $x_{ij} > 0$). Važno je napomenuti da nije uvek dobro ograničiti pretraživački prostor na skup dopustivih rešenja. U velikom broju slučajeva dozvoljavanje pretrazi da razmatra nedopustiva rešenja je poželjno, a nekad je to i neophodno. U tom slučaju, u pretraživačkom prostoru se pored dopustivih rešenja nalaze i nedopustiva.

U bliskoj vezi sa definicijom pretraživačkog prostora jeste definicija okolina. U svakoj iteraciji (lokalnog ili tabu) pretraživanja, lokalne transformacije koje mogu biti primenjene na tekuće rešenje s definišu skup susednih rešenja ili okolinu rešenja označenu sa $N(s)$. Lokalne transformacije nazivaju se još i potezima. Okolina $N(s)$ rešenja s formalno se može predstaviti kao podskup skupa pretraživačkog prostora na sledeći način:

$$N(s) = rešenja\ dobijena\ primenom\ jedne\ lokalne\ transformacije\ na\ tekuće\ rešenje\ s$$

U opštem slučaju, za svaki razmatrani problem postoji više različitih okolina nego pretraživačkih prostora što proističe iz činjenice da može postojati više različitih okolina za datu definiciju pretraživačkog prostora. Ovo se jednostavno može ilustrovati na JSSP. Neka je pretraživački prostor skup svih dopustivih rešenja. Jednostavna okolina se može dobiti razmatranjem sekvenci poslova pridruženih svakoj mašini, gde pozicija posla u sekvenci odgovara

redosledu izvršavanja posla na mašini, kada se jednom poslu promeni pozicija ili kada dva posla zamene pozicije. Nasuprot ovoj okolini koja se dobija jednostavnim potezima, neke okoline mogu biti veoma velike. U tim slučajevima operacija pretrage cele okoline je jako zahtevna operacija (u vremenskom smislu), pa je neophodno na neki način izvršiti restrikciju takvih okolina.

Različite definicije pretraživačkog prostora zahtevaju definisanje različitih okolina rešenja, što se najbolje može ilustrovati na CPLP. Ako se pretraživački prostor sastoji od binarnih vektora (tj. vektora čije su koordinate lokacijske promenljive), okolina rešenja se može dobiti primenom poteza dodavanja/odbacivanja (*Add/Drop*) ili poteza zamene (*Swap*). Potezi dodavanja/odbacivanja uključuju promenu na jednoj lokaciji (tj. na jednoj koordinati). Ako je na datoj lokaciji postavljeno postrojenje ovim potezom postrojenje se gasi, i obrnuto. Potez zamene menja lokaciju postrojenja, tj. na jednoj lokaciji se postrojenje gasi a na drugoj otvara (ovaj potez je kompozicija najpre poteza odbacivanja a zatim poteza dodavanja). U slučaju pretraživačkog prostora koji je skup temena poliedra definisanog uslovima (2.2.), (2.3.), (2.4.), navedeni potezi nemaju smisao, a definisanje odgovarajućih poteza je komplikovano.

Tabu pretraživanje je proširenje metode lokalnog pretraživanja memorijom, ali i drugim komponentama. Lokalno pretraživanje je iterativna heuristička metoda koja polazi od nekog početnog rešenja i pokušava da ga unapredi u svakoj iteraciji. U svakoj iteraciji pretraga pronalazi najbolje rešenje u okolini tekućeg rešenja koje postaje novo tekuće rešenje. Pretraga staje kada više nije moguće unaprediti tekuće rešenje. To znači da je pretraga došla u lokalni optimum i jasno je da to rešenje može biti daleko od globalnog optimuma. Cilj proširenja lokalnog pretraživanja je da se pretraga osposobi za prevazilaženje lokalnih optimuma, kao i da na efikasan način pretraga dosegne globalni optimum. Ako je potrebno minimizovati funkciju f na nekom domenu, onda šema lokalnog pretraživanja izgleda ovako:

Notacija:

- s_0 – početno rešenje,
- s – tekuće rešenje,
- s^* – najbolje poznato rešenje,
- f^* – vrednost funkcije cilja rešenja s^*,

Šema 2.1. Lokalno pretraživanje

$Inicijalizacija$:
1. $s_0 = KonstrukcijaPočetnogRešenja()$;
2. $s = s_0, s^* = s_0,\ f^* = f(s_0)$;

$Pretraga$:
3. $s' = IzaborNajboljegSusedaUOkolini(N(s))$;
4. $s = s'$;
5. **while**$(f(s) < f^*)$
6. $\quad f^* = f(s)$;
7. $\quad s^* = s$;
8. $\quad s' = IzaborNajboljegSusedaUOkolini\big(N(s)\big)$;
9. $\quad s = s'$;
   endwhile
10. **return** s^*;

Navedena šema algoritma u koraku 3. podrazumeva ispitivanje svih elemenata okoline $N(s)$ i pronalaženje najboljeg rešenja među njima. Drugi često korišćeni pristup je da se ispitivanje okoline obavlja do pronalaženja prvog rešenja koje je bolje od tekućeg rešenja.

2.2.2. Tabui

Tabu je jedan od karakterističnih elemenata tabu pretraživanja u poređenju sa lokalnim pretraživanjem. Kao što je već pomenuto, tabui se koriste kako bi se sprečili ciklusi pri udaljavanju od lokalnog optimuma kroz poteze koji ne dovode do boljih rešenja. Kada proces pretrage dospe u lokalni optimum pokušava se prevazilaženje lokalnog optimuma potezima koji ne dovode do poboljsanja. Tim potezima se u okolini tekućeg rešenja bira novo rešenje koje nije bolje od tekućeg i proces pretrage se nastavlja u okolini tog (novog) rešenja. Neophodno je onemogućiti da se pretraga vrati u već posećena rešenja jer time nastaju ciklusi, a to se postiže tako što se posećena rešenja proglase zabranjenim (tabu) u određenom broju narednih iteracija. Taj broj se naziva tabu mandat (*Tabu Tenure*). Potpuna rešenja bi se mogla čuvati, ali to nije efikasno jer zahteva veliku količinu memorije. Pored toga, potrebno je dosta vremena za proveru da li je potencijalno rešenje tabu ili ne, pa se ova opcija retko koristi. Alternativa je da se umesto celih rešenja zabrane potezi koji vode do njih. Kako će se u tom slučaju predstaviti tabui zavisi od karakteristika problema i načina predstavljanja rešenja (tabela 2.2.).

Na primer, u pomenutom problemu raspoređivanja poslova (JSSP), ako se posao j prebaci na novu poziciju u rasporedu mašine, tabuem se može proglasiti vraćanje tog posla na prethodnu. Neka se posao j premešta sa pozicije p_1 na poziciju p_2. Tabuem se može proglasiti vraćanje unazad posla j sa pozicije p_2 na poziciju p_1 što se predstavlja kao uređena trojka (j, p_2, p_1). Može se primetiti da ovaj tip tabua ne ograničava pretragu u velikoj meri, ali da se ciklusi mogu pojaviti ako se j premesti u poziciju p_3 i zatim iz p_3 u p_1. Stroži tabu bi uključivao zabranu vraćanja posla j na poziciju p_1 (bez razmatranja njegove trenutne pozicije) a to se može predstaviti kao uređeni par (j, p_1). Još stroži tabu bi u potpunosti onemogućio pomeranje posla j sa svoje (nove) pozicije, i jednostavno bi se notirao kao (j).

Tabela 2.2. Primeri tabua

Vrsta problema	**Tabu**	**Značenje**
Binarni problemi (0-1)	Indeks promenljive – (i)	Zabranjuje se promena vrednosti promenljive i sa 1 na 0 i obrnuto
Sekvence poslova	Indeks posla – (j)	Zabranjuje se promena pozicije poslu j.
	Indeks posla, njegova stara i nova pozicija – (j, p_2, p_1)	Zabranjuje se premeštanje posla j sa nove pozicije p_2 na staru poziciju p_1
	Indeks posla i njegova pozicija – (j, p)	Zabranjuje se postavljanje posla j na poziciju p.
	Indeksi dva posla – (i, j)	Zabranjuje se međusobna zamena pozicija poslovima i i j
Grafovi	Indeks luka – (i)	Zabranjuje se dodavanje (oduzimanje) luka i grafu

2.2.3. Kriterijum aspiracije

Čak i kada ne postoji mogućnost nastanka ciklusa, neki jako dobri potezi mogu biti zabranjeni što može dovesti do potpune stagnacije procesa pretraživanja. Stoga je neophodno uvesti mehanizam koji će omogućiti opoziv tabu statusa nekom potezu. Taj mehanizam naziva se kriterijum aspiracije. Najjednostavniji i najčešće korišćeni kriterijum aspiracije, koji se sreće u skoro svim implementacijama tabu pretraživanja, dopušta tabu poteze kada vode u rešenje koje je bolje od do tada najboljeg rešenja (obzirom da novo rešenje do tada nije ispitivano jer je bolje od najboljeg poznatog rešenja). Dosta komplikovaniji kriterijum aspiracije je predložen i uspešno implementiran u [9] i [29], ali se retko koristi. Ključno pravilo je da ukoliko ne može doći do formiranja ciklusa, tabui se mogu zanemariti.

Tabui zajedno sa kriterijumom aspiracije menjaju okolinu $N(s)$ tekućeg rešenja s. Skup svih rešenja iz okoline $N(s)$ koja nisu tabu zajedno sa rešenjima koja jesu tabu ali zadovoljavaju kriterijum aspiracije, naziva se skup dozvoljenih rešenja i označava se sa $\widetilde{N}(s)$. Rešenja koja se razmatraju tokom pretrage su ona iz skupa $\widetilde{N}(s)$. Na taj način okolina postaje dinamička i zavisi od prethodnog procesa pretrage zbog čega se tabu pretraživanje može posmatrati kao metoda sa dinamičkim okolinama.

2.2.4. Pretraživanje i memorija

Tabu pretraživanje sastoji se od dve osnovne komponente, a to su: (1) odgovorna pretraga i (2) adaptivna memorija.

Značaj odgovorne pretrage proizilazi iz pretpostavke da loš strateški izbor rešenja daje više informacija nego dobar slučajan izbor. U sistemu koji koristi memoriju, loš strateški izbor može da proizvede korisne informacije o tome kako je potrebno menjati strategiju. Odgovorna pretraga sadrži i elemente inteligentne pretrage jer se istraživanjem dobrih rešenja dobijaju informacije koje vode pretragu u obećavajuće regione pretraživačkog prostora.

Adaptivna memorija, informacijama koje pruža, omogućuje implementaciju procedure pretraživanja tako da to pretraživanje bude ekonomično i efikasno. Pored determinističkog pretraživanja vođenog informacijama iz memorije, TS algoritam može koristiti pretraživanje realizovano i kao procedura zasnovana na (polu)slučajnim procesima. Primer takvih procedura je procedura "simuliranog kaljenja" inspirisana procesom iz prirode.

Memorija tabu pretraživanja ima 4 dimenzije: dimenzija bliske prošlosti (*Recency*), dimenzija učestalost (*Frequency*), dimenzija kvaliteta (*Quality*) i dimenzija uticaja (*Influence*). Dimenzije bliske prošlosti i učestalosti odnose se na svojstva memorije da čuva više različitih vrsta informacija. Informacije iz bliske prošlosti odnose se na podatke o delu procesa pretrage koji se odigrao neposredno pre tekućeg trenutka, dok se učestale informacije odnose na podatke o celoj pretrazi. Dimenzija kvaliteta u vezi je sa sposobnošću da se uočavaju dobre osobine rešenja posećenih tokom pretrage. Memorija uočava elemente koji su zajednički za većinu dobrih rešenja ili puteve koji vode do njih. Ovi podaci su od velikog značaja za navođenje pretrage ka dobrim rešenjima, kao i odvraćanje od loših. Četvrta dimezija, uticaj, tiče se uticaja napravljenih izbora tokom pretrage ne samo na kvalitet već i na strukturu rešenja.

Slika 2.1. Četiri dimenzije memorije

Memorija tabu pretraživanja može biti eksplicitna ili atributna. Eksplictna memorija čuva kompletna rešenja. Obično su to elitna rešenja posećena tokom pretrage. Eksplicitna memorija može da se proširi tako da čuva atraktivne ali neistražene "susede" elitnih rešenja. Sačuvana elitna rešenja (ili njihovi susedi) koriste se da prošire lokalnu pretragu. Alternativa eksplicitnoj memoriji je atributna memorija i ona čuva informacije o atributima (elementima) rešenja koji su se menjali prilikom prelaska sa jednog rešenja na drugo. Memorija se može podeliti i na kratkoročnu i dugoročnu memoriju.

Kratkoročna memorija (*Short-Term Memory*) sadrži informacije iz bliske prošlosti pretrage i najčešće se beleži samo fiksna i ograničena količina informacija. U kratkoročnoj memoriji se čuvaju tabui pa se stoga ova memorija naziva i tabu lista. Pošto tabui najčešće nisu cela rešenja, ova memorija je atributna memorija. Najvažniji parametar tabu liste je njena dužina. Teoretski, dužina tabu liste mogla bi biti neograničena. To ne bi bilo dobro jer bi tokom pretraživanja tabu lista stalno rasla, a samim tim i vreme potrebno za proveru tabu statusa potezima. Zbog toga se najčešće dužina tabu liste fiksira na određenu vrednost koja zavisi od problema koji se razmatra i treba je pažljivo odabrati. Ta vrednost predstavlja tabu mandat i određuje broj narednih iteracija u kojima će se tabu sadržati u tabu listi. Novi tabu se dodaje na kraj liste, a ako je lista puna pre dodavanja najpre je potrebno izbaciti najstariji tabu iz liste. Ukoliko je dužina tabu liste prevelika, tabu lista može biti suviše restriktivna i na taj način smanjiti izbor potencijalnih rešenja tokom pretrage. Suviše kratka lista može prouzrokovati da se jave ciklusi dužine veće od dužine liste. Time se gubi osnovno svojstvo tabu liste – sprečavanje generisanja ciklusa.

U definiciji okoline može učestvovati više različitih poteza što implicira postojanje različitih vrsta tabua. Zbog toga se javlja problem čuvanja različitih vrsta tabua unutar jedne liste. Rešenje koje se primenjuje je da se za svaku vrstu tabua koristi posebna tabu lista u kojoj će se čuvati samo tabui jedne vrste. Upotreba višestruke tabu liste može se ilustrovati na CPLP. Neka je okolina definisana add/drop potezima i potezima zamene. Tabui vezani za add/drop poteze treba da zabrane promenu statusa lokacije i mogu se predstaviti indeksom lokacije čiji je status promenjen. Tabui za poteze zamene su složeniji. Mogu se definisati u odnosu na lokaciju gde je postrojenje zatvoreno, u odnosu na lokaciju gde je postrojenje otvoreno, u odnosu na obe lokacije (na primer, promena statusa obe lokacije je zabranjena), ili u odnosu na određenu operaciju zamene (na primer, zabranjuje se zatvaranje postrojenja na lokaciji l_1, a otvaranje postrojenja na lokaciji l_2). U ovakvim slučajevima preporučuje se korišćenje odvojenih tabu listi za svaki tip poteza.

Standardne tabu liste se najčešće implementiraju kao liste fiksne dužine. Pokazano je, međutim, da tabui fiksne dužine ne mogu uvek sprečiti cikluse, a neki autori predlažu variranje dužine tabu liste u toku pretraživanja [22], [23], [50], [55], [56]. Drugo rešenje bi bilo slučajno generisanje tabu mandata svakog poteza u okviru nekog određenog intervala. Upotreba ovakvog pristupa zahteva drugačiju organizaciju tabua. Svaki tabu u sebi mora da sadrži informaciju do koje iteracije će biti aktivan, nasuprot prethodnoj organizaciji gde je samo članstvo u tabu listi označavalo aktivan tabu status.

Dugoročna memorija (*Long-Term Memory*) se još naziva i frekventna memorija. Ova memorija obezbeđuje informacije koje su komplementarne sa informacijama iz tabu liste. Dugoročna memorija je najčešće atributna memorija i čuva informacije o elementima (atributima) prisutnim u tekućem rešenju tokom celog procesa pretrage. Informacije koje se čuvaju predstavljaju frekvecne atributa i postoje dve mogućnosti za organizaciju ovih informacija. Dugoročna memorija može biti organizovana tako da za svaki atribut čuva broj iteracija u kojima je dati atribut bio prisutan u tekućem rešenju što predstavlja meru postojanosti tog atributa. Drugi način je da se za svaki atribut čuva broj iteracija u kojima je taj atribut menjao status, odnosto napuštao ili ulazio u rešenje što predstavlja meru tranzitnosti tog atributa (tabela 2.3.). U slučaju kada je moguće identifikovati korisne regione u prostoru pretraživanja, frekventna memorija se može unaprediti tako da računa broj iteracija u ovim različitim regionima.

Tabela 2.3. Primeri dve moguće organizacije frekventne memorije – čuvanje mere postojanosti i čuvanje mere tranzitnosti

Vrsta problema	**Mera postojanosti**	**Mera tranzitnosti**
Binarni problemi (0-1)	Broj iteracija u kojima je promenljiva i bila postavljena na 1	Broj iteracija u kojima je promenljiva i menjala vrednost
Sekvence poslova	Broj iteracija kada je posao j bio postavljen na poziciju p	Broj iteracija kada je posao j menjao poziciju sa poslom i
	Prosečna vrednost fukcije cilja rešenja u kojima je posao j na poziciji p	Broj iteracija u kojima je posao j na poziciji p premešten na poziciju p' tako da važi $p' < p$.
Grafovi	Broj iteracija kada je luk i bio uključen u tekuće rešenje	Broj iteracijacija u kojima je luk i bio isključen iz rešenja kada je luk j bio uključen
	Prosečna vrednost funkcije cilja rešenja u kojima je luk i bio uključen u tekuće rešenje	Broj iteracija u kojima je luk i bio dodat rešenju potezom sa poboljšanjem

Na primer, u slučaju CPLP dugoročna memorija se može implemenirati tako da za svaku lokaciju čuva broj iteracija od početka pretrage u kojima je u tekućem rešenju na toj lokaciji bilo postavljeno postrojenje. U slučaju raspoređivanja poslova za svaki posao može čuvati broj iteracija u kojima je taj posao bio premeštan.

