UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Predrag S. Stanojević

Egzaktne i metaheurističke metode za rešavanje NP-teških lokacijskih problema

Doktorska disertacija

Beograd, 2016

UNIVERSITY OF BELGRADE
FACULTY OF MATHEMATICS

Predrag S. Stanojević

Exact and metaheuristic methods for solving NP-hard location problems

Doctoral dissertation

Belgrade, 2016

Podaci o mentoru i članovima komisije

Mentor

dr Miroslav Marić, vanredni profesor, Matematički fakultet, Univerzitet u Beogradu

Članovi komisije

dr Dušan Tošić, redovni profesor, Matematički fakultet, Univerzitet u Beogradu

dr Miodrag Živković, redovni profesor, Matematički fakultet, Univerzitet u Beogradu

dr Zorica Stanimirović, vanredni profesor, Matematički fakultet, Univerzitet u Beogradu

dr Aleksandar Takači, vanredni profesor, Tehnološki fakultet, Univerzitet u Novom Sadu

Datum odbrane:

Podaci o doktorskoj disertaciji

Naslov doktorske disertacije: Egzaktne i metaheurističke metode za rešavanje NP-teških lokacijskih problema

Rezime: U ovoj disertaciji predstavljene su metode za rešavanje lokacijskih problema optimizacije. Ovi problemi predstavljaju bitan deo široke oblasti operacionih istraživanja. Pri rešavanju lokacijskih problema potrebno je odrediti koji čvorovi u mreži će predstavljati centre (habove, uslužne centre, dobavljačke centre, ustanove i sl.), a preostale čvorove-korisnike treba pridružiti uspostavljenim centrima. Centri međusobno mogu biti ravnopravni ili može biti uspostavljena određena hijerarhija među njima. Pri tom je potrebno minimizovati ili maksimizovati vrednost funkcije cilja.

S obzirom da su lokacijski problemi najčešće NP-teški, za rešavanje primera većih dimenzija uobičajeno je da se koriste metaheuristike različitog tipa. Sva tri konkretna lokacijska problema, koja su razmatrana u ovom radu, su NP-teška: hab lokacijski problem sa jednostrukom alokacijom i sa ograničenjem kapaciteta (Capacitated Single Allocation Hub Location Problem - CSAHLP), hab lokacijski problem sa jednostrukom alokacijom bez ograničenja kapaciteta (Uncapacitated Single Allocation Hub Location Problem - USAHLP) i lokacijski problem snabdevača neograničenog kapaciteta u više nivoa (Multi Level Uncapacitated Facility Location Problem - MLUFLP).

U disertaciji je prikazan niz metoda, počev od memetskog algoritma, preko hibridne metode do iterativne heurističke metode i egzaktne metode. Memetski algoritam, koji je razvijen, sastoji se od genetskog algoritma (GA) i lokalne pretrage. Zatim, hibridna metoda GA-BnB se sastoji od genetskog algoritma i metode grananja i ograničavanja (eng. branch-and-bound, skr. BnB). Konačno, predstavljena je i metoda dekompozicije za lokacijske probleme, koja polazi od rastavljanja matematičkog modela na nadproblem i potproblem. Primena ove metode je realizovana putem dva algoritma: algoritma proste dekompozicije modela (APDM) i egzaktnog kombinatornog algoritma dekompozicije modela (KADM).

Svrha APDM je da iterativno popravlja rešenja problema, tako što u svakoj iteraciji traži novo validno rešenje nadproblema i za svako, tako dobijeno rešenje,

rešava odgovarajući potproblem. APDM se može koristiti kao heuristika za rešavanje bilo kog problema za koji je moguće definisati potproblem i nadproblem. APDM je uporediv sa poznatim heurističkim metodama i po kvalitetu rešenja koja pronalazi i po vremenu izvršavanja.

Egzaktni kombinatorni algoritam dekompozicije modela (KADM) polazi od rešenja koje je APDM pronašao i pokušava da ga poboljša. KADM analizira kombinacije za uspostavljanje tačno k centara od n čvorova. Svaka takva kombinacija prolazi kroz „sito“ - skup algoritama koji u polinomskom vremenu izvršavanja određuju neke donje granice ciljne funkcije za zadatu kombinaciju habova. Ukoliko donja granica nije manja od pronađene gornje granice, sito odbacuje tu kombinaciju. Za one kombinacije habova koje prođu kroz sito, rešava se potproblem i eventualno ažurira trenutno najbolje rešenje. Po završetku rada KADM, najbolje pronađeno rešenje je ujedno i optimalno, što je u radu i dokazano.

Ključne reči: lokacijski problemi, egzaktne metode, kombinatorna optimizacija, dekompozicija, metaheuristike
Naučna oblast: Računarstvo
Uža naučna oblast: Optimizacija
UDK broj: 004.023:519.874(043.3)

Dissertation Data

Doctoral dissertation title: Exact and metaheuristic methods for solving NP-hard location problems

Abstract: Methods for solving location optimization problems are studied and presented in this doctoral thesis. These problems represent an important part of the broad area of Operations Research (OR). Solving location problems involves determining which nodes in a network will represent hubs (service centers, supply centers, facilities etc.) and assigning the client-nodes to the established hubs. Hubs may either be of equal rank or there may be a certain hierarchy between them. The goal of such optimization is to minimize or maximize the objective function value.

Most location problems are NP-hard, so it is customary to use various metaheuristics for solving test problems of large dimensions. All three location problems, which are studied in this thesis, are NP-hard: Capacitated Single Allocation Hub Location Problem (CSAHLP), Uncapacitated Single Allocation Hub Location Problem (USAHLP) and Multi Level Uncapacitated Facility Location Problem (MLUFLP).

In this thesis several methods are presented, starting with a memetic algorithm, followed by a hybrid method and finally an iterative heuristic method and an exact method. The memetic algorithm, which has been developed, consists of a genetic algorithm (GA) and local search procedures. Next, a hybrid GA-BnB method is presented, which consists of a genetic algorithm and a branch-and-bound (BnB) method. Finally, a decomposition method for location problems is presented, which begins with splitting the model into a master problem and a subproblem. Application of this method has been realized by two algorithms: an iterative model decomposition algorithm (MDA) and an exact combinatorial hub location algorithm (CHLA).

The purpose of MDA is to iteratively improve the best solution, by finding a new feasible solution of the master problem in each iteration and then solving the corresponding subproblem for each such solution of the master problem. MDA can be used as a metaheuristic for any problem for which the master problem and the subproblem are well-defined. Experimental results demonstrate that MDA is comparable to the best known heuristic methods, in terms of, both, the solution

quality and the running time.

Exact combinatorial hub location algorithm (CHLA) uses the best solution found by MDA and tries to improve it. CHLA analyzes configurations with k hubs out of n nodes. Each such configuration is tested through a "sieve" - a set of algorithms which in polynomial time find lower bounds for a given configuration. If the lower bound is not smaller than the best known solution, the sieve rejects the hub configuration. The subproblem is solved for those hub configurations which pass through the sieve and if a new, better solution is found, the best solution is updated. At the end of the CHLA run, the best found solution is also optimal, which is proven in the thesis.