Dugoročna memorija ima glavnu ulogu u implementaciji procedura diversifikacije i intenzifikacije rešenja koje su proširenje osnovnog algoritma tabu pretraživanja, o čemu će biti reči u daljem tekstu.

2.2.5. Šema osnovnog algoritma tabu pretraživanja

Neka je potrebno minimizovati funkciju $f(s)$ na nekom domenu. Navedena šema odnosi se na takozvani TS algoritam sa najvećim poboljšanjem, jer se u svakoj iteraciji primenjuje potez koji donosi najviše poboljšanja. To je i najčešće korišćena verzija tabu pretraživanja. Ova verzija TS algoritma istražuje celu dozvoljenu okolinu $\widetilde{N}(s)$ i pronalazi njen optimum u koji prelazi. Druga manje korišćena verzija ne istražuje sva rešenja iz $\widetilde{N}(s)$ već se zaustavlja kod prvog poboljšanja i naziva se TS algoritam sa prvim poboljšanjem.

Notacija (nastavak notacije iz sekcije 2.2.1.):

- $\widetilde{N}(s)$ – podskup dozvoljenih rešenja iz skupa $N(s)$, tj. rešenja koja nisu tabu ili zadovoljavaju kriterijum aspiracije,
- T – tabu lista.

Šema 2.2. Tabu pretraživanje:

```
Inicijalizacija:
    1. s_0 = KonstrukcijaPočetnogRešenja();
    2. s = s_0, s* = s_0, f* = f(s_0), T = ∅;
Pretraga:
    3. while(!KriterijumZaustavljanja())
    4.     s' = IzborNajboljegRešenjaUOkolini(Ñ(s));
    5.     s = s';
    6.     if(f(s) < f*)
    7.         f* = f(s);
    8.         s* = s;
           endif
    9.     AžuriranjeTabuListe(T);
       endwhile
   10. return s*;
```

2.2.6. Kriterijum zaustavljanja

Teorijski, pretraživanje bi se moglo vršiti beskonačno, osim ukoliko je optimalno rešenje posmatranog problema unapred poznato. U praksi, pretraživanje se mora zaustaviti u nekom trenutku. Najčešće korišćeni kriterijumi zaustavljanja u tabu pretraživanju su:

- nakon određenog broja iteracija (ili određenog CPU vremena),
- nakon isteka zadatog vremena,
- nakon nekog broja uzastopnih iteracija bez poboljšanja vrednosti funkcije cilja (kriterijum korišćen u najvećem broju implementacija),
- kada vrednost funkcije cilja dostigne unapred određenu vrednost.

Kod kompleksnih TS algoritama koji se sastoje iz više faza, pretraga staje kada se završe sve faze, a trajanje svake faze određuje se nekim od navedenih kriterijuma.

2.2.7. Slučajno tabu pretraživanje

Klasični TS algoritam tokom pretrage mora da vrši izračunavanje funkcije cilja za svako rešenje iz okoline $N(S)$ tekućeg rešenja S, što može biti jako skupo ukoliko je skup $N(S)$ veliki. Kod slučajnog tabu pretraživanja razmatra se slučajni uzorak $N'(S)$ skupa $N(S)$ što značajno umanjuje cenu (vreme) izračunavanja. Druga pogodnost ovog pristupa je da dodata slučajnost u toku pretrage deluje protiv stvaranja ciklusa. To omogućuje korišćenje kraćih tabu listi nego u slučaju ispitivanja cele okoline. Negativna strana se ogleda u tome što je moguće zaobići veoma dobra rešenja. Takođe, moguće je uneti slučajnost i u kriterijumu aspiracije.

2.3. Proširenja osnovnih koncepata

Jednostavno tabu pretraživanje može ponekad uspešno rešiti teške problem, ali u većini slučajeva, dodatni elementi se moraju uneti u strategiju pretrage kako bi se postigla potpuna efikasnost.

2.3.1. Intenzifikacija

Intenzifikacija se bazira na menjanju pravila izbora rešenja da bi se omogućio izbor elemenata rešenja koji su se pokazali dobri kroz proces pretrage. Intenzifikacija takođe može inicirati da se proces pretrage vrati u atraktivne regione radi njihovog temeljnijeg pretraživanja. Ideja koncepta intenzifikacije pretraživanja je da treba podrobnije istražiti delove prostora pretraživanja koji se čine pogodnim kako bi se osigurao pronalazak najboljih rešenja u tom prostoru.

U opštem slučaju, intenzifikacija se zasniva na nekoj vrsti srednjeročne memorije u kojoj se za svaki element rešenja i svaku poziciju čuva maksimalan broj uzastopnih iteracija takvih da je taj element bio prisutan u tekućem rešenju na datoj poziciji. Na primer, u CPLP, za svaku lokaciju se može čuvati informacija o tome koliko dugo je ta lokacija imala otvoreno postrojenje, tj. za svaku lokaciju čuva se maksimalan broj uzastopnih iteracija takvih da je na toj lokaciji bilo otvoreno postrojenje. Čest pristup je da se umesto uvođenja nove memorije intenzifikacija realizuje na osnovu informacija iz već postojećih memorija, a pre svega dugoročne memorije. Iz dugoročne memorije kod CPLP-a može se videti koliko često je na svakoj lokaciji bilo otvoreno postrojenje i na osnovu tih informacija preduzeti određene akcije.

Tipičan pristup intenzifikacije je ponovni početak pretraživanja od do tada najboljeg poznatog rešenja pri čemu se fiksiraju elementi koji se čine dobrim. Može se fiksirati određeni broj postrojenja u lokacijama koja su često birana u prethodnim iteracijama i vršiti ograničeno pretraživanje na drugim lokacijama. Još jedna često korišćena tehnika se sastoji iz promena okolina što obezbeđuje bolje i raznovrsnije poteze. U CPLP, ako se koriste potezi dodavanja/odbacivanja, potezi zamene se mogu dodati okolini. U probabilističkom tabu pretraživanju, veličina uzorka se može povećati ili se može preći na pretraživanje celih okolina. Intenzifikacija se koristi u mnogim implementacijama tabu pretraživanja, iako nije uvek neophodna. To je zato što postoje mnoge situacije gde je normalan proces pretraživanja dovoljno temeljan.

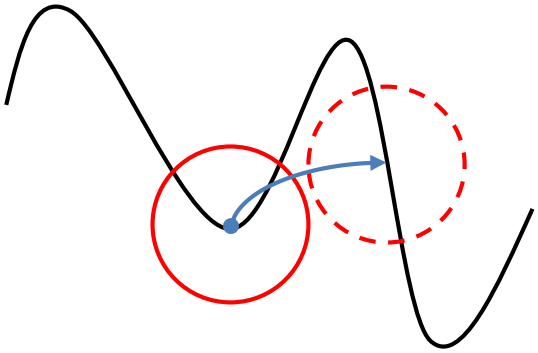
Drugi način implementacije procedure intenzifikacije je na osnovu skupa elitnih rešenja. Članstvo u skupu elitnih rešenja najčešće se određuje na osnovu unapred zadate vrednosti funkcije

cilja i ta vrednost zavisi od vrednosti funkcije cilja najboljeg rešenja. Na osnovu sačuvanih elitnih rešenja tokom pretrage jedan od načina realizacije intenzifikacije je da se u određenim trenucima pretraga premesti u neko elitno rešenje radi detaljnije pretrage te regije pretraživačkog prostora.

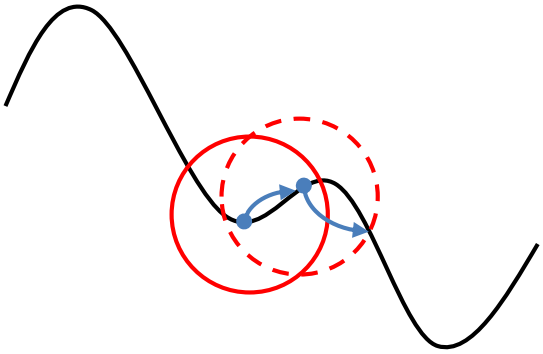
2.3.2. Diversifikacija

Jedan od glavnih problema svih metoda zasnovanih na lokalnom pretraživanju je taj da postoji tendencija da budu previše "lokalne" (kao što samo ime ukazuje). I pored korisnog efekta tabua, ovaj problem se javlja i kod tabu pretraživanja. Većina vremena, ako ne i sve, troši se u ograničenom delu prostora pretraživanja. Negativna posledica toga je da, iako se mogu dobiti dobra rešenja, najzanimljiviji delovi prostora pretraživanja mogu ostati neistraženi i time se dobiti rešenja koja su daleko od optimalnih. Diversifikacija je mehanizam koji pokušava umanjiti ovaj problem navođenjem pretraživanja u prethodno neispitane delove prostora pretraživanja. Najčešće se zasniva na nekom obliku dugoročne memorije pretraživanja, kao što je frekventna memorija, u kojoj se za svaki atribut čuva ukupan broj iteracija od početka pretraživanja tako da je taj atribut bio prisutan u tekućem rešenju.

Postoje dve najznačajnije tehnike diversifikacije. Prva, koja se naziva diversifikacija restarta (slika 2.2.), uvodi nekoliko retko korišćenih elemenata u postojeće rešenje (ili najbolje do tada poznato rešenje) i restartuje pretraživanje iz te tačke. U CPLP, ovim se može otvoriti jedno ili više postrojenja u lokacijama koje se retko koriste do određenog trenutka u procesu pretrage i nastavlja pretraživanje iz te konfiguracije postrojenja (takođe se mogu zatvoriti postrojenja u lokacijama koje se najčešće koriste). U problemu raspoređivanja poslova, posao koji nije zauzeo određenu poziciju u sekvenci poslova za određenu mašinu može se prebaciti na tu poziciju.



Slika 2.2. Diversifikacija restarta



Slika 2.3. Kontinualna diversifikacija

Drugi metod diversifikacije, koji se naziva kontinualna diversifikacija (slika 2.3.), integriše rezultate diversifikacije direktno u regularni proces pretraživanja. To se postiže promenom funkcije cilja (detaljnije o modifikaciji funkcije cilja u sekciji 2.3.3.) tako da se na vrednost funkcije cilja rešenja dodaje vrednost koja je povezana sa frekvencijom elemenata rešenja. Jedan od načina računanja nove (penalizovane) vrednosti funkcije cilja jeste računanje po formuli:

$$f' = f + d * p,$$

gde je p vrednost penala i predstavlja funkciju frekvencija elemenata rešenja. Ta vrednost je manja (ukoliko se razmatra minimizacija funkcije) kod onih rešenja čiji elementi imaju manju frekvenciju. Na taj način se obezbeđuje da će pretraga razmatrati rešenja koja sadrže retko korišćene elemente. Ta rešenja ne moraju biti dobra, međutim zbog dodavanja određene vrednosti na funkciju cilja, u krajnjem poređenju ona mogu biti bolja od nekih jako dobrih rešenja, što će proces pretrage odvesti

u neistražene regione pretraživačkog prostora. Vrednost d je parametar diversifikacije. Veća vrednost parametra d odgovara većoj diversifikaciji rešenja.

Više diskusije o navedenim metodama diversifikacije može se naći u [53]. Treći način ostvarivanja diversifikacije je strateško oscilovanje (sekcija 2.3.4.).

2.3.3. Modifikovana funkcija cilja

Postoji vise problema za koje je izračunavanje funkcije cilja preskupo. Tipičan primer je CPLP kada se pretražuje prostor sastavljen od binarnih vektora čiji elementi određuju na kojim lokacijama su postavljena postrojenja. U tom slučaju izračunavanje vrednosti funkcije cilja za svako potencijalno rešenje podrazumeva rešavanje pridruženog transportnog problema. Efikasan način za rešavanje ovog problema je ispitivanje okoline koristeći izmenjenu funkciju cilja. Za izmenjenu funkciju cilja najčešće se uzima funkcija koja je povezana sa pravom funkcijom cilja, ali je manje računski zahtevna. Na osnovu izmenjene funkcije cilja identifikuje se mali skup obećavajućih kandidata (potencijalnih rešenja koja daju najbolje vrednosti za izmenjenu funkciju cilja). Prava funkcija cilja se zatim računa na osnovu malog skupa potencijalnih poteza i najbolje rešenje postaje novo trenutno rešenje (primer ovog pristupa se može naći u [4]).

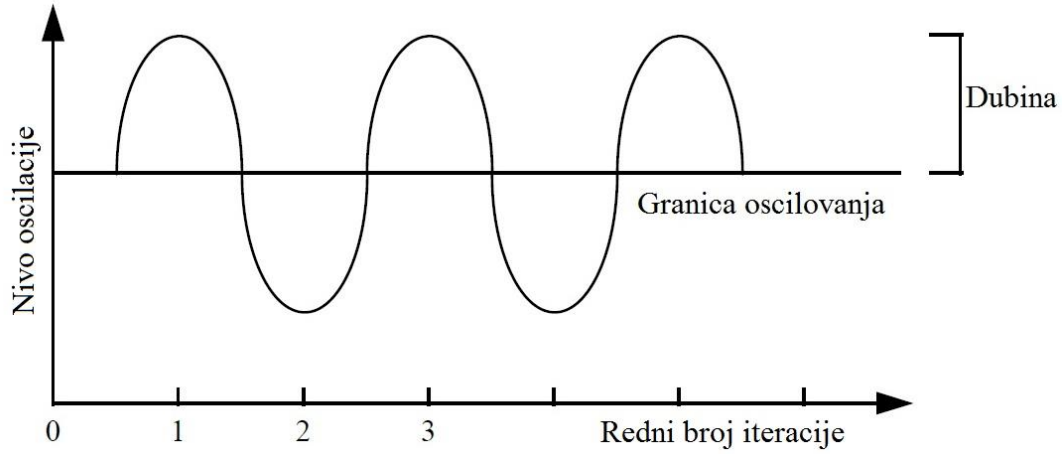
Još jedna otežavajuća okolnost koja se često javlja je što funkcija cilja ne može pružiti dovoljno informacija za efikasno usmeravanje pretraživanja ka interesantnijim delovima prostora pretraživanja. Tipičan primer ovakve situacije je slučaj kada su fiksni troškovi za otvorena postrojenja u CPLP mnogo viši od troškova pridruženog transportnog problema. U tom slučaju, preporučuje se otvaranje što manjeg broja postrojenja. Stoga je važno definisati pomoćnu funkciju cilja kako bi se usmeravalo pretraživanje u različitim smerovima. Tako se može koristiti funkcija cilja koja bi favorizovala ona rešenja koja za isti broj otvorenih postrojenja imaju veći broj onih sa malim protokom (malim zahtevima korisnika), pri tom povećavajući verovatnoću njihovog zatvaranja u narednim iteracijama. Razvoj efikasne pomoćne funkcije cilja nije uvek jednostavan i može zahtevati dugo ispitivanje i pojavu grešaka. U nekim drugim slučajevima, pomoćna funkcija cilja je očigledna [17].

2.3.4. Dozvoljavanje nedopustivih rešenja i strateško oscilovanje

Uzimanje u obzir svih ograničenja problema pri definisanju prostora pretraživanja često suviše ograničava proces pretraživanja i može dovesti do osrednjih rešenja. U tim slučajevima, relaksacija ograničenja je pogodna strategija, obzirom da proširuje prostor pretraživanja koji se može istražiti jednostavnijim okolinama. Relaksacija ograničenja se jednostavno implementira odbacivanjem izabranih ograničenja pri definisanju prostora pretraživanja i dodavanjem penala na funkciju cilja za narušavanje ograničenja. U CPLP, to se može postići dozvoljavanjem rešenja gde zahtevi korisnika prevazilaze kapacitet jednog ili više postrojenja. To, međutim, usložnjava pronalaženje odgovarajućih penalizujućih parametara za narušavanje ograničenja. Zanimljiv način za prevazilaženje ovog problema je samopodešavanje penala, na primer, parametri penalizacije se dinamički podešavaju na osnovu podataka iz skorijeg pretraživanja. Parametri se povećavaju ako su sva rešenja u poslednjih nekoliko iteracija nedopustiva i smanjuju ako su sva skorašnja rešenja dopustiva (detaljnije u [16]). Razmatranje nedopustivih rešenja može biti jako korisno jer iz nedopustivog rešenja često pretraga može doći u jako dobra dopustiva rešenja. Penali se takođe mogu sistematično modifikovati tako da usmere pretraživanje preko granica dopustivosti prostora pretraživanja čime se podstiče diversifikacija. Ova tehnika, poznata kao strateško oscilovanje,

uvedena je 1977. u [19] i od tada se koristi u implementacijama uspešnih procedura tabu pretraživanja.