Keywords: location problems, exact methods, combinatorial optimization, decomposition, metaheuristics
Scientific field: Computer science
Scientific discipline: Optimization
UDC number: 004.023:519.874(043.3)

Sadržaj

Poglavlje 1

Uvod

Razmatranje lokacijskih problema seže daleko u prošlost. Može se smatrati da je prvi lokacijski problem postavio Pjer de Ferma u 17. veku: potrebno je odrediti tačku u ravni zadatog trougla, takvu da je zbir udaljenosti te tačke od svakog od temena trougla najmanji. Danas se ovaj problem naziva Problemom minimalne sume euklidske udaljenosti tačaka. Savremenije bavljenje lokacijskim problemima počinje krajem šezdesetih godina prošlog veka. Jedan od prvih naučnih radova objavljenih iz oblasti diskretnih lokacijskih problema je rad Goldmana iz 1969. godine [36]. Prvu matematičku formulaciju hab lokacijskih problema dao je O'Keli 1987. godine [53].

1.1 Lokacijski problemi

U širokoj oblasti operacionih istraživanja (eng. operations research, skr. OR), lokacijski problemi zauzimaju značajno mesto. To su optimizacioni problemi u kojima je potrebno uspostaviti lokacije objekata tako da zadata funkcija cilja ima najmanju moguću (ili najveću) vrednost. Kao takvi, lokacijski problemi imaju konkretnu, praktičnu primenu u rešavanju problema infrastrukture, transporta, mreža različitog tipa uključujući i telekomunikacione i računarske itd. Matematičke formulacije lokacijskih problema najčešće i nastaju kao potreba da se pojedini praktični problemi reše. Na primer, dva standardna skupa test primera za hab lokacijske probleme potiču od lokacijskog problema avionskog saobraćaja između 25 američkih gradova [52] i lokacijskog problema dostavljanja pošte u 200 australijskih gradova i mesta [28]. Klasa lokacijskih problema je raznovrsna i sadrži probleme različitog tipa koje

je moguće klasifikovati na više načina.

Pomenuti Fermaov lokacijski problem spada u grupu kontinualnih problema, kod kojih je potrebno odrediti lokacije u kontinualnom prostoru. Sa druge strane, kod diskretnih lokacijskih problema potrebno je izabrati lokacije iz zadatog konačnog skupa diskretnih vrednosti. Postoje i mešoviti lokacijski problemi, kod kojih neke vrednosti mogu biti kontinualne a neke diskretne.

Kod nekih lokacijskih problema, broj objekata, koje treba locirati, je unapred zadat i takvi problemi se nazivaju endogeni. Uobičajeno je da se taj broj označava sa p. Neki od primera takvih problema su: Veberov problem [26], koji je generalizacija Fermaovog problema, zatim problemi p-medijane i p-centra [66] itd. Sa druge strane, kod egzogenih problema broj lokacija nije dat, već se dobija kao rezultat optimizacije.

Problemi kod kojih je uspostavljena hijerarhija između lokacija nazivaju se hijerarhijsko-lokacijskim problemima [47]. Kao primer može poslužiti problem uspostavljanja zdravstvenih ustanova u skupu ponuđenih gradova. U hijerarhijskoj verziji tog problema, u nekim gradovima treba uspostaviti ustanove nižeg nivoa - ambulante, a u nekima ustanove višeg nivoa - domove zdravlja.

Značajnu i često proučavanu podgrupu lokacijskih problema čine hab lokacijski problemi. Pored lociranja objekata, koji se nazivaju habovima, kod ovih problema potrebno je izvršiti i pridruživanje ne-hab objekata, takozvanih korisnika, uspostavljenim habovima. Pridruživanje korisnika habovima naziva se alokacija. Alokacija može biti jednostruka [61], kada se svaki korisnik pridružuje tačno jednom habu ili višestruka [25], kada korisnika može opsluživati veći broj habova. Alokacija, kao i lokacija, utiče na vrednost funkcije cilja. Pregled hab lokacijskih problema i nekih metoda za njihovo rešavanje dati su u radovima [5] i [11].

Raznovrsna priroda lokacijskih problema direktno utiče na matematičke modele kojima su oni predstavljeni. Broj promenljivih, broj uslova i funkcija cilja zavise od prirode problema. Neki matematički modeli lokacijskih problema predstavljeni su u radovima [10] i [17], dok je klasifikacija različitih vrsta lokacijskih problema data u [23] i [31]. Istorijski pregled mnogih lokacijskih problema i kako su oni nastali dat je u radu [9].

Pored pomenutih test primera za hab lokacijske probleme, u literaturi se za testiranje metoda za rešavanje lokacijskih problema najčešće koristi biblioteka OR-LIB (eng. operations research library) [7], koja sadrži brojne test primere, ne samo

lokacijskih, već i mnogih drugih problema iz oblasti operacionih istraživanja.

Predmet ove disertacije je razvoj metoda i algoritama za rešavanje diskretnih lokacijskih problema sa fokusom na rešavanje problema velikih dimenzija. Problemi koji su ovde razmatrani pripadaju grupi NP-teških problema te složenost pronalaženja optimalnog rešenja vrlo brzo raste sa porastom dimenzije problema.

1.2 Složenost i NP-teški problemi

Problem za koji ne postoji efektivan algoritam, koji bi ga u opštem slučaju rešio, naziva se algoritamski nerešiv problem. Neki naizgled jednostavni problemi su algoritamski nerešivi. Na primer: ako je dat konačan skup celobrojnih kvadratnih matrica, da li je njihovim množenjem, u bilo kom redosledu i sa ponavljanjem, moguće dobiti 0-matricu? Opšti slučaj ovog problema je nerešiv, što znači da nije moguće dati odgovor na to pitanje. Drugi poznati primeri algoritamski nerešivih problema su: problem zaustavljanja Tjuringove mašine, Hilbertov problem odlučivosti logike prvog reda, problem nalaženja celobrojnog rešenja polinomske Diofantove jednačine itd.

Drugu klasu čine problemi za koje se zna da su algoritamski rešivi. Pod tim terminom se podrazumeva da postoji Tjuringova mašina koja u konačnom broju koraka rešava taj problem, za svaki dozvoljeni skup ulaznih podataka. Ulazni podaci određuju dimenziju te konkretne instance problema koja je njima predstavljena. Ta dimenzija se označava jednim prirodnim brojem n.

Intuitivno je jasno da su neki problemi teže rešivi a neki lakše. Složenost je funkcija kojom se to izražava i njome se meri koliko neki algoritam koristi memorijskog prostora ili procesorskog vremena, u zavisnosti od dimenzije problema n. Ta složenost je predstavljena nekom funkcijom $g(n)$. Podrazumeva se da $g(n)$ predstavlja složenost „najteže“ instance problema dimenzije n, odnosno maksimalno potrebno vreme izvršavanja ili maksimalnu količinu memorije. Za analizu je svakako interesantnija vremenska složenost izvršavanja nekog algoritma.

Definicija 1. *Neka je funkcija* $g : \mathbb{N}^+ \to \mathbb{R}^+$*, takva da* $g(n)$ *predstavlja složenost nekog algoritma za dimenziju ulaznih podataka* n*. Neka je* $f : \mathbb{N}^+ \to \mathbb{R}^+$ *takva da* $\exists c \in \mathbb{R}^+$ *tako da* $\forall n \in \mathbb{N}^+, g(n) \leq c \cdot f(n)$*. Tada se oznakom* $O(f(n))$ *označava asimptotska složenost tog algoritma.*

Kada se govori o složenosti algoritma za rešavanje nekog problema, zapravo se govori o asimptotskoj složenosti, u oznaci $O(f(n))$.

Definicija 2. *Skup* $\mathbb{P}$ *je skup svih problema za koje postoji algoritam čije je vreme izvršavanja polinomsko, odnosno koji se izvršava u vremenu* $O(n^k)$*, za neko* k*.*

Definicija 3. *Skup* $\mathbb{NP}$ *je skup svih algoritamski rešivih problema odlučivosti za koje se tačnost bilo kog ponuđenog rešenja može proveriti u polinomskom vremenu u odnosu na dimenziju date instance problema.*

Prethodne definicije su intuitivne i neformalne. Formalne definicije, koje se zasnivaju na problemima odlučivosti nekog jezika L, determinističkoj i nedeterminističkoj Tjuringovoj mašini se mogu naći u [68].

Očigledno je da ako je moguće rešiti neki problem u polinomskom vremenu, onda je moguće i proveriti neko njegovo rešenje u polinomskom vremenu, odnosno, jasno je da $\mathbb{P} \subseteq \mathbb{NP}$. Veruje se da obrnuto ne važi.

Hipoteza 1. $\mathbb{P} \neq \mathbb{NP}$

Pitanje da li su klase algoritamski rešivih problema $\mathbb{P}$ i $\mathbb{NP}$ identične prvi je postavio Kurt Gedel 1956. godine. Iako je opšte prihvaćeno mišljenje da su te dve klase različite, dokaz i dalje ne postoji. Smatra se da je to najvažnije nerešeno pitanje računarstva i nalazi se na listi sedam najvažnijih pitanja matematike, takozvanih milenijumskih problema, koja je objavljena 2000. godine.

Definicija 4. *Problem A je svodljiv na problem B ako i samo ako postoji algoritam kojim je moguće u polinomskom vremenu prevesti svaku instancu problema A na neku instancu problema B.*

Lema 1. *(Lema o svodljivosti) Ako problem A može da se svede na problem B u polinomskom vremenu, tada važi:*

$B \in \mathbb{P} \Rightarrow A \in \mathbb{P}$*;*

$B \in \mathbb{NP} \Rightarrow A \in \mathbb{NP}$*.*

Dokaz leme o svodljivosti može se naći u [68]. Ona omogućava da $\mathbb{NP}$ problemi budu klasifikovani i da među njima budu određeni oni koji su „najteži“. Naime, ako je problem A svodljiv na problem B, onda se može smatrati da je problem B „teži“, u smislu da ako postoji algoritam za nalaženje svakog rešenja problema B, onda

postoji i algoritam za nalaženje svakog rešenja problema A, dok obrnuto ne mora važiti.

Definicija 5. $\mathbb{NP}$*-težak je onaj problem na koji svaki problem iz klase* $\mathbb{NP}$ *može da se svede.*

Definicija 6. $\mathbb{NP}$*-kompletan je onaj* $\mathbb{NP}$ *problem koji je* $\mathbb{NP}$*-težak.*

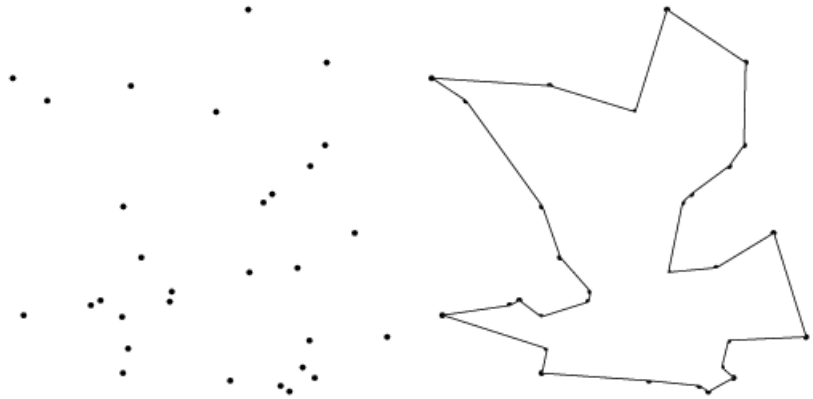
Definicija 7. *Problem zadovoljivosti proizvoljne formulove Bulove algebre, odnosno problem nalaženja vrednosti logičkih promenljivih za koju je ta formula tačna, nazivamo* **SAT** *problemom.*

Stiven Kuk je 1971. godine dokazao da je **SAT** problem $\mathbb{NP}$-kompletan, odnosno da postoje $\mathbb{NP}$ problemi na koje je svaki drugi $\mathbb{NP}$ problem svodljiv u polinomskom vremenu. Ubrzo zatim, Karp je 1972. godine formulisao listu od 21-og kombinatornog i graf problema i dokazao njihovu $\mathbb{NP}$-kompletnost. Za mnoge probleme je od tada dokazano da su $\mathbb{NP}$-kompletni. Neki od poznatijih takvih problema su problem trgovačkog putnika (slika 1.1), problem ranca, problem 0-1 linearnog programiranja, opšti oblici enigmatskih problema kao što su sudoku, kakuro (ukršteni brojevi) i drugi.

Kada bi važilo $\mathbb{P} = \mathbb{NP}$, tada bi sve $\mathbb{NP}$ probleme bilo moguće rešiti u polinomskom vremenu izvršavanja. Kako se smatra da su te dve klase različite, vreme rešavanja $\mathbb{NP}$ i $\mathbb{NP}$-teških problema, sa povećanjem dimenzije ulaznih podataka, raste brže od polinomskog vremena, eksponencijalnom brzinom. Zbog toga je ove probleme teško ili nemoguće rešiti na egzaktan način, pa se pristupa nalaženju približnih rešenja, za šta se koriste (meta)heurističke metode.

1.3 Metaheurističke metode

Pod pojmom heuristička metoda smatra se svaka metoda za nalaženje bilo kog dopustivog rešenja nekog problema. Pronađeno rešenje ne mora biti optimalno, ali mora biti dopustivo. Sa druge strane, pretpostavlja se da se heurističkom metodom problem rešava brže nego metodama koje bi dale egzaktno rešenje. Takođe, ukoliko postoji više rešenja datog problema, heurističkom metodom se pronalaze neka od tih rešenja, a ne obavezno sva. Neke heurističke metode mogu, pored samog rešenja, dati i preciznost tog rešenja, odnosno gornju granicu udaljenosti od optimalnog rešenja.



Slika 1.1: Problem trgovačkog putnika i optimalno rešenje (izvor: WolframAlpha)

Dakle, heurističke metode traže balans između optimalnosti, potpunosti i preciznosti rešenja sa jedne strane, i vremena izvršavanja, sa druge. Kvalitet heurističke metode zavisi od svih pomenutih faktora.

U heurističke metode spadaju: metode lokalne pretrage, konstruktivne metode, matematičke metode, metode mašinskog učenja, iterativne metode i hibridne metode.