U opštem slučaju, strateško oscilovanje je metoda koja upravlja potezima i vodi pretragu u odnosu na određenu kritičnu granicu koja se naziva granica oscilovanja. Najčešće je granica oscilovanja upravo granica dopustivosti. Kritična granica je mesto gde bi proces pretrage u uobičajenim okolnostima stao. Umesto toga, kada se dosegne kritična granica, pravila za izbor poteza se modifikuju tako da se ta granica pređe. Pretraga se dalje nastavlja do određene "dubine" sa druge strane granice. Dubina može biti zadata na različite načine, na primer: broj koraka (iteracija) nakon prelaska granice oscilovanja, ili maksimalna vrednost penala nedopustivosti rešenja. Pretraga se zatim zaokreće ka granici oscilovanja, ponovo prelazi granicu oscilovanja, ali sa druge strane, i nastavlja dalje sve do nove tačke zaokreta. Pretraga naizmenično prelazi granicu oscilovanja, zbog čega je ova metoda i dobila ime. Vraćanje pretrage istim putem prilikom oscilovanja nije moguće zbog karakteristike tabu pretraživanja da izbegava cikluse. Nivo oscilacije (slika 2.4.) predstavlja stepen nedopustivosti (dopustivosti) i odnosi se na to koliko je pretraga otišla daleko preko granice oscilovanja.



Slika 2.4. Strateško oscilovanje

Jednostavan primer strateškog oscilovanja može se ilustrovati na CPLP, ali i drugim problemima čije se rešenje može predstaviti kao binarni vektor. Kod CPLP postavljanje binarih promenljivih sa 0 na 1 (tj. potez dodavanja) u tekuće rešenje pomera tekuće rešenje ka granici dopustivosti. Proces se može nastaviti kroz nedopustiva rešenja primenom poteza dodavanja. Posle određenog broja koraka, smer kretanja pretrage se okreće primenom poteza odbacivanja, tj. postavljanja promenljivih sa 1 na 0.

2.3.5. Tabu pretraživanje i strategija liste kandidata

U situacijama kada je skup dozvoljenih rešenja $\widetilde{N}(S)$ veoma veliki ili je skupo odrediti njegove elemente, strategija liste kandidata ima suštinsku ulogu u smanjivanju broja elemenata skupa $\widetilde{N}(S)$. U mnogim slučajevima TS algoritam se koristi da upravlja procesom pretrage prilikom rešavanja teških problema koji u sebi sadrže dodatni podproblem (ili čak više podproblema). Tada je prilikom pretraživanja rešenja potrebno rešavati i odgovarajući podproblem, što može da bude veoma skupo. Zbog toga je neophodno naći efikasnu metodu za izdvajanje delova okolina koji sadrže rešenja sa željenim osobinama i njihovo detaljnije ispitivanje. Takva je strategija liste kandidata koja kvalitetna rešenja ili poteze smešta u listu radi kasnijeg detaljnog ispitivanja.

Upotreba liste kandidata kod velikih problema može značajno poboljšati rezultate osnovnog algoritma. Ova metoda utiče ne samo na brzinu pretrage, već i na kvalitet rešenja.

Pre opisa strategije liste kandidata, koja je posebno korisna u tabu pretraživanju, važno je napomenuti da efikasnost implementacije ove strategije može biti poboljšana uvođenjem relativno jednostavne memorijske komponente. Pravilnom koordinacijom sa ovom memorijom može se redukovati složenost pronalaženja najboljeg (ili skoro najboljeg) poteza. Na primer, kod problema raspoređivanja poslova, prilikom poteza zamene pozicija između dva posla, cena tog poteza može se izračunati bez kompletnog izračunavanja funkcije cilja (cena poteza može se odrediti u odnosu na poboljšanje vrednosti funkcije cilja – što je poboljšanje manje potez je skuplji).

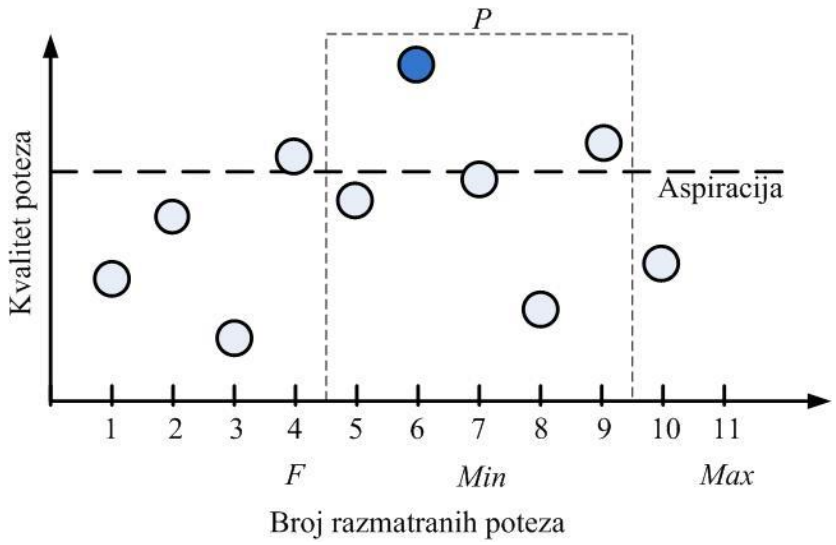
Postoje različite vrste strategija liste kandidata, a u radu su predstavljene (1) strategija P aspiracije i (2) lista elitnih kandidata.

Strategija P aspiracije (*Aspiration Plus Strategy*) obezbeđuje graničnu vrednost za kvalitet poteza na osnovu istorije pretraživanja. Ova procedura izračunava poteze (odnosno njihov kvalitet – cenu) sve do pronalaženja poteza koji zadovoljava zadatu graničnu vrednost. Nakon tog trenutka, izračunava se još dodatnih P poteza i najbolji od svih poteza se bira za izvršavanje.

Da bi se obezbedilo da se ne razmatra ni premalo ni premnogo poteza uvode se vrednosti Min i Max koje obezbeđuju da broj razmatranih poteza bude između njih. Interpretacija ovih vrednosti je sledeća. Neka je sa F (*First*) označen redni broj poteza kada je prvi put zadovoljena granična vrednost. Tada, ako vrednosti Min i Max nisu navedene, ukupan broj razmatranih poteza biće $F + P$. U suprotnom, neka su Min i Max navedene. Ako je tada $F + P < Min$ tada se vrši izračunavanje Min poteza. Ako je $F + P > Max$ tada se izračunava Max poteza. Ovi uslovi nameću sledeću činjenicu. Ako je razmatrano $Max - P$ poteza i nije pronađen ni jedan koji zadovoljava graničnu vrednost, tada F postaje jednako sa $Max - P$.

Ova strategija predstavljena je na slici 2.5. U primeru koji je na slici četvrti potez zadovoljava graničnu vrednost aspiracije pa je zbog toga $F = 4$. Vredost P je postavljena na 5, pa se zato ukupno izračunava 9 poteza i od svih bira se najbolji potez. Vrednost parametra Min postavljena je na 7, pa će najmanje 7 poteza biti razmatrano čak iako je F mala vrednost takva da $F + P < 7$. (U ovom slučaju parametar Min nije mnogo restriktivan jer će on biti razmatran samo ako je $F < 2$). Slično, vrednost parametra Max postavljena je na 11, što znači da će najviše 11 poteza biti razmatrano čak iako je F dosta veliko i zadovoljava $F + P > 11$. (Ovde je parametar Max veoma restriktivan). Na slici se vidi da je šesti razmatrani potez najkvalitetniji.

Linija aspiracije predstavlja zadatu graničnu vrednost kvaliteta poteza i može se dinamički podešavati. Na primer, ako su nekoliko prethodno razmatranih poteza bili kvalitetni i donosili napredak funkciji cilja, može se predpostaviti da će i sledeći potez biti takav pa se linija aspiracije može podići. Slično se mogu podešavati i parametri Min i Max tako da zavise od željenog broja rešenja koja prevazilaze zadatu graničnu vrednost. Za vreme sekvence manje kvalitetnih (loših) poteza koji ne donose poboljšanje u odnosu na funkciju cilja, linija aspiracije se može spustiti. Još jedan jednostavan način za podešavanje linije aspiracije je da granična vrednost zavisi od kvaliteta prvih Min razmatranih poteza.

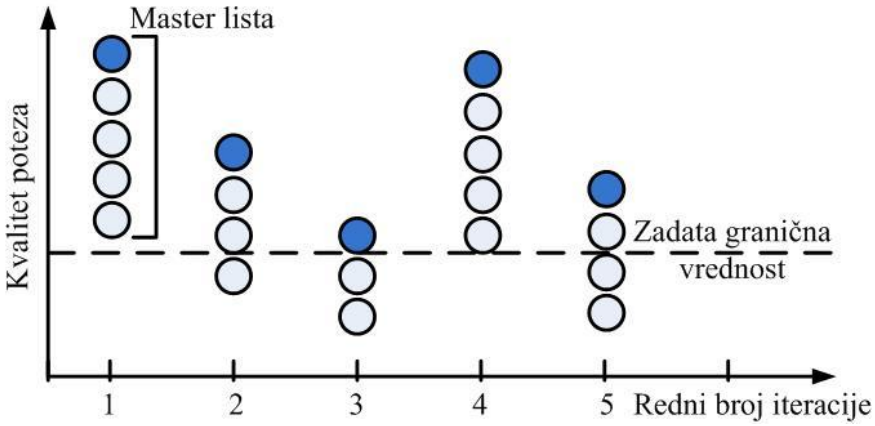


Slika 2.5. Strategija ***P*** aspiracije

Strategija P aspiracije uključuje i neke druge strategije kao specijalne slučajeve. Na primer, strategija prvog poboljšanja (*First Improving Strategy*) se dobija za $P = 0$. Linija aspiracije se podešava tako da prihvati prvi potez koji donosi poboljšanje, a parametri Min i Max se ignorišu (zapravo, radi se o klasičnoj pretrazi sa prvim poboljšanjem). Malo naprednija strategija dopušta da se vrednost P može povećavati ili smanjivati u odnosu na kvalitet inicijalnog broja poteza.

U opštem slučaju, prilikom primene strategije P aspiracije važno je obezbediti da se u svakoj iteraciji potezi koji se razmatraju razlikuju od onih prethodno razmatranih. Takođe, zbog karakteristika tabu pretraživanja ne razmatraju se inverzni potezi prethodno razmatranih poteza pa se time izbegavaju ciklusi.

Druga srategija, liste elitnih kandidata, najpre generiše master listu ispitivanjem svih (ili relativno velikog broja) poteza i izborom k najboljih, gde je k parametar metode. Zatim se u nizu narednih iteracija izvršava najbolji (sledeći) potez iz master liste sve dok je kvalitet poteza iznad unapred zadatog nivoa ili nije prekoračen zadati broj iteracija koje primenjuju poteze iz liste. Proces se nastavlja generisanjem nove master liste i izvršavanjem poteza iz nje. Ova strategija je grafički prikazana na slici 2.6.



Slika 2.6. Strategija liste elitnih kandidata

Motivacija za ovu metodu leži u pretpostavci da će dobar potez, iako nije izvršen u sadašnjosti, ostati dobar potez u budućnosti (tj. u nekom broju narednih iteracija). Posle izvršenog jednog poteza, priroda ostalih poteza može biti promenjena. Pretpostavka je da će promenjeni potezi (ili većina njih) naslediti dobre osobine od svojih prethodnika. Na slici se vidi da se u prvoj iteraciji izvršava najbolji potez. Njegovo izvršavanje iniciralo je pad kvaliteta ostalih poteza u master listi. Pretpostavlja se da su ti potezi i dalje dovoljno kvalitetni i da njihovo izvršavanje unosi dobre elemente u tekuće rešenje. Stoga se ti potezi i izvršavaju. U četvrtoj iteraciji se ponovo formira master lista jer je kvalitet preostalih poteza u prethodnoj listi pao ispod zadovoljavajućeg nivoa.

Strategija liste elitnih kandidata može biti proširena metodom *P* aspiracije dozvoljavanjem razmatranja dodatnih poteza izvan master liste. Jedan od načina da se proširenje realizuje je sledeći. Pošto se master lista generiše ispitivanjem velikog broja poteza, prvi (najbolji) potez u master listi ima visok kvalitet pa bi se u tom slučaju razmatranje dodatnih poteza izostavilo. Dodati potezi mogu se razmatrati u sledećim iteracijama, gde kvalitet rešenja dobijenog sledećim potezom može predstavljati liniju aspiracije (graničnu vrednost). Tada se može metodom *P* aspiracije pokušati sa pronalaženjem boljeg poteza, a ako to nije moguće izvršava se prvi potez u master listi.

3. Generalizovani problem pridruživanja

3.1. Uvod u probleme pridruživanja

Izraz "problem pridruživanja" (*Assignment Problem* – AP) prvi put se javlja 1952. u [60]. Međutim, ono što se smatra početkom istraživanja problema pridruživanja i razvoja metoda za njihovo rešavanje jeste objavljivanje rada [33] 1955. godine u kome je autor Harold Kuhn predstavio takozvani Mađarski algoritam za rešavanje ovog problema u polinomijalnom vremenu. Od tada do danas nastale su mnoge varijante ovog problema.

Problemi pridruživanja odnose se na pronalaženje optimalnog povezivanja elemenata dva ili više skupova, a dimenzija konkretnog problema odnosi se na broj skupova kao i njihov broj elemenata. Pošto se najčešće radi o povezivanju elemenata dva skupa jedan od njih može biti nazvan skupom zadataka, a drugi skupom agenata. Originalna verzija problema pridruživanja podrazumeva da se svaki zadatak pridružuje tačno jednom agentu, kao i da svaki agent izvršava tačno jedan zadatak (radi se o "jedan-jedan" pridruživanju). Najčešće verzije problema pridruživanja odnose se na povezivanje više zadataka sa jednim agentom, a takav je i generalizovani problem pridruživanja (*Generalized Assignment Problem* – GAP).

Generalizovani problem pridruživanja odnosi se na situaciju u kojoj je potrebno raspodeliti n poslova između m agenata ($n \geq m$) tako da troškovi pridruživanja i izvršavanja tih poslova budu minimalni. Druga varijanta ovog problema podrazumeva raspoređivanje poslova tako da profit bude maksimalan. Agenti imaju (različite) kapacitete izražene jedinicama resursa i svaki posao mora biti dodeljen tačno jednom agentu. Jednom agentu može biti dodeljen veći broj poslova pri čemu se mora voditi računa da to dodeljivanje zadovoljava kapacitete agenata, dok jedan posao može biti izvršavan od strane samo jednog agenta. Cena izvršavanja određenog posla razlikuje se od agenta do agenta. Takođe, određeni posao zauzima različitu količinu kapaciteta različitim agentima.

3.2. Matematička formulacija GAP-a

Neka je $I = \{1, 2, \dots, m\}$ skup agenata, a $J = \{1, 2, \dots, n\}$ skup poslova. GAP može biti formulisan kao problem celobrojnog programiranja na sledeći način:

$$\min z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.1.}$$

pri ograničenjima:

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i \quad \forall i \in I, \tag{3.2.}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in J, \tag{3.3.}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, \quad \forall j \in J, \tag{3.4.}$$

gde c_{ij} predstavlja cenu pridruživanja agenta i poslu j, a_{ij} predstavlja količinu resursa koju agent i troši na izvršavanje posla j i b_i predstavlja kapacitet agenta i. U nekim varijantama GAP-a količina resursa za izvršavanje posla j ista je za sve agente tako da se a_{ij} zamenjuje sa a_j. Ograničenje (3.2.)

obezbeđuje da kapaciteti agenata ne budu prekoračeni, ograničenje (3.3.) omogućuje da svaki posao bude dodeljen tačno jednom agentu dok se uslov (3.4.) odnosi na binarnu prirodu promenljivih x_{ij}, pri čemu x_{ij} uzima vrednost 1 ako je posao j dodeljen agentu i, 0 inače.

Mnogo problema iz realnog života mogu biti formulisani kao GAP. Na primer, dizajn komunikacionih mreža sa ograničenim kapacitetima u svakom čvoru, pridruživanje poslova računarima u računarskim mrežama, raspoređivanje procesa procesorima kod računara sa više procesora i sl. Takođe GAP se javlja kao podproblem u mnogim drugim problemima kao na primer kod rutiranja vozila, raspoređivanja kapaciteta, lokacijskih problema, itd. GAP je NP-težak problem [15], [48], tako da sa porastom vrednosti m i n njegovo egzaktno rešavanje postaje znatno složenije, a u pojedinim slučajevima i nemoguće. Zbog toga sve je veći značaj heurističkih algoritama za rešavanje ovog problema. Nekoliko egzaktnih algoritama za rešavanje GAP-a predstavljeno je u [37], [41], [49] dok se primene heurističkih metoda mogu naći u [14], [42], [45], [61], [63].

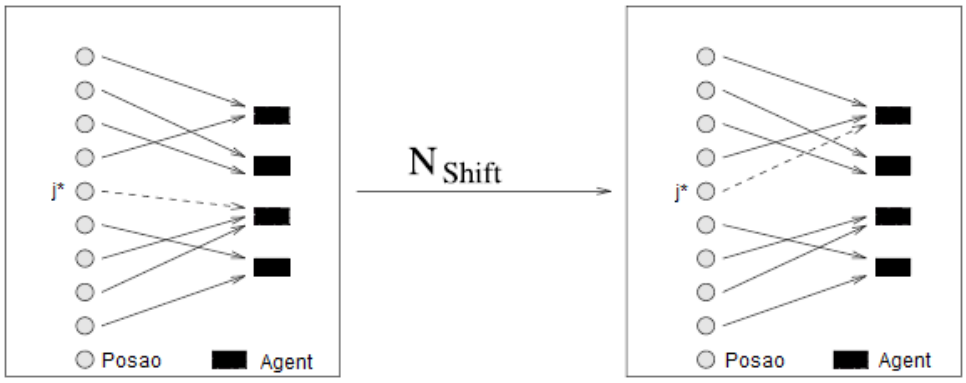
3.3. Rešavanje GAP-a algoritmom tabu pretraživanja

3.3.1. Predstavljanje rešenja i njegove okoline

Matematički gledano, rešenje GAP-a je matrica $[x_{ij}]$ dimenzije $m \times n$ čiji su elementi iz skupa $\{0,1\}$. Ako je $x_{ij} = 1$ to znači da je posao j pridružen agentu i. Ovakvo rešenje se može predstaviti i kao n-dimenzioni vektor $s = (s_1, s_2, \dots, s_n)$ gde je na j-tom mestu redni broj agenta kome je pridružen posao j, tj. $\forall s_j \in I \;\; s_j = i \Leftrightarrow x_{ij} = 1$. Za potrebe računara rešenje se predstavlja kao n-dimenzioni niz čiji su elementi iz skupa $\{0, 1, \dots, m-1\}$.