Heurističke metode koje nisu osmišljene isključivo za jedan problem, već su, uz određena prilagođavanja, primenljive na veliki broj problema, nazivaju se metaheurističke metode. Savremene metaheurističke metode su brojne, raznovrsne i moguće ih je podeliti na više načina:

- Metaheuristike inspirisane prirodom kao model uzimaju neki prirodni proces i prilagođavaju ga rešavanju velike klase problema. Veliku grupu takvih metaheuristika čine evolutivni algoritmi, u koje spadaju genetski algoritmi, genetsko programiranje, evolutivno programiranje i drugi. Metode mravljih ili pčelinjih kolonija i optimizacija rojem čestica pripadaju metaheuristikama koje se zasnivaju na grupnoj inteligenciji skupa samoorganizujućih jedinki. Zatim, metoda simuliranog kaljenja je inspirisana metalurškim procesom kaljenja, te se i ona može svrstati u ovu grupu.

- Metode sa jednim rešenjem u svakom trenutku manipulišu samo jednim rešenjem koje menjaju i popravljaju. Neke od metaheuristika tog tipa su: tabu pretraga (eng. tabu search), metoda promenljivih okolina (eng. variable neighborhood search; skr. VNS), GRASP metoda i simulirano kaljenje (eng. sim-

ulated annealing; skr. SA). U ovu grupu spadaju i metode lokalne pretrage o kojima će više reči biti u odeljku 1.3.2.

- Matematičke metaheurističke metode (matheuristike, eng. matheuristics) se zasnivaju na modifikaciji matematičkog modela kojim je predstavljen problem, tako da je novodobijeni model moguće rešiti lakše ili brže i od tako dobijenog rešenja dobiti približno rešenje polaznog problema. Najpoznatija takva metoda je Lagranževa relaksacija. Metode dekompozicije, poput Bendersove dekompozicije i Dancig-Vulf dekompozicije dele polazni problem na potprobleme koji su lakši za rešavanje. Rešavaju ih iterativno i u tom procesu u model dodaju bilo nove promenljive bilo nove uslove. Više o dekompoziciji biće reči u poglavlju 4.

- Memetski algoritmi predstavljaju spoj neke od populacionih metoda i neke od metoda sa jednim rešenjem. Sam naziv ove grupe algoritama u osnovi ima reč *meme* koja potiče od grčke reči *mimema* i označava imitaciju, mimikriju [22]. Prvi put se pominju 1989. godine [51] i od tada su prošli nekoliko faza razvoja. Jedna od osnovnih varijanti jeste da se genetskom algoritmu doda novi operator lokalne pretrage, koji lokalno pokušava da popravi rešenje nekih jedinki iz populacije. Kasnije su razvijeni memetski algoritmi koji koriste *meme* zapisane u samom genetskom kodu. Ova vrsta metaheurističkih metoda je vrlo aktuelna i nastavlja da se razvija.

- Savremeni pristup primene metaheurističkih metoda sve češće se okreće hibridizaciji neke od pomenutih metaheuristika i drugih optimizacionih metoda, na primer matematičkog programiranja ili metode grananja i ograničavanja. Primenjene metode se mogu izvršavati bilo konkurentno bilo konsekutivno, tako da razmenjuju podatke i pronađena rešenja i koriste ih u daljem procesu optimizacije. Neki primeri hibridnih metoda prikazani su u radovima [13], [2], [63], [45].

Na mnoge od pomenutih metoda moguće je primeniti paradigme paralelnog programiranja. Na taj način se može postići ubrzanje u pronalaženju rešenja, a može se i pretražiti veći deo prostora rešenja i time poboljšati kvalitet pronađenog rešenja ili smanjiti gornja granica razlike do optimalnog rešenja.

1.3.1 Genetski algoritmi

Genetski algoritmi pripadaju grupi populacionih stohastičkih algoritama evolutivne optimizacije. Ideja o primeni simuliranih evolutivnih procesa sa ciljem kreiranja mašine koja uči potiče iz 1950. godine i izneo ju je Alan Tjuring. Za razliku od principa bioinformatike, gde se naučno znanje iz informatike primenjuje na rešavanje problema iz biologije, medicine i drugih prirodnih nauka, kod evolutivnih algoritama i drugih algoritama inspirisanih prirodom, situacija je obrnuta: prirodni procesi se simuliraju i primenjuju na rešavanje problema matematike i informatike.

Prve simulacije koje su imale sve suštinske elemente današnjih genetskih algoritama izvršio je Aleks Frejzer od 1957. godine pa nadalje. Popularizaciji genetskih algoritama je najviše doprinela knjiga Džona Holanda „Prilagodljivost u prirodnim i veštačkim sistemima“ iz 1975. godine [37].

Osnovna ideja genetskih algoritama jeste da se potencijalno rešenje nekog problema predstavi „jedinkom“: strukturom podataka čije elemente nazivamo „genima“, a da se zatim na skup jedinki, koji nazivamo „populacijom“, iterativno primenjuju evolutivni procesi i tako stvaraju nove jedinke koje predstavljaju nova potencijalna rešenja. Ti evolutivni procesi, koji se nazivaju genetskim operatorima, su: prilagođenost, selekcija, ukrštanje i mutacija i inspirisani su pojmovima i procesima iz teorije evolucije Čarlsa Darvina.

Algoritam 1 prikazuje osnovnu strukturu genetskog algoritma.

Algoritam 1: Osnovni genetski algoritam (GA)

```
UlazniPodaci();
PseudoslučajnoGenerisanjePopulacije();
while not KrajRadaGA() do
    RačunanjeFunkcijeCilja();
    primena operatora:
        Prilagođenost;
        Selekcija;
        Ukrštanje;
        Mutacija;
    NovaGeneracija();
end
Rezultati();
```

Nakon unosa ulaznih podataka, početna populacija jedinki se kreira na pseu-

doslučajan način. Svaka jedinka je struktura podataka koja predstavlja jedno moguće rešenje problema koji se rešava; poželjno je da što više tako predstavljenih rešenja bude dopustivo, ali to ne mora ili ne može uvek biti slučaj. Na koji način je jedinka, odnosno odgovarajuća struktura podataka, implementirana zavisi od samog problema. Često se primenjuje binarno kodiranje jedinki, ali kodiranje može biti i celobrojno ili mešovito, a mogu se koristiti i apstraktnije strukture podataka. Međutim, s obzirom da je efikasnost izvršavanja bitna odlika genetskih algoritama, binarno kodiranje obično vodi ka najefikasnijem pronalaženju rešenja. Broj jedinki u početnoj populaciji zavisi od implementacije i kreće se od svega nekoliko jedinki pa do više stotina jedinki. Veći broj jedinki znači i veću raznovrsnost genetskog materijala, ali može voditi sporijoj konvergenciji, kako po broju iteracija tako i po vremenu potrebnom za primenu genetskih operatora u svakoj iteraciji. Generisanje početne populacije je pseudoslučajno i ono može biti potpuno slobodno, tj. neusmereno ili usmereno, gde se pojedine vrednosti gena favorizuju u odnosu na druge. Usmereno generisanje početne populacije za cilj ima da ubrza konvergenciju, ali postoji opasnost da se smanji raznovrsnost genetskog materijala.

Potom se pristupa primeni evolutivnih procesa. Oni se primenjuju iterativno, dok se ne ispuni neki kriterijum zaustavljanja. Svaka iteracija predstavlja jednu generaciju populacije jedinki. Kako svaka jedinka predstavlja jedno rešenje problema, potrebno je odrediti kolika je vrednost tog rešenja i za to služi funkcija cilja. U svakoj generaciji prvo se računa vrednost funkcije cilja za svaku jedinku posebno, a zatim se primenjuju genetski operatori. Prvi je operator prilagođenosti (eng. fitness). Njegova uloga je da oceni vrednost funkcije cilja svake jedinke u poređenju sa drugima. Obično veća vrednost funkcije prilagođenosti znači da je jedinka bolje prilagođena. Ukoliko se rešava optimizacioni problem u kome je potrebno naći maksimalno moguće rešenje, onda se za prilagođenost jednostavno može uzeti vrednost same funkcije cilja. Često je korisno da prilagođenost pripada intervalu $[0, 1]$ i u te svrhe se za prilagođenost koristi normalizovana vrednost funkcije cilja: $p_i = \frac{c_i - min(c)}{max(c) - min(c)}$, gde je c vektor vrednosti funkcije cilja svih jedinki, a p vektor vrednosti funkcije prilagođenosti. To je samo jedan mogući vid ocene prilagođenosti neke jedinke; u zavisnosti od problema, funkcija prilagođenosti može biti i složenija, tako da, pored vrednosti c_i u obzir uzima i druge činioce, na primer, jedinstvenost genetskog materijala, standardnu devijaciju itd. Ukoliko rešenje koje

neka jedinka predstavlja nije dopustivo, obično joj se dodeljuje prilagođenost 0.

Zatim se primenjuje evolutivni operator selekcije. Kao što se to dešava i u prirodi, neke jedinke će proizvesti narednu generaciju a neke neće učestvovati u tom procesu. Operator selekcije po nekom kriterijumu bira jedinke koje će svoje gene preneti na potomstvo. Prosta selekcija, koja se naziva rulet selekcija, pseudoslučajno bira jedinke iz populacije uz verovatnoću koja je proporcionalna funkciji prilagođenosti. To je samo jedan mogući način da se izvrši selekcija. Često primenjivan operator selekcije je turnirska selekcija, koja simulira međusobnu borbu jedinki za učešće u stvaranju potomstva. Turnirska selekcija se realizuje tako što se sve jedinke pseudoslučajno podele u određeni broj grupa sa određenim brojem članova, a onda se iz svake grupe bira najbolja jedinka i ona prolazi selekciju. Tako izabrane jedinke stvaraju potomstvo putem ukrštanja.

Operator ukrštanja (eng. crossover) simulira proces razmene gena pri ćelijskoj deobi. Iz populacije koja je prošla selekciju, pseudoslučajno se biraju po dva „roditelja“ koji će proizvesti dve nove jedinke - „potomke“. Dva roditelja razmenjuju genetski materijal putem ukrštanja, koje može biti izvedeno na različite načine. Obično je u pitanju poziciono ukrštanje: odrede se pozicije u genetskom kodu svake jedinke i potomci se stvaraju tako što se prepišu geni jednog roditelja do prve pozicije a zatim drugog roditelja do sledeće pozicije ili do kraja zapisa, i tako naizmenično u zavisnosti od broja pozicija. Jednopoziciono ukrštanje je najjednostavnije. U tom slučaju će jedan potomak dobiti prvi deo genetskog koda od prvog roditelja, a drugi deo koda od drugog, a drugi potomak obrnuto. Ukrštanje može biti i dvopoziciono i višepoziciono. Moguće je i uniformno ukrštanje, pri čemu se najpre pseudoslučajno generiše binarna maska dužine jednake dužini celokupnog zapisa jedinke, a zatim se jedan potomak stvara tako što nule i jedinice u binarnoj maski određuju da li se na to mesto prepisuje gen prvog ili drugog roditelja, dok je kod drugog potomka obrnuto. Druge implementacije operatora ukrštanja su moguće ali se ređe koriste.

Operator mutacije simulira istoimeni proces koji se odigrava u molekulima DNK. Pri prostoj mutaciji, prvo se određuje verovatnoća da na nekom mestu u genetskom kodu dođe do mutacije, odnosno promene vrednosti tog gena i zatim se na osnovu te verovatnoće pseudoslučajno određuje hoće li do mutacije doći, za svaku jedinku posebno. U slučaju binarnog zapisa to dovodi do invertovanja bita u zapisu gena. Značaj operatora mutacije je veliki, jer povećava raznovrsnost genetskog materijala

i sprečava preranu konvergenciju ka lokalnom minimumu. Upravo zbog toga, često se primenjuje strategija takozvanih „zamrznutih gena“: ukoliko sve jedinke (ili veliki procenat njih) u populaciji imaju istu vrednost određenog gena, onda se verovatnoća da dođe do mutacije upravo tog gena povećava za određeni faktor. Pored proste mutacije, primenjuju se i drugi operatori mutacije, koji, na primer, koriste binomnu ili normalnu raspodelu verovatnoće da dođe do mutacije. Različiti operatori mutacije objašnjeni su u [20].

Konačno, kada evolutivni operatori stvore nove jedinke, novu generaciju moguće je izabrati na više načina. Određeni procenat će predstavljati nove jedinke a preostali stare. Često se primenjuje elitistička strategija, da jedinke sa najboljom vrednošću funkcije prilagođenosti uvek prelaze u novu generaciju. Nasuprot toj strategiji, primenjuje se strategija slučajnog izbora jedinki koje prelaze u novu generaciju. Konačno, moguće je i potpuno zameniti staru generaciju jedinki novim jedinkama.

Iterativni proces stvaranja novih generacija jedinki se nastavlja sve dok se ne ispuni neki od kriterijuma zaustavljanja genetskog algoritma. Mogući uslovi za prekid rada GA su: ukupan broj generacija veći od dozvoljenog, broj uzastopnih generacija bez popravljanja najboljeg rešenja veći od dozvoljenog, prekoračeno maksimalno vreme izvršavanja, prevelika sličnost jedinki u populaciji i drugi.

Iz prethodno izloženog se vidi da je prilikom implementacije GA neophodno ne samo izabrati odgovarajuće operatore, nego i da postoji puno parametara tih operatora čije vrednosti nisu univerzalne, već ih je potrebno odrediti za konkretan problem koji se rešava. Često se obave preliminarna testiranja sa različitim podešavanjima parametara na nekom podskupu test primera, a onda se ona podešavanja koja su dala najbolje rezultate primene na ceo skup testova. Takođe, uobičajeno je da se svaki test genetskog algoritma obavi više puta sa različitim vrednostima ulaznog parametra generatora pseudoslučajnih brojeva, jer to značajno utiče na generisanje početne populacije jedinki. Stabilnost genetskog algoritma se meri standardnom devijacijom od najboljeg postignutog rešenja pri takvom višestrukom testiranju.

U savremenim primenama genetskih algoritama, sve češće se koristi njihova hibridizacija sa nekom drugom heuristikom. Takve hibridne metode se nazivaju memetski algoritmi i u njima se pojedine jedinke iz genetskog algoritma prosleđuju heurističkoj metodi koja pokušava da poboljša to rešenje, pretražujući njegovu okolinu i težeći ka lokalnom minimumu. U pitanju može biti metaheuristička metoda ili

metoda koja je specifična za zadati problem.

1.3.2 Lokalna pretraga i memetski algoritmi

Lokalna pretraga u širem smislu se realizuje preko metode koja polazi od nekog rešenja i traži nova rešenja u okolini tog rešenja sa ciljem nalaženja boljeg.