Neka je sa D označen skup svih dopustivih rešenja. Za svako rešenje s definiše se njegova okolina $N(s) \subseteq S$ kao skup svih rešenja iz S koja su na neki način "blizu" do rešenja s. Okolina nekog rešenja se može definisati na razne načine. Jedna od najčešće korišćenih okolina je okolina promene (N_{Shift} okolina) koja se još naziva i okolina 1 i definiše se na sledeći način. Neka je dato neko rešenje $s = (s_1, s_2, \dots, s_n)$, tada se pod N_{shift} okolinom (slika 3.1.) podrazumeva skup svih rešenja koja se od rešenja s razlikuju u samo jednoj koordinati, tj:

$$N_{shift}(s) = \{(s'_1, s'_2, \dots, s'_n) : \exists j^* \in J \text{ tako da } s'_{j^*} \neq s_j \text{ i } \forall j \neq j^* \;\; s'_j = s_j\}.$$



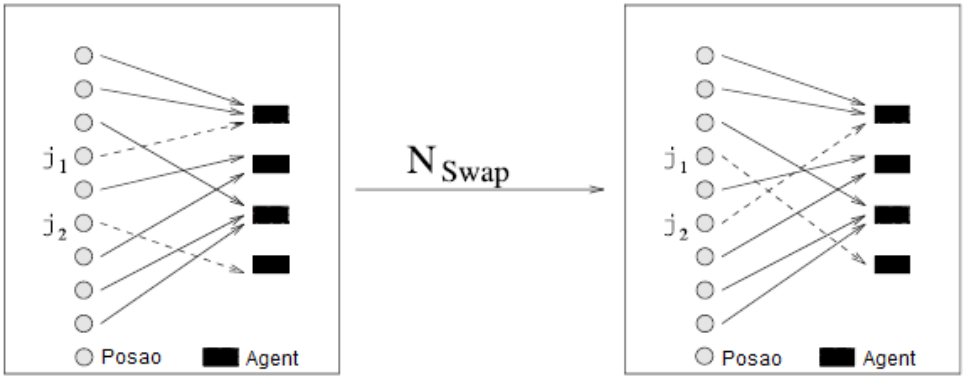
Slika 3.1. Okolina promene (N_{Shift} okolina)

Promena agenta poslu j naziva se potez promene, pa se N_{Shift} okolina rešenja s dobija primenom svih poteza promene na rešenje s. Druga veoma često korišćena okolina je okolina zamene (N_{Swap} okolina) koja se dobija tako što u rešenju s dva posla međusobno razmene dodeljene agente (slika 3.2.), što se može zapisati ovako:

$$N_{Swap}(s) = \{(s'_1, s'_2, \dots, s'_n) : \exists j_1, j_2 \in J, s_{j_1} \neq s_{j_2} \text{ tako da } s'_{j_1} = s_{j_2},\ s'_{j_2} = s_{j_1}, \forall j \neq j_1, j_2\ s'_j = s_j\}.$$

Zamena agenata između dva posla naziva se potez zamene, a N_{Swap} okolina rešenja s dobija primenom svih poteza zamene na rešenje s. Okolina $N_{Swap}(s)$ se sastoji od rešenja koja su iste strukture kao i rešenje s, tj. broj poslova pridružen svakom agentu ostaje isti.

U slučaju jednostavne okoline TS neće dati željene rezultate, zbog toga se u radu koristi okolina koja predstavlja uniju navedenih okolina, tj. $N(s) = N_{shift}(s) \cup N_{swap}(s)$. Na ovaj način dobija se na raznovrsnosti rešenja u toku pretrage.



Slika 3.2. Okolina zamene ($\boldsymbol{N_{Swap}}$ okolina)

3.3.2. Početno rešenje i RGAP

Kao i u mnogim drugim heuristikama i ovde se javlja problem početnog rešenja. Jedan od načina da se dođe do početnog rešenja je slučajno biranje elemenata niza koji predstavlja rešenje. Najčešće takvo rešenje je nedopustivo. Ako bi se ovaj postupak nastavio sve do prvog dopustivog rešenja, vreme izvršavanja programa bi znatno poraslo. Kod velikh instanci generisanje dopustivog rešenja na slučajan način bilo bi gotovo nemoguće pa samim tim bi vreme izvršavanja težilo ka beskonačnosti.

U radu se koristi sledeći pristup za generisanje početnog rešenja. Najpre, može se primetiti da su elementi matrice cena $[c_{ij}]$ i matrice kapaciteta $[a_{ij}]$ obrnuto proporcionalni, tj. kako raste cena izvršavanja posla j od strane agenta i, tako se smanjuje količina resursa koje posao j zauzima agentu i. Zato se na početku izvršavanja programa generiše matrica proizvoda $[p_{ij}]$ istih dimenzija kao i matrica $[c_{ij}]$ (odnosno $[a_{ij}]$) tako da za njene elemente važi $p_{ij} = a_{ij} \cdot c_{ij}$. Prilikom generisanja početnog rešenja razmatra se matrica $[p_{ij}]$ i j-tom poslu se dodeljuje agent k takav da je $p_{kj} = \min_{i \in I} p_{ij}$. Dakle, u koloni j matrice $[p_{ij}]$ nađe se najmanji element i redni broj vrste određuje agenta koji se pridružuje poslu j. Dodeljivanje agenata na ovaj način može da dovede do prekoračenja kapaciteta agenata, zbog toga se kod svakog dodeljivanja proverava da li ono zadovoljava kapacitet određenog agenta. U slučaju da ne zadovoljava onda se traži sledeći po

veličini element matrice $[p_{ij}]$ u koloni j i redni broj te kolone predstavlja redni broj agenta koji treba dodeliti poslu j, itd. Ovako se može desiti da se posao j ne može dodeliti ni jednom agentu zbog zauzetosti svih agenata. Tada se poslu j ipak mora dodeliti neki agent pa se u tom slučaju uzima poslednji izračunati agent.

Ovakav algoritam za generisanje početnog rešenja može proizvesti nedopustivo rešenje. Međutim, to rešenje se sastoji od elemenata koji su intuitivno "dobri", pa je pretpostavka da će se tokom pretrage vrlo brzo doći do veoma dobrog dopustivog rešenja. Zbog toga se to početno rešenje ne odbacuje, ali je zato potrebno modifikovati početni problem. Pocetni problem GAP se u radu modifikuje relaksacijom, tj. odbacivanjem ogranjičenja za kapacitete agenata i tako se dobija RGAP. Na taj način ne pravi se razlika između dopustivih i nedopustivih rešenja. Da bi se tokom pretrage izbegla nedopustiva rešenja potrebno je modifikovati i funkciju cilja što je urađeno na sledeći način:

$$z' = z + M * excess = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + M * excess.$$

U datom izrazu M predstavlja veliki pozitivan broj, dok je $excess$ ukupno prekoračenje kapaciteta svih agenata i računa se na sledeći način: $excess = \sum_{i=1}^{m} excess_i$, gde je $excess_i = max\{\sum_{j=1}^{n} a_{ij} x_{ij} - b_i, 0\}$. Ako je rešenje dopustivo onda će biti $\forall i \in I \;\; excess_i = 0$, a samim tim i $excess = 0$, pa se vrednost funkcije cilja neće menjati. Ako je rešenje nedopustivo, sva prekoračenja se dodatno naplaćuju i funkcija cilja se uvećava za tu vrednost pa takvo rešenje postaje jako skupo i vrlo verovatno neće biti razmatrano u procesu pretrage. Na taj način pretraživački prostor S postaje skup svih mogućih (dopustivih i nedopustivih) rešenja, tj. $S = \{(s_1, s_2, \dots, s_n) : s_i \in \mathbb{N}, 1 \leq s_i \leq m\}$. Eksperimentalno se pokazuje da ako je početno rešenje nedopustivo, proces pretrage vrlo brzo napušta nedopustivi prostor upravo zbog skupog naplaćivanja prekoračenja.

U opštem slucaju ideja relaksacije podrazumeva odbacivanje nekog od (veoma restriktivnih) uslova ograničenja u cilju svođenja problema na novi problem kojeg je lakše resiti. Pomenuta relaksacija GAP-a koristi se i u [13] gde se veličina M dinamički menja u toku izvršavanja programa, dok se u [61] koristi linearna relaksacija, tj. binarne promenljive se posmatraju kao realne.

3.3.3. Predloženi algoritam tabu pretraživanja

U ovom odeljku dat je opis predloženog algoritma tabu pretraživanja za rešavanje generalizovanog problema pridruživanja. Tabui su predstavljeni kao uređeni parovi (j, i) sa značenjem da je zabranjeno pridružiti posao j agentu i. Ovakvim tabuima zabranjuju se samo potezi promene. Zabranjivanje i poteza zamene deluje previše restriktivno pa se ti potezi ne zabranjuju. Takođe, zbog toga što potezi zamene ne menjaju strukturu rešenja vrlo često su ti potezi oni koji donose značajna poboljšanja Tabui se smeštaju u listu dužine L. Kriterijum aspiracije je standardni i opisan je u poglavlju o tabu pretraživanju. Dugoročna memorija se predstavlja kao matirca L dimenzije $m \times n$ tako da element na mestu (i, j) predstavlja broj iteracija u kojima je posao j bio dodeljen agentu i. Ovakva reprezentacija dugoročne memorije omogućuje njeno veoma jednostavno ažuriranje koje se svodi na prolazak kroz niz i uvećenje odgovarajućih elemenata matrice L.

Najpre se kreira početno rešenje na gore opisan način. Algoritam se sastoji iz tri osnovne faze koje se naizmenično smenjuju, a to su: (1) faza kratkoročne memorije (faza pretraživanja), (2) faza intenzifikacije, i (3) faza diversifikacije.

Algoritam kreće od početnog rešenja dobijenog na gore opisani način. Najpre se izvršava faza pretrage u kojoj se pokušava sa unapređivanjem tekućeg rešenja. Pretražuje se cela okolina tekućeg rešenja i bira se najbolje dozvoljeno rešenje koje će postati novo tekuće rešenje. Ova faza se završava kada više nije moguće unaprediti tekuće rešenje, tj. kada pretraga dostigne lokalni optimum.

Nakon faze kratkoročne memorije primenjuje se faza intenzifikacije čiji je cilj da podrobnije istraži atraktivne delove pretraživačkog prostora. Intenzifikacija je u radu implementirana kao procedura koja generiše novo rešenje od komponenti koje su bile prisutne u više od 85% ukupnog broja iteracija. Te komponente se fiksiraju što na neki način predstavlja smanjenje dimenzije problema. U nastavku rada program pokušava da nađe najbolje vrednosti za komponente koje nisu fiksirane. Eksperimentalno se pokazuje da ako se navedeni procenat smanji veliki broj komponenti postaje fiksiran, pa intenzifikacija neće dati očekivane rezultate. Slično se dešava i sa većim procentom.

Rezultat faze intenzifikacije predstavlja početno rešenje za novu fazu kratkoročne memorije. Nakon završetka faze kratkoročne memorije primenjuje se faza diversifikacije. Ova faza implementira diversifikaciju restarta. Na osnovu dugoročne memorije nalaze se komponente rešenja koje su bile najmanje prisutne u tekucem rešenju i uključuju se u sledeće rešenje. Na taj način TS odlazi iz lokalnog optimuma u neko lošije rešenje i iz tog dela pretraživačkog prostora pokušava dostići globalni optimum. Slučajno se bira k poslova i njima se dodeljuju agenti koji su tokom pretrage najmanji broj iteracija bili dodeljeni tim poslovima. Ovakva implementacija zavisi od parametra k koji treba pažljivo odabrati. Ukoliko je k mali broj, izvršavanje diversifikacije je brzo. U tom slučaju je mala raznovrsnost rešenja koja se mogu tako dobiti. Samim tim se smanjuje verovatnoća odlaska u rešenje koje će proces pretrage dovesti do globalnog optimuma. Obrnuto važi za veliko k. U radu se za k uzima vrednost jednaka četvrtini broja poslova ($^{n}/_{4}$) i do nje se došlo eksperimentalnim putem.

Posle završetka faze diversifikacije ciklus kreće ispočetka sa fazom kratkoročne memorije. Algoritam se zaustavlja ukoliko se ispuni bar jedan od sledeća dva uslova: (1) u zadatom broju uzastopnih iteracija nije bilo poboljšanja, ili (2) program je izvršio zadati broj iteracija. Najbolje posećeno rešenje smatra se optimalnim.

3.3.4. Eksperimentalni rezultati

Za testiranje predloženog algoritma korišćene su instance dostupne na http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html. Za testiranje korišćen je Asus K53SV-SX471 sa procesorom Intel Core i7 – 2630QM 2.0 GHz (up to 2.9 GHz) sa 4 jezgra i 6 GB RAM memorije. Algoritam je implementiran u programskom jeziku *C#*. Testiranje svake instance ponovljeno je 20 puta i dobijeni su rezultati prikazani u tabeli 3.1.

U prikazanoj tabeli 3.1. kolona označena sa O odnosi se na teorijske optimalne vrednosti funkcije cilja. Sa f_{best} označena je najbolja vrednost funkcije cilja od 20 vrednosti dobijenih testiranjem. t_{best} označava vreme kada algoritam prvi put dođe u najbolju vrednost koju može da dostigne, i računa se na sledeći način:

$$t_{best_p} = \sum_{i=1}^{20} t_{best_{pi}} /20,$$

gde je t_{best_p} vreme koje se odnosi na p-tu instancu, a $t_{best_{pi}}$ vreme p-te instance u i-tom izvršavanju. Sa t_{total} označeno je ukupno vreme izvršavanja programa i ono se računa na sledeći način:

$$t_{total_p} = \sum_{i=1}^{20} t_{total_{pi}} /20,$$

a oznake t_{total_p} i $t_{total_{pi}}$ imaju isto značenje kao t_{best_p} i $t_{best_{pi}}$. Vreme je izraženo u milisekundama. Kvalitet rešenja u svih 20 izvršavanja meri se relativnom greškom rešenja izraženom u procentima (*prg*), koja prikazuje odnos dobijenog rešenja i optimalnog rešenja. Procentualna greška rešenja se računa po formuli:

$$prg_p = \frac{1}{20}\sum_{i=1}^{20} rg_{pi},$$

pri čemu se sa rg_{pi} označava relativna greška u i-tom izvršavanju p-te instance. U situacijama kada je vrednost O_p poznata, rg_{pi} se izračunava na osnovu formule:

$$rg_{pi} = 100 \cdot \frac{TS_{pi} - O_p}{O_p},$$

gde je sa TS_{pi} označena vrednost funkcije cilja rešenja p-te instance u i-tom izvršavanju. Kada O_p nije poznato, rg_{pi} se izračunava na osnovu formule: $rg_{pi} = 100 \cdot \frac{TS_{pi} - NDS}{NDS}$, gde je sa NDS (najbolje do sada) označeno najbolje postignuto rešenje u svim izvršavanjima. Kvalitet rešenja ocenjuje se i sa *σ,* odnosno standardnom devijacijom relativne greške za rg_{pi} koja se dobija iz formule:

$$\sigma_p = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(rg_{pi} - prg_p)^2}.$$

Tabela 3.1. Rezultati testiranja

	m	n	O	f_{best}	t_{best}	t_{total}	σ
gap1	5	15	261	261	0.5	1.0	0.0
gap2	5	20	277	277	2.65	6.15	0.98
gap3	5	25	438	438	4.25	11.05	0.28
gap4	5	30	406	406	7.15	19.6	0.56
gap5	8	24	403	403	4.3	13.3	0.47
gap6	8	32	525	525	9.4	32.2	0.35
gap7	8	40	646	651	29.7	85.45	0.28
gap8	8	48	797	800	65.7	169.1	0.36
gap9	10	30	482	483	15.75	36.05	0.25
gap10	10	40	638	638	33.45	92.1	0.42
gap11	10	50	573	574	76.4	206.5	0.51
gap12	10	60	974	977	169.15	452.25	0.26
gapa1	5	100	1698	1698	72.45	714.5	0.0
gapa2	5	200	3235	3235	91.8	8442.95	0.0
gapa3	10	100	1360	1360	526.65	1770.2	0.04
gapd4	10	200	---	12834	27640.2	55378.95	0.18
gapd5	20	100	---	6460	2125.35	5068.85	0.25

Iz tabele 3.1. mogu se izvesti zaključci o predloženom algoritmu. Prvo što se može primetiti da je algoritam u većini slučajeva dosegao teorijski optimum. U slučajevima gde nije tako, algoritam je bio veoma blizu teorijskog optimuma. Zatim, na osnovu kolone $\boldsymbol{t_{best}}$ primećuje se da algoritam vrlo brzo dolazi do najboljeg posećenog rešenja. Razlog za to leži u efikasnosti metode za konstrukciju početnog rešenja. Ova metoda ima potencijal da za kratko vreme generiše veoma kvalitetno početno rešenje. Iz takvog rešenja u nekoliko iteracija algoritam dostiže najbolje posećeno rešenje.

Na osnovu testiranih instanci može se videti da tačnost algoritma ne opada sa porastom dimenzija problema. U pojedinim instancama (*gapa1, gapa2*) algoritam je bio veoma precizan bez obzira na njihove (relativno) velike dimenzije. Zbog toga, može se pretpostaviti da su rešenja poslednje dve instance veoma bliska teorijskom optimumu, mada se ova tvrdnja ne može dokazati.

Takođe, u nekim slučajevima (*gap2, gap4*) može se primetiti pad kvaliteta rešenja. To ukazuje na činjenicu da nedostatak raznovrsnosti rešenja tokom pretrage. Razlog za to je faza diversifikacije. U predloženom algoritmu implementirana je diversifikacija restarta. Pored pozitivnih osobina, diversifikacija restarta u pojedinim situacijama može znatno pogoršati kvalitet tekućeg rešenja, a pretpostavka je da se u navedenim instancama upravo to dogodilo. Zbog toga, za dalji rad predlaže se hibridizacija navedenog algoritma sa nekom metodom koja će povećati raznovrsnost rešenja u toku pretrage. Detaljnije o hibridizaciji u nastavku teksta (poglavlje 5.)

4. Problem rutiranja vozila sa vremenskim ograničenjima

4.1. Uvod u probleme rutiranja

Teoretska istraživanja i praktične aplikacije na polju rutiranja vozila počela su 1959. godine sa problemom otpremanja kamiona (*Truck Dispatching Problem* - TDP) koga su prvi put postavili Dantzig i Ramser u [7]. Zadatak je bio da se pronađe optimalno rutiranje flote kamiona za dostavu benzina između skladišta benzina i velikog broja benziskih stanica koje se snabdevaju iz skladišta. Koristeći metod zasnovan na linearnom programiranju, njihova izračunavanja dovela su do približno optimalnog rešenja sa četiri rute za problem sa dvanaest benziskih stanica. Kasnije je problem otpremanja kamiona preimenovan u problem rutiranja vozila (*Vehicle Routing Problem* – VRP).

Narednih 50 godina od objavljivanja rada Dantzig-a i Ramser-a, učinjen je značajan pomak na tom polju. Danas, Google Scholar pri pretraživanju reči *problem rutiranja vozila* (odnosno *Vehicle Routing Problem*) daje više od 100.000 rezultata. Izdanje *OR/MS Today* iz juna 2006. sadrži pregled od 17 proizvođača (i prodavaca) komercijalnog softvera za rutiranje vozila čiji su softverski paketi sposobni da reše instance problema sa 1000 lokacija (potrošača, korisnika…), 50 ruta i uskih dvočasovnih vremenskih ograničenja [26]. U praksi, rutiranje vozila je jedan od najvećih dostignuća operacionih istraživanja. Na primer, svakog dana 103.500 vozača UPS-a (*United Parcel Service* – Američka globalna kompanija za dostavu pošiljki) prati kompjuterski generisane rute i prenosi u proseku 15,6 miliona paketa.

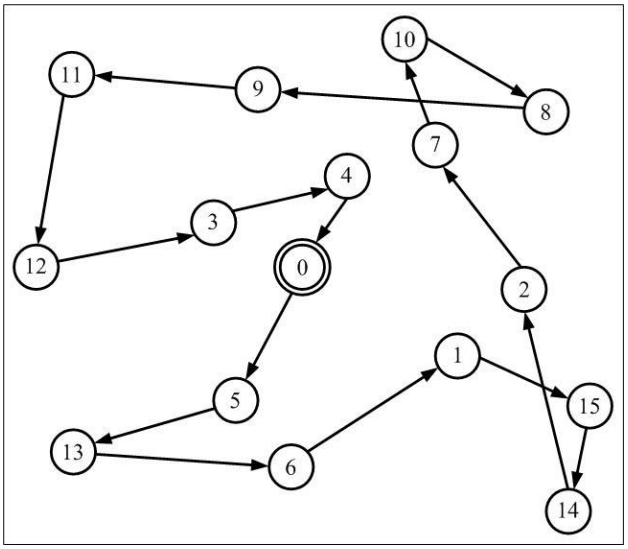
Najznačajnije studije o VRP su objavljene u periodu od 1971. (počevši od *Distribution Menagement*, autori Eilon, Watson-Gandy, Christofides) do 2002. (*The Vehicle Routing Problem*, autori Toth i Vigo). Međutim, poslednjih godina dolazi do velikog napretka i novih izazova usled tehnoloških inovacija kao što je globalni pozicioni sistem (GPS), radio frekvencijska identifikacija, itd. Skup tehnika za modelovanje i rešavanje standardnog VRP sa kapacitetima i njegovim brojnim varijantama je značajno poboljšan. Istraživači i projektanti su razvili brže, tačnije algoritme za rešavanje i bolje modele koji pružaju mogućnost rešavanja velikih instanci problema.

Transport je važan segment ljudskih aktivnosti. Neki oblik transportovanja susreće se u skoro svim čovekovim aktivnostima: telefoniranje, slanje elektronske (ili klasične) pošte, putovanje (autobusom, avionom), itd. Teretni transport je danas najvažniji oblik transportovanja jer se njime obavlja prevoz robe do korisnika. Sa porastom cene nafte na svetskom tržištu sve je veći značaj problema rutiranja vozila. Kompanije koje se bave prevozom robe efikasnim rutiranjem mogu ostvariti značajne uštede. Troškovi putovanja u SAD-u procenjani su na 45 milijardi američkih dolara na godišnjem nivou, a godišnji obrt kapitala u poslu transporta robe u Evropi procenjen je na 168 milijardi američkih dolara [31]. U Velikoj Britaniji, Francuskoj i Danskoj, na primer, cena transporta predstavlja 15%, 9% i 15% nacionalnih troškova, respektivno [31], [34]. Procenjeno je i da cena distibucije proizvoda učestvuje sa skoro 50% u ukupnoj ceni logističkih troškova, a u nekim industrijama kao što su industrije hrane i pića cena distribucije može učestvovati i sa 70% u ukupnoj ceni logističkih troškova [8]. U radu [27] pominje se da je 1989. godine od celokupnog

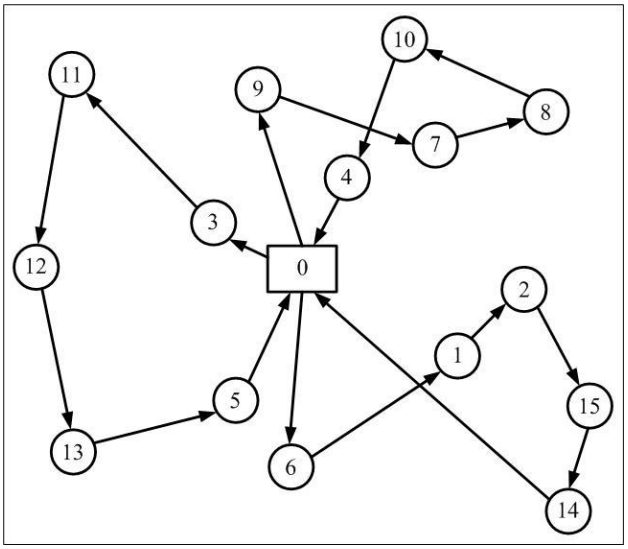
transporta robe u 76.5% slučajeva transport obavljen pomoću vozila, što dodatno ukazuje na značaj ovog problema.

4.2. Problemi rutiranja

Čuveni problem kombinatorne optimizacije jeste problem putujućeg trgovca (*Travelling Salesman Problem* - TSP). U ovom problemu zadate su lokacije koje trgovac treba da poseti, kao i polazna lokacija. Potrebno je odrediti kojim redosledom trgovac treba da obiđe lokacije tako da pređe najmanje rastojanje. Pri tom trgovac kreće iz početne lokacije, na kraju se vraća u početnu lokaciju i nije mu dozvoljeno da ostale lokacije poseti više od jednog puta. Problem je dovoljno jednostavan tako da ga mogu razumeti laici, ali i dovoljno izazovan da se može postaviti akademskoj zajednici. TSP je predstavljen na slici 4.1., gde je lokacija 0 početna lokacija.



Slika 4.1. Problem trgovačkog putnika



Slika 4.2. Problem rutiranja vozila

Uopštenje problema trgovačkog putnika je problem rutiranja vozila. Kod ovog problema data je flota vozila koja se nalaze u jednom skladištu (ili u više skladišta). Zadat je i skup lokacija (korisnika) koje vozila treba da posete radi dostavljanja robe korisnicima na tim lokacijama. Potrebno je vozilima posetiti sve lokacije tačno jednom tako da je pređeno rastojanje najmanje (često se u problemima rutiranja zahteva i minimizacija broja vozila). Pri tom, svako vozilo koje napusti skladište, na kraju svoje rute mora se vratiti u skladište. Iz postavke problema rutiranja vozila može se videti da se zapravo radi o višestrukom problemu putujućeg trgovca. Za razliku od TSP-a koji za cilj ima minimizaciju pređenog rastojanja, VRP ima dvostruki cilj: (1) minimizaciju pređenog rastojanja i (2) minimizaciju broja vozila. Između ova dva cilja postoji određena veza, međutim važno je shvatiti da se radi o prilično različitim ciljevima. Na primer, ako je potrebno minimizovati pređeno rastojanje, u određenim situacijama uključenje dodatnog vozila može uticati da se pređeno rastojanje smanji. Na slici 4.2. predstavljen je problem rutiranja vozila. Pravougaonik predstavlja skladište, dok su sa krugovima označene lokacije koje je potrebno posetiti. Na slici je prikazano rešenje koje koristi tri vozila (tj. sastoji se od tri rute).

VRP je jedan od najvažnijih problema kombinatorne optimizacije. U [58] autori su pokazali da korišćenje računarskih metoda u procesu transporta robe rezultuje uštedu od 5% do 20% cene

transporta. U [1] predstavljeno je nekoliko studija u kojima primene algoritama za rešavanje VRP-a imaju vodeću ulogu u uštedi i smanjenju cene transporta.

Mnogo problema iz realnog života mogu se predstaviti kao problemi rutiranja. To je upravo doprinelo pravoj ekspanziji varijanti ovog problema. Često razmatrana varijanta VRP-a je problem rutiranja vozila sa kapacitetima (*Capacitated Vehicle Routing Problem* – CVRP). Kod CVRP-a data je flota identičnih vozila sa ograničenim kapacitetom i dati su korisnici sa svojim zahtevima. Vozila obilaze i opslužuju zadate lokacije (korisnike) na isti način kao kod VRP-a stim što je potrebno poštovati kapacitete vozila. Cilj je (kao i kod VRP-a) da se pronađe rutiranje koje će minimizovati pređeno rastojanje. Druga poznata varijanta VRP-a je problem rutiranja vozila mešovite flote (*Mixed Fleet Vehicle Routing Problem* – MFVRP, drugi često korišćeni naziv je *Heterogeneus Fleet VRP*). Kod ovog problema flota se sastoji od više vrsta vozila. Svaka vrsta ima različiti kapacitet, a isto tako različitim vrstama vozila odgovara različita cena prelaska puta između lokacija.

Jedna od najčešće razmatranih varijanti je problem rutiranja vozila sa vremenskim ograničenjima (*Vehicle Routing Problem with Time Windows* – VRPTW). Ovaj problem se može shvatiti kao kombinacija problema rutiranja i problema raspoređivanja, što je veoma česta situacija u problemima iz realnog života. Neki od njih su rutiranje školskih autobusa, isporuka štampe, sakupljanje komunalnog otpada, pošta, itd. Kao i kod CVRP-a dato je skladište i skup korisnika na određenim lokacijama sa određenim zahtevima, kao i flota vozila istog tipa. Pretpostavka je da je dat $N+1$ korisnik $C \in \{0,1,2,\dots,N\}$, i K vozila, $V \in \{1,2,\dots,K\}$. Skladište se označava kao korisnik 0. Svaka ruta počinje iz skladišta, obilazi korisnike i na kraju se vraća u skladište. Broj ruta jednak je broju upotrebljenih vozila. Jedno vozilo je pridruženo jednoj ruti. Dvema lokacijama i i j pridružena je cena c_{ij} prelaska iz i-te u j-tu lokaciju, kao i vreme t_{ij} potrebno da se taj put pređe.

Najčešći način zadavanja instanci problema je zadavanjem koordinata $N+1$ tačke. Tada je rastojanje između i-te i j-te lokacije određeno Euklidskim rastojanjem d_{ij}. Može se smatrati da se sva vozila kreću istom brzinom, pa sva vozila prelaze isto rastojanje u zadatom vremenu. Ova pretpostavka pojednostavljuje problem jer se troškovi putovanja c_{ij}, vreme putovanja t_{ij} i rastojanje d_{ij} mogu poistovetiti.

Svaki korisnik može biti posećen tačno jednom od strane tačno jednog vozila. Svako vozilo ima isti kapacitet q_k, a svakom korisniku pridružen je različiti zahtev m_i. Suma svih zahteva korisnika jedne rute mora biti manja ili jednaka od q_k, što znači da kapacitet vozila ne sme biti prekoračen. Vremnska ograničenja dodeljena su svakom korisniku i predstavljena su vremenskim intervalima gde je dat najraniji i najkasniji trenutak dospeća u datu lokaciju. Vozilo mora da stigne u lokaciju pre najkasnijeg trenutka dospeća l_i lokacije i. Vozilo može da stigne u neku lokaciju pre najranijeg trenutka dospeća e_i lokacije i, i u tom slučaju vozilo čeka do odgovarajućeg trenutka. Svaki korisnik ima svoje servisno vreme, tj. vreme potrebno za istovar robe koju on zahteva. To je vreme koje vozilo mora da provede u određenoj lokaciji. Vozila moraju da završe svoje rute i vrate se u skladište pre zadatog vremena. U suštini to vreme predstavlja vremensko ograničenje skladišta.

Problem rutiranja vozila pripada klasi NP-teških problema [35], [38], pa samim tim isto važi i za VRPTW (ali i ostale varijante VRP-a). VRPTW je u prošlosti bio predmet intenzivnih istraživanja, kako na polju egzaktnih tako i na polju heurističkih pristupa optimizacije. Raniji pregledi metoda za rešavanje VRPTW-a mogu se naći u [10], [24], [25], [52]. U [6] i [11] autori su

se fokusirali na egzaktne metode. Detaljnije egzaktne metode razmatrane su u [5], [34]. Zbog svoje složenosti, ali i velike primene u realnom životu, VRPTW zahteva rešavanje tehnikama sposobnim da proizvedu kvalitetna rešenja u ograničenom vremenskom roku. Takve tehnike su heurističke. Heurističke metode za rešavanje VRPTW-a mogu biti podeljene u sledeće kategorije:

- konstruktivne heuristike,
- heuristike poboljšanja,
- metaheuristike.

Konstrukcione heuristike su sekvencijalni ili paralelni algoritmi koji imaju za cilj generisanje inicijalnog rešenja koje se dalje može poboljšavati heuristikama poboljšanja ili metaheuristikama. Sekvencijalni algoritimi grenerišu rutu za svako vozilo, jedno za drugim. Ovi algoritmi u sebi sadrže funkciju selekcije kojom se određuje naredni korisnik koji će biti umetnut u rutu, ruta u kojoj će izabrani korisnik biti umetnut, kao i njegova pozicija u toj ruti. Paralelni algoritmi generišu rute za sva vozila odjednom, pri čemu se pre same konstrukcije određuje broj ruta. Poslednjih godina mnogo autora je predstavilo nove heurističke pristupe, posebno metaheurističke, za rešavanje VRPTW-a. Detaljan prikaz klasičnih heuristika za konstrukciju i poboljšanje ruta dat je u [2]. Značaj ovih heuristika jeste u tome što sa malim računarskim zahtevima daju relativno kvalitetno rešenje. Isto tako, ove heuristike su glavne komponente svih metaheuristika za rešavanje VRPTW. Detaljan prikaz metaheuristika kao i njihovo međusobno poređenje dato je u [3].

4.3. Matematička formulacija VRPTW-a

Postoje tri vrste promenljivih odlučivanja kod ovog problema. Binarne promenljive x_{ijk} $(i, j \in \{0,1, \dots, N\}; k \in \{1,2, \dots, K\}; i \neq j)$ određuju strukturu ruta, tj. $x_{ijk} = 1$ ako vozilo k putuje iz lokacije i u lokaciju j. Promenljive t_i određuju vreme dolaska u lokaciju i, a promenljive w_i se odnose na vreme čekanja u lokaciji i. Cilj je odrediti rute tako da su zadovoljeni svi uslovi ograničenja a da ukupna cena putovanja bude minimalna. Problem se matematički može formulisati na sledeći način.

Promenljive odlučivanja:

x_{ijk} $x_{ijk} = 1$ ako vozilo k posle korisnika i posećuje korisnika j, 0 inače;
$x_{ijk} \in \{0,1\},\ i \neq j,\ i,j \in \{0,1,2, \dots, N\}$.

t_i vreme dolaska u i-tu lokaciju, $t_i \in \mathbb{R}, t_i \geq 0$.

w_i vreme čekanja u i-toj lokaciji, $w_i \in \mathbb{R}, w_i \geq 0$.

Parametri:

K ukupan broj vozila.

N ukupan broj korisnika.

C_i i-ti korisnik, $i \in \{1,2, \dots, N\}$.