Lokalna pretraga je takođe i termin koji se koristi za grupu metaheuristika koje se zasnivaju na principu pretraživanja okoline nekog rešenja. U njih spadaju: metoda promenljivih okolina, tabu pretraga, iterativna lokalna pretraga itd.

U slučaju memetskog algoritma, jedinke iz genetskog algoritma se prosleđuju nekoj od metoda lokalne pretrage, koje mogu biti:

- Metaheuristička lokalna pretraga;
- Neka druga metaheuristika;
- Algoritam lokalne pretrage specifičan za problem koji se rešava. Koristeći neka inherentna svojstva objekata, koji se javljaju u zadatom problemu, dizajnira se algoritam koji menja genetski kôd jedinke, tako da se očekuje da nova jedinka predstavlja bolje rešenje od polaznog;
- Neka metoda koja za zadato parcijalno rešenje problema pronalazi celo rešenje koje ga sadrži. Na primer, kod nekih hab lokacijskih problema može se u polinomskom vremenu naći optimalno rešenje za zadati skup habova putem dinamičkog programiranja. U drugim slučajevima se može koristiti metoda grananja i ograničavanja, ili matematička metaheuristika kao što je Lagranževa relaksacija ili čak i univerzalni rešavači problema matematičkog programiranja.

Kod memetskih algoritama najveća prednost je ubrzanje konvergencije i postizanje lokalnih minimuma koje bi inače bilo teško ili nemoguće dostići. S druge strane, najveća opasnost je da preveliki broj jedinki prebrzo konvergira ka istom lokalnom minimumu, koji može biti dalekog od globalnog minimuma, i da se na taj način smanji raznovrsnost genetskog materijala i onemogući dalja konvergencija.

1.4 Metoda grananja i ograničavanja

Metoda grananja i ograničavanja (eng. branch and bound; skr. BnB) spada u grupu konstruktivnih metoda matematičke optimizacije - rešenje problema se postupno

konstruiše tako što se ispituju moguće vrednosti jedne po jedne promenljive, sve dok se ne konstruiše celo rešenje. Pretpostavimo da se radi o problemu minimizacije. Metoda se realizuje pomoću dve ključne funkcije:

- Funkcija Grananje (eng. branch) služi da domen iz kojih neka promenljiva uzima vrednosti podeli u podskupove. U slučaju binarne promenljive to su uvek dve grane - 0 i 1. U slučaju kontinualnih promenljiivih, domen se deli na konačan broj intervala. U slučaju diskretnih promenljivih podela je moguća ili na intervale ili na konkretan konačan skup dopustivih vrednosti.

- Granica (eng. bound) je funkcija koja određuje donju granicu bilo kog rešenja koje sadrži neko parcijalno rešenje R.

Algoritam 2 predstavlja rekurzivnu verziju BnB algoritma za slučaj kada promenljive uzimaju vrednosti iz konačnog diskretnog skupa. Moguća je i ekvivalentna iterativna implementacija koja koristi pomoćnu strukturu podataka, na koju se stavljaju i sa koje se uzimaju vrednosti promenljivih. U slučaju podele domena na intervale, kada intervali postanu dovoljno mali, neophodno je da vrednosti promenljivih dobiju diskretne vrednosti, te se implementira na sličan način.

Algoritam 2: Osnovna varijanta algoritma grananja i ograničavanja - BnB

ulaz : Parcijalno rešenje R
izlaz: Rešenje M (globalna promenljiva)
odabrati narednu promenljivu $P \notin R$;
$G = Grananje(R, P)$;
foreach $g \in G$ **do**
  $S = R \cup \{P_g\}$;
  if *S je celo rešenje* **then**
    if $Vrednost(S) < Vrednost(M)$ **then**
      $M = S$;
    end
  end
  else if $Granica(S) < Vrednost(M)$ **then**
    $BnB(S)$;
  end
end

Algoritam BnB koristi globalnu promenljivu M koja u svakom trenutku rada algoritma ima vrednost najboljeg pronađenog rešenja. Funkcija $Vrednost(M)$ daje vrednost funkcije koju je potrebno minimizovati, za dato rešenje M. Na početku

je $M = \emptyset$ i $Vrednost(M) = +\infty$. Algoritam ima jedan ulazni parametar koji predstavlja parcijalno rešenje koje je konstruisano do tog trenutka. Zato se algoritam poziva sa $BnB(\emptyset)$.

Prvi korak je izbor naredne promenljive po čijim vrednostima će se vršiti grananje. Redosled biranja promenljivih je jako bitan za brzinu rada algoritma. Izbor naredne promenljive zavisi od samog problema i ne postoji univerzalan princip koji bi vodio optimalnoj brzini rada algoritma. Zatim se vrši grananje po dopustivim vrednostima izabrane promenljive P za trenutno parcijalno rešenje R. Za svaku takvu vrednost g, odnosno „granu“, formira se novo rešenje S, tako što se promenljivoj P dodeli vrednog g (u oznaci P_g) i pridoda na postojeće parcijalno rešenje R. Ukoliko je S celo rešenje problema, u kome su sve promenljive fiksirane, računa se vrednost tog rešenja i upoređuje sa najboljom poznatom vrednošću. Ukoliko S ima bolju vrednost od M, tada najbolje poznato rešenje postaje S, odnosno $M = S$. Ukoliko S i dalje predstavlja parcijalno rešenje, proverava se donja granica tog parcijalnog rešenja i samo ukoliko je ona manja od vrednosti rešenja M, pristupa se rekurziji gde se isti algoritam poziva, ali sa parcijalnim rešenjem S kao parametrom.

Jasno je da je, u opštem slučaju, složenost ovog algoritma eksponencijalna, odnosno jednaka je proizvodu kardinalnosti svih promenljivih. Na primer, ukoliko su sve promenljive binarne i ima ih n, složenost algoritma BnB je $O(2^n)$. Efikasnost prikazanog algoritma zavisi od više faktora:

- Za odsecanje grana ključna je funkcija $Granica(S)$, odnosno što bolje određivanje donje granice za dato parcijalno rešenje S. Jedna univerzalna mogućnost, koja se često primenjuje, jeste da se kao donja granica uzima vrednost rešenja linearne relaksacije problema matematičkog programiranja (skraćeno LP-rešenje od eng. linear programming).

- Prilikom izvršavanja algoritma, implicitno se gradi stablo pretraživanja (eng. search tree). Da bi se to stablo sporije širilo, korisno je prvo birati promenljive P sa malim brojem mogućih vrednosti, međutim, to nije univerzalno pravilo.

- Efikasnost algoritma takođe zavisi od redosleda kojim se obilaze grane $g \in G$. Ako se u prvim koracima prvo izaberu „loše“ grane, odnosno vrednosti promenljivih koje vode ka rešenjima koja su daleko od optimalnog, puno vre-

mena će se provesti u pretraživanju dela stabla sa puno takvih „loših“ rešenja, dok se najbolje rešenje zapravo krije u drugom delu stabla koje će, u tom slučaju, biti pretraživano pri kraju rada algoritma. Sa druge strane, ukoliko bi se u svakom koraku uvek prvo obilazile grane koje vode ka optimalnom rešenju, onda bi nakon pronalaska tog rešenja algoritam jako efikasno vršio odsecanja grana putem uslova $Granica(S) < Vrednost(M)$.