C_0 centralno skladište.

d_{ij} Euklidsko rastojanje između lokacije i i lokacije j.

c_{ij} cena prelaska puta između lokacije i i lokacije j.

t_{ij} vreme potrebno da se pređe rastojanje između lokacije i i lokacije j.

m_i zahtev korisnika na i-toj lokaciji.
q_k kapacitet k-tog vozila.
e_i najranije vreme dospeća u i-tu lokaciju.
l_i najkasnije vreme dospeća u i-tu lokaciju.
f_i vreme opsluživanja i-tog korisnika.
r_k maksimalno dozvoljeno vreme putovanja za k-to vozilo.

$$min\, z = \sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} \sum_{k=1}^{K} c_{ij} x_{ijk}, \tag{4.1.}$$

pri uslovima: $$\sum_{k=1}^{K} \sum_{j=1}^{N} x_{ijk} \leq K,\ i = 0, \tag{4.2.}$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik},\ i = 0, k \in \{1,2,\dots,K\}, \tag{4.3.}$$

$$\sum_{k=1}^{K} \sum_{j=1, j\neq i}^{N} x_{ijk} = 1,\ i \in \{1,2,\dots,N\}, \tag{4.4.}$$

$$\sum_{k=1}^{K} \sum_{i=1, i\neq j}^{N} x_{ijk} = 1,\ j \in \{1,2,\dots,N\}, \tag{4.5.}$$

$$\sum_{i=0}^{N} m_i \sum_{j=0, j\neq i}^{N} x_{ijk} \leq q_k,\ k \in \{1,2\dots,K\}, \tag{4.6.}$$

$$\sum_{i=0}^{N} \sum_{j=0, j\neq i}^{N} x_{ijk}(t_{ij} + f_i + w_i) \leq r_k,\ k \in \{1,2,\dots,K\}, \tag{4.7.}$$

$$t_0 = w_0 = f_0 = 0, \tag{4.8.}$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{N} x_{ijk}(t_i + w_i + f_i + t_{ij}) = t_j,\ j \in \{1,2,\dots,N\}, \tag{4.9.}$$

$$e_i \leq (t_i + w_i) \leq l_i,\ i \in \{0,1,\dots,N\}. \tag{4.10.}$$

Sa (4.1.) je označena funkcija cilja problema. Uslov (4.2.) znači da najviše K vozila napušta skladište, odnosno da je dozvoljeno postojanje najviše K ruta. Uslov (4.3.) obezbeđuje da svako vozilo polazi iz skladišta i vraća se u njega. Ograničenje da svaki korisnik mora biti posećen tačno jednom od strane tačno jednog vozila zadato je sa (4.4.) i (4.5.). Uslov (4.6.) onemogućuje da kapacitet vozila bude prekoračen. Uslov (4.7.) se odnosi na maksimalno vreme k-te rute, i ako drugačije nije naglašeno za svako k važi $r_k = l_0$. Izrazi (4.8.) – (4.10.) definišu vremenska ograničenja. Ovi izrazi u potpunosti određuju skup dopustivih rešenja za VRPTW.

4.4. Rešavanje VRPTW-a algoritmom tabu pretraživanja

4.4.1. Pretraživački prostor i okolina

Za potrebe TS algoritma neophodno je definisati pretraživački prostor kao i okolinu rešenja. Rešenje VRPTW-a je skup od m ruta koje zadovoljavaju navedene uslove ograničenja (4.2.) – (4.10.). Svaka ruta se može predstaviti nizom brojeva gde taj niz počinje i završava se sa 0, a brojevi između prvog i poslednjeg elementa niza su redni brojevi lokacija. Njihov redosled odgovara redosledu obilaska lokacija u toj ruti. Na primer, na slici 4.2. predstavljen je problem koji se sastoji od jednog skladišta i 15 korisnika. Predstavljeno je i rešenje koje se sastoji od sledećih ruta 0 – 9 – 7 – 8 – 10 – 4 – 0, 0 – 6 – 1 – 2 – 15 – 14 – 0, 0 – 3 – 11 – 12 – 13 – 5 – 0.

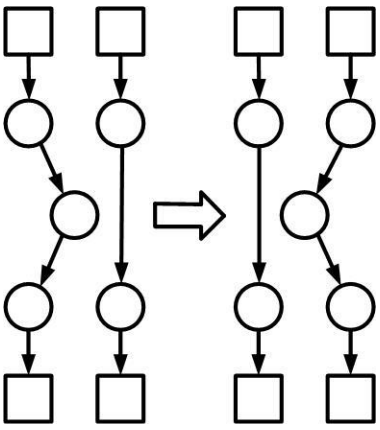
Bitna osobina predloženog algoritma tabu pretraživanja za rešavanje VRPTW-a jeste, pored pretrage u skupu dopustivih rešenja, mogućnost pretraživanja i nedopustivih rešenja. Da bi to bilo moguće neophodno je modifikovati funkciju cilja (4.1.). Modifikacijom funkcije cilja algoritam više

ne razmatra da li je rešenje dopustivo ili ne, već sva rešenja dolaze u obzir. Onda se pretraživački prostor S sastoji se svih mogućih (dopustivih i nedopustivih) rešenja. Zbog toga, svaki element pretraživačkog prostora S je skup ruta za koji važi (1) da svaka ruta počinje i završava se u skladištu i (2) da svaki korisnik pripada tačno jednoj ruti. Za svaki element (rešenje) $s \in S$ neka je sa $c(s)$ označeno ukupno pređeno rastojanje (tj. $c(s) = z$), sa $q(s)$ označeno prekoračenje kapaciteta svih vozila, a sa $w(s)$ označeno ukupno prekoračenje svih vremenskih ograničenja. Ako rešenje s ima R ruta i ako je $x^+ = max\{x, 0\}$, onda se prekoračenje kapaciteta vozila računa po formuli $q(s) = \sum_{r=1}^{R}(Q_r - q_k)^+$, gde je Q_r ukupni zahtev svih korisnika r-te rute. Ukupno prekoračenje svih vremenskih ograničenja se računa po formuli $w(s) = \sum_{i=0}^{N}(t_i - l_i)^+$. Modifikovana funkcija cilja rešenja s računa se na sledeći način:

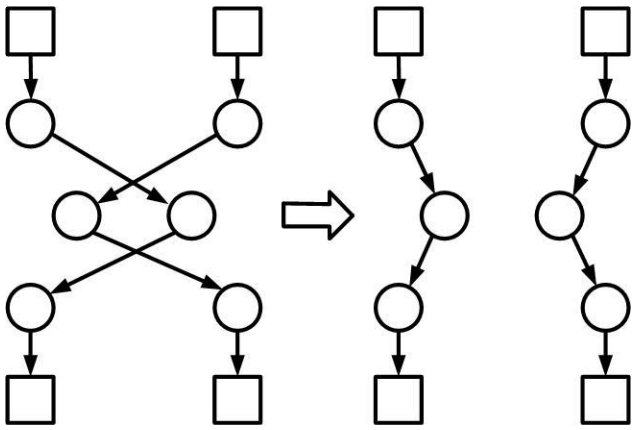
$$z'(s) = c(s) + \alpha q(s) + \beta w(s),$$

gde su α i β pozitivni parametri. Ovakvom modifikacijom funkcije cilja izvršena je relaksacija polaznog problema, i to tako što su zanemareni uslovi ograničenja (4.6.), (4.7.), (4.8.), (4.10.). Modifikovana funkcija cilja za dopustivo rešenje ima istu vrednost kao i originalna funkcija cilja. Kod nedopustivih rešenja ta nedopustivost se dodatno naplaćuje vrednošću izraza $p(s) = \alpha q(s) + \beta w(s)$. Dinamičkim podešavanjem parametara α i β pokušava se pametno vođenje pretrage kroz pretraživački prostor. Navedena relaksacija i dinamičko menjanje parametara α i β predstavlja vrstu strateškog oscilovanja.

Da bi se definisala okolina rešenja neophodno je definisati poteze nad datim rešenjem. Jedan od najjednostavnijih je potez realokacije (*realocation move*). Ovim potezom određeni korisnik napušta rutu kojoj pripada i prelazi u neku drugu rutu. Potez realokacije se grafički može predstaviti kao na slici 4.3. gde je kvadratom označeno skladište, a krugom korisnička lokacija. Sva rešenja koja se mogu dobiti primenom poteza realokacije na konkretno rešenje s čine okolinu realokacije tog rešenja u oznaci $N_r(s)$.



Slika 4.3. Potez realokacije

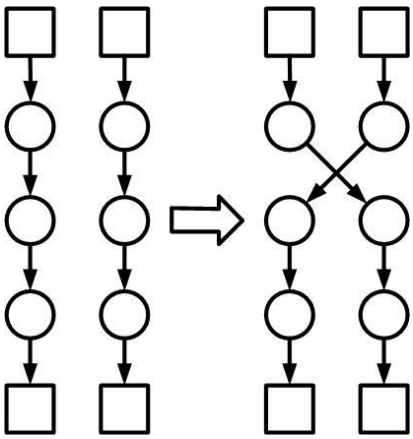


Slika 4.4. Potez zamene

Drugi često razmatrani potez je potez zamene (*exchange move*). Njime dva korisnika koji pripadaju različitim rutama međusobno menjaju rute (slika 4.4.). Potezima zamene definiše se okolina zamene koja se označava sa $N_e(s)$.

Potez koji često dovodi do poboljšanja je 2-opt* potez. Osnova ideja ovog poteza je zamena korisnika između dve rute tako što se deo prve rute počveši od nekog korisnika menja sa delom druge rute počevši od nekog korisnika. Ovaj potez predstavljen je na slici 4.5., a odgovarajuća

okolina rešenja s označava se sa $N_{2-opt^*}(s)$. Navedeni potezi i odgovarajuće okoline učestvuju u formiranju finalne okoline rešenja koja se koristi za potrebe tabu pretraživanja. Pored ovih poteza postoji mnoštvo drugih poteza a njihov detaljan prikaz kao i pregled radova u kojima su ti potezi korišćeni može se naći u [2].



Slika 4.5. Potez 2-opt*

4.4.2. Konstrukcija početnog rešenja

Za generisanje početnog rešenja u ovom radu koristi se heuristika umetanja (*Push-Forward Insertion Heuristic - PFIH*) [51]. PFHI heuristika je efikasna konstruktivna strategija koja izračunava cenu svake lokacije uzimajući u obzir njenu geografsku poziciju, njeno najkasnije vreme dospeća, kao i ugla koji ta pozicija zahvata u odnosu na skladište, i na osnovu cene lokacije generiše rute. Rute se generišu tako što se lokacije sortiraju prema izračunatoj ceni i zatim se u tom redosledu pokušava njihovo umetanje u već postojeće rute. Dakle, krene se od najjeftinije lokacije i pošto na početku nema generisanih ruta, kreira se prva ruta koja u sebi (pored skladišta) ima samo tu lokaciju. Za svaku ostalu lokaciju postupak je sledeći. Pokušava se umetanje tekuće lokacije u sve postojeće rute i bira se ona ruta tako da umetanje date lokacije u nju najmanje uvećava funkciju cilja. Ukoliko umetanje nije moguće zbog narušavanja nekih od uslova ograničenja, kreira se nova ruta koja sadrži samo tekuću lokaciju.

Redosled razmatranja lokacija u direktnoj je vezi sa kvalitetom početnog rešenja, ali isto tako i konačnog rešenja. Imajući to u vidu u Marius Solomon je u svom radu [51] došao do formule za računanje cene lokacije, koja glasi:

$$c_i = -\mu d_{0i} + \nu l_i + \eta \left[\left(\frac{p_i}{360}\right) d_{0i}\right],$$

gde je p_i ugao lokacije i u odnosu na poziciju skladista, a μ, ν i η pozitivni parametri. U istom radu empirijski se došlo do vrednosti za navedene parametre, i to $\mu = 0.7$, $\nu = 0.1$ i $\eta = 0.2$.

4.4.3. Predloženi algoritam tabu pretraživanja

U ovom odeljku biće predstavljen predloženi algoritam tabu pretraživanja za rešavanje VRPTW-a. Predloženi algoritam polazi od početnog rešenja generisanog na opisani način u prethodnom odeljku. Slično kao kod rešavanja prethodnog problema, u izvršavanju programa postoje faza kratkoročne memorije i faza intenzifikacije. U predloženom algoritmu procedura diversifikacije je kontinualna, tj. diversifikacija rešenja je uključena u proces pretrage. Svakom

rešenju $\bar{s} \in N(s)$ takvom da je $z'(\bar{s}) \geq z'(s)$ na vrednost funkcije cilja dodaje se vrednost proporcionalna frekvenciji elemenata rešenja $\bar{s}$, odnosno vrši se dodatna penalizacija rešenja $\bar{s}$. Preciznije, dugoročna memorija za svaki element rešenja obezbeđuje podatke o njegovoj učestalosti u tekućem rešenju. Neka je sa ρ_{ik} označeno koliko je puta lokacija i bila u k-toj ruti. Vrednost funkcije cilja uvećava se za $p(\bar{s}) = \lambda\sqrt{mn}\sum_{(i,k)\in B(\bar{s})}\rho_{ik}$. U navedenoj formuli sa $B(\bar{s})$ označen je skup parova (i,k) takvih da se u rešenju $\bar{s}$ u k-toj ruti nalazi i-ta lokacija. Sa n je označen je broj korisničkih lokacija, dok je sa m označen broj ruta. Izrazom $\sqrt{mn}$ vrši se korekcija vrednosti penala sa dimenzijom problema koji se razmatra. Parametar λ služi za kontrolu inteziteta diversifikacije. Ako je $z'(\bar{s}) < z'(s)$ tada je $p(\bar{s}) = 0$. Dakle, kada u okolini tekućeg rešenja postoji neko rešenje bolje od tekućeg, pretraga prelazi u najbolje takvo rešenje (jer se ta rešenja ne penalizuju). Međutim, kada je tekuće rešenje lokalni optimum, onda se vrši penalizacija svih rešenja iz okoline (pa i tekućeg rešenja) i među njima se bira najbolje. Na taj način pretraga prelazi u lošije rešenje, ali sa ciljem da se prevaziđe lokalni optimum. Pri tom, povratak u prethodno rešenje onemogućen je mehanizmom tabua. Penalizacija rešenja ima za cilj usmeravanje pretrage ka slabije istraženim delovima pretraživačkog prostora.

Faze kratkoročne memorije i intenzifikacije se naizmenično smenjuju. Faza kratkoročne memorije vrši lokalno pretraživanje koje se obavlja u okolini dobijenoj kombinovanjem navedenih okolina i to na sledeći način. Najpre se pretražije cela N_r okolina. Ukoliko je pronađeno bolje rešenje od tekućeg, onda pretraga prelazi u to rešenje. U suprotnom, istražuje se cela okolina zamene (N_e). Ako se u njoj pronađe bolje rešenje od tekućeg, pretraga prelazi u to rešenje. Ako to nije slučaj pretraga se nastavlja u okolini N_{2-opt^*}. Ovakva pretraga koristi ideju metode promenljivih okolina sa ciljem da se smanji vreme izvršavanja algoritma. Navedeni način pretrage je neka vrsta pretraživanja sa prvim poboljšanjem – pretraga prelazi u "prvo" najbolje rešenje dobijeno potpunom pretragom navedenih okolina. Eksperimentalno se pokazuje da ukoliko se vrši potpuno pretraživanje okoline $N(s) = N_r(s) \cup N_e(s) \cup N_{2-opt^*}(s)$ kvalitet rešenja nije poboljšan a vreme izvršavanja se znatno povećava. Zato se u radu koristi prethodno pomenuti način pretrage.

Faza intenzifikacije generiše novo rešenje koje će biti početno rešenje za narednu fazu kratkoročne memorije. Cilj je da se dobije rešenje koje će se sastojati iz dobrih sekvenci korisničkih lokacija. Intenzifikacija je implementirana tako da koristi ideju *PFIH* metode. U ovom slučaju redosled razmatranja lokacija određen je poslednjim tekućim rešenjem i to na sledeći način. Razmatraju se prvo sve prve lokacije u svim rutama redom počevši od prve. Jedna po jedna lokacija se dodaje novom rešenju na opisani način u odeljku 4.4.2. Kada se završi sa prvim lokacijama, prelazi se na druge lokacije u svim rutama redom počevši od prve, zatim na treće, itd. Pošto nisu sve rute iste dužine, kada se završi sa svim lokacijama neke rute ta ruta se dalje ne razmatra.

Algoritam tabu pretraživanja se zaustavlja ukoliko se ispuni bar jedan od sledeća dva uslova: (1) u zadatom broju uzastopnih faza kratkoročne memorije nije bilo poboljšanja, ili (2) program je izvršio zadati broj iteracija. Najbolje posećeno rešenje smatra se optimalnim.

4.4.4. Eksperimentalni rezultati

Za testiranje predloženog algoritma korišćene su instance kao u radu [47] koji se bavi minimizacijom iskorišćenih vozila. Instance su podeljene u dve grupe od po 30 instanci, a lokacije su određene Dekartovim koordinatama. U prvoj grupi su instance sa 25 korisničkih lokacija, dok su u drugoj grupi instance sa 50 korisničkih lokacija. Svaka grupa se sastoji od 6 skupova od po 5 instanci. Prva dva skupa su označena sa $r1$ i $r2$ i kod ovih instanci lokacije su raspoređene na slučajan način. Druga dva skupa instanci označena su sa $c1$ i $c2$ i kod njih su lokacije grupisane. Treći par skupova ($rc1$ i $rc2$) odnosi se na instance čiji je raspored lokacija kombinacija uniformno raspoređenih i grupisanih. Skupovi označeni jedinicom karakterišu se lokacijama sa uskim vremenskim intervalima, dok se skupovi označeni dvojkom sastoje od instanci čije su lokacije sa širokim vremenskim intervalima. Takođe, skup lokacija odgovarajuće instance prve grupe je podskup skupa lokacija odgovarajuće instance druge grupe, tj. instance druge grupe su proširenja instanci prve grupe.