- Ukoliko je pre pokretanja algortima BnB poznato neko „dobro“ rešenje M, neke grane će ranije biti odsečene. Zato je korisno pronaći polazno rešenje M, na primer nekom heuristikom, a zatim ga popravljati algoritmom BnB. Tada će do odsecanja doći mnogo ranije i ona „loša“ rešenja neće biti razmatrana.

Gotovo uvek algoritam BnB pronalazi neko rešenje u vremenu O(n), gde je n broj promenljivih. To nije slučaj onda kada su vrednosti promenljivih jako zavisne jedna od drugih. U tom slučaju, za dato parcijalno rešenje R i izabranu promenljivu P, funkcija $Grananje(R, P)$ može vratiti prazan skup, jer ne postoji nijedna vrednost promenljive P koja bi vodila dopustivom rešenju. Tada algoritam mora da se vraća unazad (eng. backtracking) i traži nove vrednosti prethodno izabranih promenljivih.

Metoda grananja i ograničavanja se može koristiti i kao heuristika, tako da pronalazi dobra dopustiva rešenja, bez iscrpljivanja svih mogućnosti i dokazivanja optimalnosti. To se postiže na neki od sledećih načina:

- Prevremen prekid izvršavanja algoritma BnB. Na primer, moguće je završiti pretragu nakon određenog vremena ili nakon određenog broja čvorova u stablu ili na zahtev korisnika.

- Ograničiti broj grana koje se pretražuju u svakom koraku. Na primer, nakon poziva funkcije $Grananje$, na neki način proceniti koje vrednosti promenljive su perspektivne, odnosno potencijalno vode najboljim rešenjima i ograničiti pretragu na nekoliko najboljih takvih vrednosti a ostale vrednosti ignorisati.

- Umesto uslova $Granica(S) < Vrednost(M)$ koristi se uslov $Vrednost(M) - Granica(S) > \epsilon$, gde je ϵ zadata tačnost. Na taj način bi se dobilo rešenje za koje se zna da se nalazi u ϵ-okolini optimalnog rešenja.

Jedna implementacija paralelizovanog BnB algoritma prikazana je u radu [64]. O metodama i primenama metode grananja i ograničavanja može se detaljnije naći u [42].

Poglavlje 2

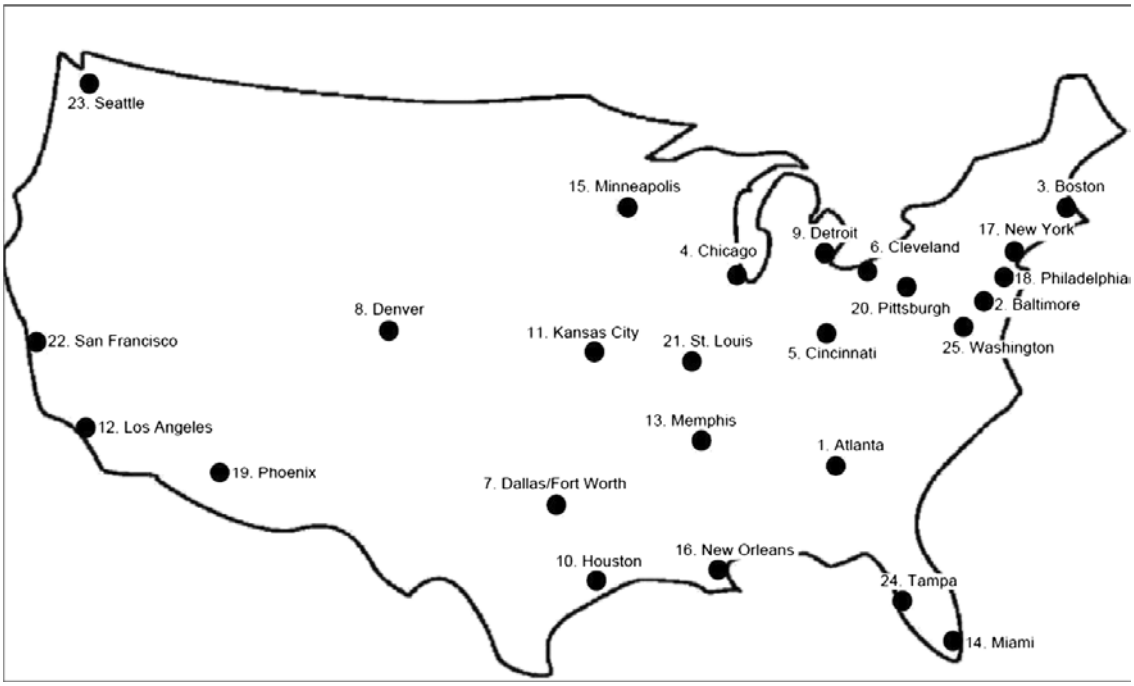
Lokacijski problemi

Hab lokacijski problemi matematičke optimizacije, koji pored lociranja objekata uključuju i takozvanu alokaciju, odnosno pridruživanje korisnika uspostavljenim habovima, brojni su i raznovrsni. Cilj je izvršiti lokaciju i alokaciju tako da funkcija koja predstavlja sumu određenih troškova ima minimalnu vrednost i da su pri tom zadovoljeni neki uslovi. Iako su u pitanju međusobno vrlo srodni i naizgled slični problemi, metode za rešavanje tih problema su po pravilu prilagođene samo jednom konkretnom problemu i ne postoji univerzalna metoda. Detaljan pregled hab lokacijskih problema i metoda za njihovo rešavanje može se videti u [38], [11] i [5]. Većina tih optimizacionih problema pripada klasi $\mathbb{NP}$-teških, za šta postoje dokazi, na primer u [35] i [19].

U ovom poglavlju biće razmatrana tri hab lokacijska problema i navedeni odgovarajući matematički modeli.

2.1 Hab lokacijski problem sa jednostrukom alokacijom bez ograničenja kapaciteta

Jedan od najviše proučavanih hab lokacijskih problema je hab lokacijski problem sa jednostrukom alokacijom bez ograničenja kapaciteta (eng. Uncapacitated Single Allocation Hub Location Problem, skr. USAHLP). Problem je inspirisan potrebom da se optimizuje protok putnika u američkom aviosaobraćaju. Prvu matematičku formulaciju problema USAHLP dao je O'Keli 1987. godine, koji je i dokazao da je problem $\mathbb{NP}$-težak [53]. Formulacija je testirana na skupu test primera koji sadrži 25 američkih gradova (slika 2.1). Test primere je objavio isti autor 1986. godine



Slika 2.1: 25 američkih gradova iz CAB test primera [52] (preuzeto iz [55])

[52] i oni su, u proširenom izdanju sa 50 gradova, korišćeni za testiranje metoda za rešavanje mnogobrojnih hab lokacijskih problema, pod nazivom CAB test primeri (eng. Civil Aeronautics Board data set).