Tabela 4.1. Rezultati testiranja instanci sa 25 korisničkih lokacija

	nv	c	f_{best}	rv	t_{best}	t_{total}	σ
r101_25	10	50	636.29	9	159	2729.50	0.00
r102_25	10	50	587.60	8	4529.55	13681.30	0.00
r103_25	10	50	538	7	8946.95	21085.40	0.01
r104_25	10	50	547.62	7	4898.95	17711.25	0.00
r105_25	10	50	597.47	8	6552.70	12786.15	0.00
r201_25	10	100	464.39	4	1503.30	17166.75	0.00
r202_25	10	100	424.45	4	6558.90	25014.60	0.00
r203_25	10	100	430.41	4	32160.15	32442.55	0.00
r204_25	10	100	411.66	4	11617.40	32358.30	0.00
r205_25	10	100	425.26	4	5060.20	23704.45	0.00
c101_25	10	50	517.68	10	362.80	6930.85	0.00
c102_25	10	50	518.71	10	10054.75	18133.45	0.00
c103_25	10	50	519.08	10	9267.65	18263.15	0.00
c104_25	10	50	519.05	10	10439.90	19703.3	0.00
c105_25	10	50	517.68	10	6016.05	13604.20	0.00
c201_25	10	75	464.48	7	1535.15	18616.05	0.00
c202_25	10	75	464.48	7	6301.40	23954.7	0.05
c203_25	10	75	464.48	7	7380.95	25681.25	0.00
c204_25	10	75	473.94	7	10387.70	29822.05	0.00
c205_25	10	75	471.50	7	4861.75	23405.95	0.00
rc101_25	10	75	728.07	8	4075.45	8854.55	0.00
rc102_25	10	75	722.55	8	1899.70	8432.50	0.00
rc103_25	10	75	722.21	8	3222.25	12383.15	0.02
rc104_25	10	75	715.07	8	7493.70	20061.95	0.00
rc105_25	10	75	722.48	8	118.95	6040.55	0.00
rc201_25	10	100	535.17	6	20679.90	28489.25	0.00
rc202_25	10	100	528.55	6	3654.20	21274.35	0.00
rc203_25	10	100	517.40	6	24763.55	32786.05	0.00
rc204_25	10	100	512.99	6	5962.30	27262.20	0.00
rc205_25	10	100	522.72	6	3333.20	20504.75	0.00

Algoritam je implementiran u programskom jeziku *C#*, a za testiranje korišćen je računar Asus K53SV-SX471 sa procesorom Intel Core i7 – 2630QM 2.0 GHz (up to 2.9 GHz) sa 4 jezgra i 6 GB RAM memorije. Testiranje svake instance ponovljeno je 20 puta. U tabeli 4.1 prikazani su rezultati testiranja prve grupe instanci, tj. instanci sa 25 lokacija, dok su u tabeli 4.2. prikazani rezultati testiranja instanci sa 50 lokacija.

Oznake f_{best}, t_{best}, t_{total}, i σ imaju isto značenje kao i u sekciji 3.3.4. koja se bavi rezultatima testiranja instanci GAP-a. Sa nv označen je broj raspoloživih vozila, a sa rv broj iskorišćenih vozila u najboljem rešenju. Kapacitet vozila označen je sa c.

Tabela 4.2. Rezultati testiranja instanci sa 50 korisničkih lokacija

	nv	c	f_{best}	rv	t_{best}	t_{total}	σ
r101_50	15	100	1046.68	12	4121.30	21088.75	0.00
r102_50	15	100	936.82	11	13831	70124.95	0.17
r103_50	15	100	863.94	10	17087.70	105730.50	0.25
r104_50	15	100	767.74	8	67275.35	147719.65	0.08
r105_50	15	100	942.53	10	1681.40	36412.20	0.00
r201_50	15	150	808.45	6	100227.40	133581.70	0.31
r202_50	15	150	766.15	7	97094.45	138764.96	0.08
r203_50	15	150	709.94	6	75859.05	151608.45	0.00
r204_50	15	150	662.70	6	62206.50	165124.95	0.00
r205_50	15	150	738.59	6	39374.95	141370.40	0.00
c101_50	15	100	589.31	9	47230	74108.55	0.00
c102_50	15	100	602.33	10	71194.25	110393.45	0.00
c103_50	15	100	613.56	9	28862.95	148558.50	0.00
c104_50	15	100	616.56	9	153235.30	159411.75	0.07
c105_50	15	100	597.10	9	48436.20	94172.65	0.00
c201_50	15	250	448.54	4	867.15	111827.45	0.00
c202_50	15	250	457.79	4	100195.70	138827.70	0.00
c203_50	15	250	481.45	4	14703.15	157631.85	0.00
c204_50	15	250	489.77	4	89092.55	191625.15	0.00
c205_50	15	250	462.04	4	83208.20	152717.10	0.00
rc101_50	15	100	1005.48	10	22969.50	43331.90	0.10
rc102_50	15	100	987.22	10	11089.40	63939.70	0.00
rc103_50	15	100	936.68	10	47289.75	102370.20	0.23
rc104_50	15	100	949.96	10	106304.70	143190.4	0.08
rc105_50	15	100	980.31	10	16858.05	67173.15	0.00
rc201_50	15	150	891.73	7	95608.60	134436.05	0.05
rc202_50	15	150	858.25	7	90958.95	141527.70	0.42
rc203_50	15	150	829.16	7	50254.45	150323.85	0.00
rc204_50	15	150	811.19	7	64863.75	174032.70	0.00
rc205_50	15	150	878.48	8	103198.20	139889.80	0.00

Na osnovu prve tabele može se videti da je preciznost algoritama veoma velika kada se radi o instancama sa 25 korisničkih lokacija. U skoro svim slučajevima standardna devijacija relativne greške (σ) bila je jednaka nuli. U slučajevima gde nije bilo tako, veličina σ je bila veoma mala, ne veća od 0.05 (*c202_25*). Razlog za to je upotreba kontinualne diversifikacije. Kontinualna diversifikacija utiče na pretragu neprestano u svim trenucima, za razliku od diversifikacije restarta. Takođe, eksperimentalno je pokazano da primenjena procedura intenzifikacije daje veoma kvalitetna rešenja, što je još jedan razlog za preciznost predloženog algoritma. Pozitivan uticaj na

proces pretrage imalo je strateško oscilovanje i modifikacija funkcije cilja (sekcija 4.4.1.). Pažljivim dinamičkim menjanjem parametara α i β omogućeno je da pretraga u određenim trenucima razmatra i nedopustiva rešenja. To je povećalo često kritikovanu slabu raznovrsnost u izboru tekućeg rešenja.

Vreme do dostizanja najboljeg rešenja (t_{best}) je zadovoljavajuće. Međutim, vreme izvršavanja kompletnog algoritma je neznatno veće od očekivanog. Razlog za to je često izračunavanje vrednosti penala prilikom diversifikacije rešenja (sekcija 4.4.3).

Iz tabele 4.2. može se videti da je tačnost algoritma opada, a vreme izvršavanja algoritma raste.Tačnost je i dalje na zadovoljavajućem nivou, jer veliki broj instanci i dalje ima standardnu devijaciju relativne greške jednaku nuli.

Upoređivanjem rezultata testiranja navedene dve grupe instanci može se videti ponašanje algoritma. Vremenski, efikasnost algoritma opada sa porastom broja korisničkih lokacija. Razlog za ovakvo ponašanje je taj što sa porastom broja lokacija veličina okoline brzo raste. Tačnost rešenja takođe opada, ali u manjoj meri. Zbog ovakvih osobina algoritma predlaže se njegova hibridizacija što je tema narednog poglavlja.

5. Hibridizacija

Proteklih godina postalo je jasno da primena samih metaheuristika na optimizacione probleme često daje dobra rešenja, ali ne i najbolja. Kombinovanje metaheuristika sa drugim optimizacionim tehnikama može proizvesti novu fleksibilniju i efikasniju metodu za rešavanje velikih instanci problema. Takve metode se nazivaju hibridne metode. U poslednje vreme raste interesovanje za hibridne metode, što je podstaknuto činjenicom da su najbolji rezultati u velikom broju akademskih i praktičnih problema dostignuti upravo ovim metodama. Cilj hibridizacije je da se dobre osobine dve ili više metoda sjedine u novu, složeniju, ali i efikasniju metodu. Ono što je plan za dalji ravoj algoritma jeste hibridizacija sa navedenim metaheuristikama u ovom pogavlju.

5.1. Hibridizacija sa algoritmom simuliranog kaljenja

Simulirano kaljenje je metoda koja je dobila ime po analogiji sa kaljenjem čelika. Prilikom kaljenja čelik se zagreva do određene temperature, a zatim se ostavlja da se polako hladi. Simulirano kaljenje je iterativna metoda pretraživanja koja u svakoj iteraciji na slučajan način bira jedno rešenje iz okoline tekućeg rešenja. Ako je to rešenje bolje, onda ono postaje novo tekuće rešenje. Ako je novo rešenje lošije od tekućeg, ono ipak postaje tekuće rešenje ali sa određenom verovatnoćom. Verovatnoća prihvatanja lošijeg rešenja vremenom opada i obično zavisi od parametra koji se naziva temperatura, što daje sličnost sa procesom kaljenja čelika. U početku je ta verovatnoća velika, pa će u cilju prevazilaženja lokalnog optimuma lošije rešenje biti prihvaćeno. Pred kraj izvršavanja algoritma verovatnoća prihvatanja lošijeg rešenja je jako mala i to se verovatno neće ni desiti, jer se smatra da je optimum dosegnut ili se nalazi blizu najboljeg posećenog rešenja, pa se izbegava pogoršanje tekućeg rešenja.

Razlike između simuliranog kaljenja i tabu pretraživanja su prilično upadljive. Nesumnjivo najistaknutija je eksploatacija memorijskih struktura u algoritmu tabu pretraživanja, što je odsutno kod simuliranog kaljenja. Uvođenje memorijskih struktura podrazumeva i različitosti u samom mehanizmu pretrage koja se kod simuliranog kaljenja zasniva na slučajnosti, nasuprot pretrage TS algoritma.

Tabu pretraživanje pretražuje celu tekuću okolinu (ili njen veliki deo) da bi identifikovao poteze visokog kvaliteta. Ovo se razlikuje sa simuliranim kaljenjem koje na slučajan način bira potez i na njega primenjuje unapred definisani kriterujum prihvatanja koji odlučuje da li će se taj potez izvršiti. Kriterijum prihvatanja poteza zanemaruje kvalitet ostalih mogućih poteza. Pomenuta razlika ukazuje na prednost tabu pretraživanja na polju pretrage okoline. Zbog navedene razlike simulirano kaljenje zahteva veći broj iteracija u odnosu na tabu pretraživanje za dobijanje rešenja istog kvaliteta. Samim tim i vreme izvršavanja programa je veće. Pored toga, simulirano kaljenje nije uvek u stanju da dosegne optimalno rešenje iako je jako blizu njega. Zato se često koristi hibridizacija u kojoj SA generiše početno rešenje za TS. SA zbog svoje prirode jednostavno može naći kvalitetne delove pretraživačkog prostora koje će TS efikasno istražiti. Sličan pristup prikazan je u [62].

Takođe, i tabu pretraživanje ima nedostatke. I pored pozitivnog uticaja tabu liste u sprečavanju ciklusa, može se desiti da se jave ciklusi. To je slučaj kada je dužina ciklusa veća od dužine tabu liste (dužina tabu liste je osetljiv parametar TS algoritma o čemu je bilo reči). U takvim

situacijama deterministička priroda tabu pretraživanja je nedostatak jer upravo ona otežava prevazilaženju ciklusa. U cilju prevazilaženja nedostataka TS algoritma, može se razmatrati njegova kombinacija sa SA. U toj kombinaciji prethodno kritikovana stohastička karakteristika simuliranog kaljenja dobija pozitivan kontekst, jer slučajnim biranjem rešenja izbegavaju se ciklusi u pretrazi. Ovakvi pristupi u kojima se TS algoritam poboljšava komponentama simuliranog kaljenja mogu se naći u [12], [40].

Još jedan nedostatak SA algoritma se prevazilazi hibridizacijom sa TS-om. Naime, zbog nedostatka memorije a i zbog osobina procedure pretraživanja simuliranog kaljenja, moguće je da SA više puta poseti jedno rešenje. Ova situacija se izbegava hibridizacijom sa TS algoritmom, tj. upotrebom njegovih memorijskih struktura.

5.2. Hibridizacija sa genetskim algoritmima

Genetski algoritmi predstavljaju metodologiju pretrage diskretnog skupa rešenja na način koji je sličan biološkim procesima u prirodi. Metodologija je prilično opšta i može se primeniti na različite probleme koji ispunjavaju sledeće uslove:

- rešenje problema se može predstaviti kao niska karaktera,
- funkcija pogodnosti (*fitness function*), koja predstavlja kvalitet rešenja, može biti izračunata za svaku validnu nisku karaktera,
- niske koje u sebi sadrže delove veoma dobrih rešenja imaju veću vrednost funkcije pogodnosti od onih koje predstavljaju prosečna rešenja.

Genetski algoritmi, kao što im samo ime govori, jesu familija algoritama, a ne jedan jedinstveni algoritam. To znači da kod svih varijanti postoji osnovna ideja, dok konkretne aplikacije mogu biti veoma različite. Osnovna ideja je oponašanje evolucijskih procesa u prirodi. Genetski algoritmi su populacijski algoritmi, tj. operišu sa skupom rešenja (populacijom). Od svih rešenja na određeni način biraju se parovi (ili podskupovi) rešenja koji predstavljaju roditeljska rešenja i na osnovu njih procesom ukrštanja dobijaju se rešenja potomci. Rešenja potomci mutiraju (tj. razvijaju se) i ona najbolja će "preživeti" i kreirati novu generaciju. Način biranja roditeljskih rešenja, način njihovog ukrštanja, kao i mutiranje potomaka zavise od konkretnog problema i konkretne implementacije algoritma.

Osnovna razlika GA i TS algorima je to što su genetski algoritmi populacijski, nasuprot tabu pretraživanju koje nastoji da popravi (jedno) tekuće rešenje. Posledica rada samo sa jednim rešenjem je da tabu pretraživanje može da zaobiđe atraktivne delove pretraživačkog prostora. Sa druge strane, zbog razmatranja čitave populacije rešenja genetski algoritmi uspevaju da lociraju atraktivne delove pretraživačkog prostora. Njihov nedostatak je što ponekad nisu u mogućnosti da dosegnu optimalno rešenje iako je populacija koncentrisana oko optimalnog rešenja. Zbog toga se genetski algoritmi često kombinuju sa nekim pretraživačkim metodama. Još jedan problem GA sastoji se u tome da sa porastom dimenzije problema koji se razmatra opada kvalitet rešenja, što je posledica operatora ukrštanja koji može značajno da smanji kvalitet rešenja.

Zbog svoje populacijske prirode, intutivnija je kombinacija ovih algoritama u kojoj se GA unapređuje funkcionalnostima tabu pretraživanja. Jedan mogući pristup odnosi se na poboljšanje kvaliteta jedinki populacije primenom tabu pretraživanja. Posle generisanja svake generacije algoritam tabu pretraživanja se može iskoristiti za potrebe operatora genetskog algoritma. Česta je upotreba za definisanje operatora mutacije i to tako što se jedinke nove populacije koriste kao početna rešenja i unapređuju se tabu pretraživanjem. Primer ovakvog operatora mutacije može se naći u [36]. Dužina izvršavanja tabu pretraživanja se obično ograničava na unapred zadato vreme jer bi u suprotom vreme izvršavanja hibridnog algoritma značajno poraslo. Pored definisanja efikasnog operatora mutacije, tabu pretraživanje se često koristi za generisanje nove populacije

[54], [57]. U [54] se određeni procenat jedinki nove populacije generiše tabu pretraživanjem, dok se ostatak generiše standarnim operatorom ukrštanja. U [57] se TS algoritam koristi za poboljšanje svih jedinki nove populacije.

Pored navedene hibridizacije moguća je i suprotna, tj. hibridizacija u kojoj je tabu pretraživanje glavni nosilac nove hibridne metode. Ova hibridizacija se ređe koristi. Poznato je da su intenzifikacija i diversifikacija osetljivi delovi TS algoritma. Intenzifikacija, koja ima za cilj produkciju kvalitetnog rešenja, može biti realizovana kao vrsta genetskog algoritma čija početna populacija može biti skup elitnih rešenja posećenih tokom pretrage. Bitno je da tako realizovana intenzifikacija bude procedura sa malim zahtevima za resurse (kako vremenske tako i memorijske) da se ne bi smanjila efikasnost tabu pretraživanja. Cilj ovakve hibridizacije je dobijanje novog, efikasnijeg algoritma koji bi prevazišao manu originalnog tabu pretraživanja da (ponekad) zaobilazi čitave delove pretraživačkog prostora.

5.3. Hibridizacija sa metodom promenljivih okolina

Metoda promenljivih okolina (*Variable Neighborhood Search* – VNS) je relativno novija iterativna metoda koja pokušava da prevaziđe lokalni optimum menjanjem strukture okoline [39], [28]. Osnovna verzija ove metode polazi od datog početnog rešenja i u svakoj iteraciji u okolini tekućeg rešenja na slučajan način bira susedno rešenje koje postaje polazno rešenje za lokalno pretraživanje. Lokalno pretraživanje se izvršava do lokalnog optimuma. Ukoliko je lokalni optimum bolje rešenje od tekućeg rešenja, onda on postaje novo tekuće rešenje. Ako to nije slučaj, proširuje se okolina tekućeg rešenja i u njoj se ponovo slučajno bira susedno rešenje. Ono što zahteva VNS jeste definisanje niza okolina $N^k, k \in \{k_{min}, \dots, k_{max}\}$. Obično se taj niz definiše tako da kako raste k tako raste i kardinalnost okoline N^k, tj. $|N^{k_{min}}(s)| < |N^{k_{min}+1}(s)| < \cdots < |N^{k_{max}}(S)|$.

Prvo što se kod VNS-a uočava je upotreba lokalnog pretraživanja koje može biti neefikasno. Ono što se nameće je upotreba tabu pretraživanja umesto lokalnog pretraživanja. Tabu pretraživanje pomoću svojih memorijskih struktura može prevazići prvi lokalni minimum koji zaustavlja obično lokalno pretraživanje. Samim tim smanjuje se verovatnoća da novo rešenje bude lošije od tekućeg rešenja. To dalje smanjuje potrebu za čestom promenom okoline, što može povoljno uticati na vreme izvršavanja programa. Drugi osetljivi deo metode promenljivih okolina je slučajno biranje susednog rešenja u tekućoj okolini. Čest slučaj jeste izbor lošeg susednog rešenja što dalje utiče na efikasnost izvršavanja faze pretrage. U [30] primenjeni su navedeni pristupi tako što su elementi tabu pretraživanja ugrađeni i u proces pretrage i u proces biranja susednog rešenja u tekućoj okolini (predloženi algoritam nazvan je *Variable Neighborhood Tabu Search* – VNTS).

6. Zaključak

U ovom radu razmatrana je metaheuristika tabu pretraživanja. Na samom početku predstavljeni su istorijski podaci o nastanku i razvoju ove metaheuristike kao i osnovna varijanta algoritma. Pored osnovnih komponenti algoritma kao što su tabu lista i tabui, dat je opis proširenja osnovnih koncepata među kojima je najviše posvećena pažnja intenzifikaciji i diversifikaciji, kao i modifikaciji funkcije cilja. Algoritam tabu pretraživanja primenjen je na dva različita NP-teška problema.

Prvi od razmatranih problema je generalizovani problem pridruživanja. Prilikom njegovog rešavanja izvršena je relaksacija problema čime je omogućeno strateško oscilovanje, tj. naizmenično posećivanje dopustivih i nedopustivih rešenja. U predloženom algoritmu ciklično se smenjuju faze pretraživanja, intenzifikacije i diversifikacije. Faza diversifikacije je implementirana kao diversifikacija restarta. Na kraju poglavlja o GAP-u dati su rezultati testiranja u kojima se vidi da je algoritam u većini slučajeva dostigao teorijski optimum. U slučajevima kada nije bilo tako, algoritam je bio veoma blizu teorijskog optimuma. Posebni značaj algoritma je rešavanje instanci za koje se ne zna teorijski optimum (*gapa1*, *gapa2*). Predloženi algoritam je u kratkom roku dostizao najbolje rešenje, a razlog tome je posebno konstruisana heuristika za generisanje početnog rešenja. Manja preciznost u pojedinim rezultatima prouzrokovana je proceduri diversifikacije restarta. Ipak, eksperimentalno je pokazano da u predloženom algoritmu kontinualna diversifikacija poboljšava preciznost algoritma, ali značajno povećava vreme izvršavanja. Stoga je ova vrsta diversifikacije odbačena.

Drugi razmatrani problem je problem rutiranja vozila sa vremenskim ograničenjima koji pripada familiji problema rutiranja vozila. U poglavlju o ovom problemu najpre se govori o nastanku problema rutiranja vozila i istorijskom razvoju njegovih varijanti. I kod ovog problema prilikom pretrage omogućena je poseta nedopustivih rešenja, a sva prekoračenja se dodatno naplaćuju. Predloženi algoritam sastoji se od naizmeničnog smenjivanja faze pretraživanja i faze intenzifikacije, dok je faza diversifikacije implementirana kao kontinualna diversifikacija, tj. diversifikacija je uključena u proces pretrage. Na kraju su dati rezultati testiranja dve grupe instanci. Prva grupa se sastoji od instanci sa 25 a druga od instanci sa 50 lokacija i za njih važi da je skup lokacija odgovarajuće instance prve grupe podskup skupa lokacija odgovarajuće instance druge grupe. Algoritam je kod obe grupe instanci pokazao veliku preciznost, naročito kod instanci sa 25 korisničkih lokacija. Pozitivan uticaj na preciznost algoritma imala je procedura kontinualne diversifikacije, kao i strateško oscilovanje. Dinamičkim podešavanjem parametara α i β u izmenjenoj funkciji cilja proces pretrage se naizmenično kretao kroz dopustiva i nedopustiva rešenja. To je uveliko poboljšalo raznovrsnost u izboru tekućeg rešenja i pozitivno uticalo na krajnje rezultate predloženog algoritma. Ipak, vreme izvršavanja veće je od očekivanog. Sa porastom dimenzija instanci iz eksperimentalnih rezultata može se videti pad preciznosti algoritma i porast vremena izvršavanja. O ovim problemima diskutovano je u sekciji 4.4.4. Kao rešenje predlaže se hibridizacija sa nekom od metoda navedenih u poglavlju 5.

Na osnovu eksperimentalnih rezultata oba problema može se zaključiti da je procedura kontinualne diversifikacije imala bolji uticaj na proces pretrage u odnosu na diversifikaciju restarta. Takođe, dinamičko podešavanje parametara α i β u modifikovanoj funkciji cilja VRPTW-a imalo je bolji efekat na rezultate algoritma od pristupa koji koristi statički parametar M u modifikaciji funkcije cilja GAP-a. Ipak, dinamičko podešavanje parametara je jako osetljivi deo implementacije strateškog oscilovanja. Ukoliko se proces menjanja parametara ne odredi valjano, strateško oscilovanje neće dati željene rezultate. Zbog lošeg izbora parametara pretraga može provoditi previše vremena bilo u dopustivom bilo u nedopustivom skupu rešenja. Time se gubi osnovno svojstvo ove tehnike, a to je obezbeđivanje raznovrsnosti rešenja.

Na kraju rada, u poglavlju 5, razmatrani su dalji koraci u razvoju algoritma tabu pretraživanja. Predlaže se hibridizacija algoritma sa nekom od navedenih metaheuristika u tom poglavlju. Prikazane su mane i prednosti svih metaheuristika, kao i poboljšanje koje se očekuje eventualnom hibridizacijom.

7. Literatura

[1] **Baker, B. M., Ayechew, M. A.**: A genetic algorithm for the vehicle routing problem. *Computers & Operations Research 30*, 787–800, (2003)

[2] **Bräysy, O., Gendreau, M.**: Vehicle Routing Problem with Time Windows, Part I: Route Constructions and Local Search Algorithms. *Transportation Science 39*, 104-118, (2005)

[3] **Bräysy, O., Gendreau, M.**: Vehicle Routing Problem with Time Windows, Part II: Metaheuristics. *Transportation Science 39*, 119-139, (2005)

[4] **Crainic, T. G., Gendreau, M., Soriano, P., Toulouse, M.**: A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research 41*, 359–383 (1993)

[5] **Cook, W., Rich, J. L.**: A parallel cutting-plane algorithm for the vehicle routing problems with time windows. Technical report TR99-04, Department of Computational and Applied Mathematics, Rice University, Houston, TX, (1999)

[6] **Cordeau, J. F., Desaulniers, G., Desrosiers, J., Solomon, M. M., Soumis, F.**: The VRP with time windows. Toth, P., Vigo, D., eds. *The Vehicle Routing Problem, SIAM Monographs on Discrete Mathematics and Applications*. SIAM, Philadelphia, PA, 157–194, (2001)

[7] **Dantzig, G., Ramser, J.**: The truck dispatching problem. *Management Science 6*, 80–91 (1959)

[8] **De Backer B., V. Furnon, P. Prosser, P. Kilby, P. Shaw.**: Local search in constraint programming: Application to the vehicle routing problem. *Proc.CP-97 Workshop Indust.Constraint-Directed Scheduling*, Schloss Hagenberg, Austria, 1–15, (1997)

[9] **De Werra, D., Hertz, A.**: Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum 11*, 131–141 (1989)

[10] **Desrochers, M., J. K. Lenstra, Savelsbergh, M. W. P., Soumis, F.**: Vehicle routing with time windows: Optimization and approximation. Golden, B., Assad, A.: eds. Vehicle Routing: Methods and Studies. Elsevier Science Publishers, Amsterdam, The Netherlands, 65–84, (1988)

[11] **Desrosiers, J., Dumas, Y., Solomon, M. M., Soumis F.**: Time constrained routing and scheduling. Ball, M. O., Magnanti, T. L., Monma, C. L., Nemhauser, G. L., eds. *Handbooks in Operations Research and Management Science 8: Network Routing*. Elsevier Science Publishers, Amsterdam, The Netherlands, 35–139, (1995)

[12] **Dharmapriya, U. S. S, Siyambalapitiya, S. B., Kulatunga, A. K.**: Simulated Annealing and Tabu Search Based Hybrid Algorithm for Multi depot Vehicle Routing Problem with Time Windows and Split Delivery, *International Conference on Industrial Engineering and Operations Management Dhaka, Bangladesh, January 9 – 10,* (2010)

[13] **Diaz, J. A., Fernandez, E.**: A Tabu search heuristic for the generalized assignment problem, *European Journal of Operation Research 132*, 22-38, (2001)

[14] **Feltl, H., Raidl, G. R.**: An improved hybrid genetic algorithm for the generalized assignment problem, *Proceedings of the 2004 ACM Symposium on Applied Computing*, Nicosia, Cyprus, 990–995, (2004)

[15] **Fisher, M. L., Jaikumar, R., Van Wassenhove, L. N.**: A multiplier adjustment method for the generalized assignment problem, *Management Science 32*, 1095–1103, (1986)

[16] **Gendreau, M., Hertz, A., Laporte, G.**: A tabu search heuristic for the vehicle routing problem. *Management Science 40*, 1276–1290 (1994)

[17] **Gendreau, M., Soriano, P., Salvail, L.**: Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research 41*, 385–403 (1993)

[18] **Glover, F.**: Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research 13*, 533–549 (1986)

[19] **Glover, F.**: Heuristics for integer programming using surrogate constraints. *Journal of Applied Mathematics and Decision Sciences 8*, 156–166 (1977)

[20] **Glover, F., Laguna, M.**: *Tabu Search*, Kluwer, Boston, MA., (1997)

[21] **Glover, F., Taillard, E. D., de Werra, D.**: A user's guide to tabu search, *Annals of Operations Research 41*, 3-28, (1993)

[22] **Glover, F.**: Tabu search – Part I. *ORSA Journal on Computing 1*, 190–206 (1989)

[23] **Glover, F.**: Tabu search – Part II. *ORSA Journal on Computing 2*, 4–32 (1990)

[24] **Golden, B. L., Assad, A. A.**: Perspectives on vehicle routing: Exciting new developments. *Operational Research. 34*, 803–809, (1986)

[25] **Golden, B. L., Assad, A. A.**: Vehicle Routing: Methods and Studies. Elsevier Science Publishers, Amsterdam, The Netherlands, (1988)

[26] **Hall. R.**: On the road to integration. *OR/MS Today 33(3)*, 50–57 (2006)

[27] **Halse, K.**: Modeling and solving complex vehicle routing problems. Ph.D. thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, (1992)

[28] **Hansen, P., Mladenović, N., Pérez-Brito, D.**: Variable neighborhood decomposition search, *Journal of Heuristics 7*, 335–35, (2001)

[29] **Hertz, A., De Werra, D.**: The tabu search metaheuristic: how we used it. *Annals of Mathematics and Artificial Intelligence 1*, 111–121 (1991)

[30] **José A. Moreno Péres, J. Marcos Moreno-Vega, Inmaculada Rodríguez Martín**: Variable neighborhood tabu search and its application to the median cycle problem, *European Journal of Operational Research 151*, 365–378, (2003)

[31] **King, G. F., Mast, C. F.**: Excess travel: causes. Extent and consequences. *Transportation Research Record 1111*, 126–134, (1997)

[32] **Kirkpatrick, S., Gelatt Jr., C. D., Vecchi, M. P.**: Optim. Simulated Annealing. *Science 220*, 671–680 (1983)

[33] **Kuhn H. W.**: The Hungarian method for the assignment problem, *Naval Research Logistics Quarterly 2 (1&2)* 83–97 (original publication), (1955)

[34] **Larsen, J.**: Parallelization of the vehicle routing problem with time windows. Ph.D. thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, (1999)

[35] **Lenstra, J. K., Rinnooy, Kan, A. H. G.**: Complexity of vehicle and scheduling problem. *Networks 11*, 221–227, (1981)

[36] **Mak, K. L., Sun, D.**: A new hybrid Genetic Algorithm and Tabu Search method for Yard Cranes Scheduling with Inter-crane Interference, *World Congress on Engineering 2009 Vol I*, (2009)

[37] **Martello, S., Toth, P.**: Knapsack Problems, *Algorithms and Computer Implementations*, Wiley, New York, (1990)

[38] **Mingozzi, A., Giorgi S., Baldacci R.**: Solution of a large scale traveling salesman problem. *Management Science 6*, 80–91, (1999)

[39] **Mladenović, N., Hansen, P.**: *Variable neighborhood search. Computers & Operations Research 24*, 1097–1100, (1997)

[40] **Nader, A., Hamid, D.**: A hybrid Tabu-SA algorithm for location-inventory model with considering capacity levels and uncertain demands, *Journal of Information and Computing Science Vol. 3, No. 4*, 290-304, (2008)

[41] **Nauss, R. M.**: Solving the generalized assignment problem: an optimizing and heuristic approach, *Informs Journal of Computing 15*, 249–266, (2003)

[42] **Osman, I. H.**: Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches, *OR Spektrum 17*, 211–225, (1995)

[43] **Osman, I. H., Kelly, J. P.**: *Meta-Heuristics: Theory and Applications*, Kluwer, Boston, MA., (1996)

[44] **Pardalos, P. M., Resende, M. G. C.**: *Handbook of Applied Optimization*, Oxford University Press, New York., (2002)

[45] **Randall, M.**: Heuristics for ant colony optimisation using the generalised assignment problem, *Proceedings of IEEE Congress on Evolutionary Computation*, Portland, Oregon, USA, 1916–1923, (2004)

[46] **Ribeiro, C. C., Hansen, P.**: *Essays and Surveys in Metaheuristics*, Kluwer, Norwell, MA. (2002)

[47] **Rousseau, L. M., Gendreau, M., Pesant, G.**: Solving small VRPTWs with Constraint Programming Based Column Generation, *CAPIOR'02*, (2002)

[48] **Sahni, S., Gonzalez, T.**: P-complete approximation problems. *Journal of ACM 23*, 555–565, (1976)

[49] **Savelsbergh, M.**: A branch-and-price algorithm for the generalized assignment problem, *Operations Research 45*, 831–841, (1997)

[50] **Skorin-Kapov, J.**: Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing 2*, 33–45 (1990)

[51] **Solomon, M. M.**: Algorithms for the vehicle routing problem and scheduling problem with time windowconstraints. *Operational Research 35*, 254–265, (1987)

[52] **Solomon, M. M., Desrosiers, J.**: Time window constrained routing and scheduling problems. *Transportation Science 22*, 1–13, (1988)

[53] **Soriano, P., Gendreau, M.**: Diversification strategies in tabu search algorithms for the maximum clique problem. *Annals of Operations Research 63*, 189–207 (1996)

[54] **Sudhakaran, M., Ajay-D-Vimal Raj, P.**: Integrating genetic algorithms and tabu search for unit commitment problem, *International Journal of Engineering, Science and Technology Vol. 2, No. 1*, pp. 57-69, (2010)

[55] **Taillard, É. D.**: Robust taboo search for the quadratic assignment problem. *Parallel Computing 17*, 443–455 (1991)

[56] **Taillard, É. D.**: Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research 47*, 65–74 (1990)

[57] **Thamilselvan, R., Balasubramanie, P.**: Integration of Genetic Algorithm with Tabu Search for Job Shop Scheduling with Unordered Subsequence Exchange Crossover, *Journal of Computer Science 8 (5)*, 681-693, (2012)

[58] **Toth, P., Vigo, D.**: Branch-and-bound algorithms for the capacitated VRP. *Society for Industrial and Applied Mathematics*, 29-51, (2001)

[59] **Voss, S., Martello, S., Osman, I. H.,Roucairol, C.**: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, Boston, MA., (1999)

[60] **Votaw, D. F., Orden, A.**: The personnel assignment problem, Symposium on Linear Inequalities and Programmng, SCOOP 10, US Air Force, pp. 155–163, (1952)

[61] **Woodcock, A. J., Wilson, J. M.**: A hybrid tabu search/branch & bound approach to solving the generalized assignment problem, *European Journal of Operation Research 207*, 566–578, (2010)

[62] **Yiannis, A. K., Pavlos, S. G., Emmanuel, S. K.**: Hybrid Simulated Annealing–Tabu Search Method for Optimal Sizing of Autonomous Power Systems With Renewables, *IEEE Transactions on sustainable energy, Vol. 3, No. 3*, 330-338, (2012)

[63] **Özbakir, L., Baykasoğlu, A., Tapkan, P.**: Bees algorithm for generalized assignment problem, *Applied Mathematics and Computation 215*, 3782–3795, (2010)