Problem je definisan na sledeći način:

- Dat je protok (količina) saobraćaja između svaka dva čvora neke mreže.
- Saobraćaj između čvorova ne ide direktno, već se usmerava preko habova i sastoji se iz tri deonice: prikupljanje (eng. collection) od polaznog čvora do haba, transfer od jednog haba do drugog i distribucija od drugog haba do odredišnog čvora.
- Potrebno je odrediti koji od čvorova će predstavljati habove (lokacija). Habova može biti proizvoljan broj, ali su dati fiksni troškovi uspostavljanja haba u nekom čvoru.
- Takođe, treba izvršiti alokaciju ne-hab čvorova habovima. Alokacija je jednostruka, što znači da se saobraćaj iz jednog čvora uvek usmerava preko istog haba, bez obzira na destinaciju. Distribucija ka nekom čvoru se vrši uvek iz istog haba, bez obzira iz kog čvora taj saobraćaj potiče.

- Saobraćaj koji potiče iz nekog haba se prikuplja u istom tom habu a transfer i distribucija se obavljaju na isti način.
- Ne postoji ograničenje kapaciteta, odnosno količine saobraćaja koji može biti usmeren ka nekom habu (eng. uncapacitated).
- Lokaciju i alokaciju treba izvršiti tako da ukupna suma troškova, koja se sastoji od fiksnih troškova uspostavljanja habova, zatim troškova prikupljanja, transfera i distribucije, bude minimalna.

Matematički model je formulisan pomoću problema mešovitog celobrojnog linearnog programiranja (eng. mixed integer linear programming, skr. MILP), korišćenjem sledećih ulaznih parametara:

- Mreža I koja sadrži n čvorova, $|I| = n$;
- Matrica C_{ij} - troškovi transporta između bilo koja dva čvora $i, j \in I$. $C_{ij} = C_{ji}$ i $C_{ii} = 0$;
- Matrica W_{ij} - količina saobraćaja koja polazi iz čvora $i \in I$ a za odredište ima čvor $j \in I, W_{ij} \geq 0$; Matrica W ne mora biti simetrična i W_{ii} ne mora biti 0;
- Vektor F_i - fiksni troškovi za uspostavljanje habova, za svaki čvor $i \in I$;
- $\chi, \alpha, \delta \in \mathbb{R}$ - težinski faktori kojima se određuje cena prikupljanja (od čvora do haba), transfera (od haba do haba) i distribucije (od haba do odredišta).

Lokacija i alokacija u mreži su realizovani tako da sav tok saobraćaja, od polaznog čvora i do odredišnog čvora j, mora biti usmeren preko nekih habova, redom, k i l $\in H \subseteq I$, gde skup H predstavlja uspostavljene habove. Bitna pretpostavka je da direktan saobraćaj između dva ne-hab čvora nije dozvoljen već on uvek mora biti usmeren preko habova.

USAHLP je problem sa jednostrukom alokacijom, što znači da saobraćaj iz jednog čvora uvek mora biti usmeren preko istog haba, bez obzira na odredište. Za neki čvor i, oznaka $hab(i)$ predstavlja hab kom je taj čvor pridružen u prethodnom smislu, odnosno alociran. Podrazumeva se da je svaki hab alociran sam sebi.

Cena protoka saobraćaja od polazišta i do odredišta j je data sa $cena(i, j) = W_{ij}(\chi C_{ik} + \alpha C_{kl} + \delta C_{lj})$, gde je i pridružen habu k, odnosno $k = hab(i)$ i, slično, j je pridružen habu $l = hab(j)$.

Težinski faktori χ, α i δ treba da imaju takve vrednosti da opravdavaju postojanje habova i usmeravanje saobraćaja preko njih. To praktično znači da cena transfera mora biti manja od cene prikupljanja i distribucije, odnosno $\alpha \leq \chi, \delta$.

Ukupna cena svih troškova u mreži I, sa uspostavljenim habovima $H \subseteq I$, je data sledećom jednakošću:

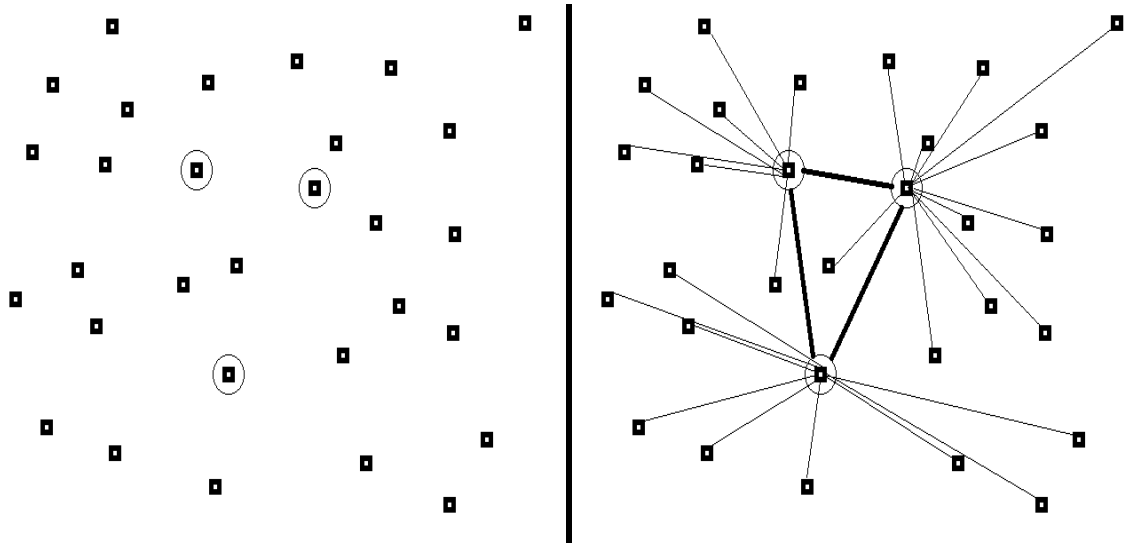
$$\begin{aligned} ukupno &= \sum_{h \in H} F_h + \sum_i \sum_j cena(i,j) = \sum_{h \in H} F_h + \sum_i \sum_j W_{ij}(\chi C_{ik} + \alpha C_{kl} + \delta C_{lj}) \\ &= \sum_{h \in H} F_h + \sum_i \sum_j W_{ij}\chi C_{ik} + \sum_i \sum_j W_{ij}\alpha C_{kl} + \sum_i \sum_j W_{ij}\delta C_{lj} \\ &= \sum_{h \in H} F_h + \chi \sum_i O_i C_{ik} + \alpha \sum_i \sum_j W_{ij} C_{kl} + \delta \sum_j D_j C_{lj} \\ &= \sum_{h \in H} F_h + \sum_i (\chi O_i + \delta D_i) C_{ik} + \alpha \sum_i \sum_j W_{ij} C_{kl} \end{aligned} \tag{2.1}$$

gde je $O_i = \sum_j W_{ij}$ ukupna količina saobraćaja koji polazi iz čvora i, a $D_j = \sum_i W_{ij}$ je ukupna količina saobraćaja koja za odredište ima čvor j, i važi $k = hab(i) \in H$ i $l = hab(j) \in H$. U poslednjoj jednakosti, prva suma je suma fiksnih troškova uspostavljanja habova, druga suma predstavlja troškove prikupljanja i distribucije a poslednja, dvostruka, suma predstavlja troškove transporta između habova.

Na slici 2.2 dat je primer mreže sa 30 čvorova i jedno dopustivo rešenje. U prvom delu slike, tri čvora su zaokružena, što označava da je izvršena lokacija, odnosno da su u njima uspostavljeni habovi. Zatim su preostali čvorovi pridruženi habovima, svaki tačno jednom habu, odnosno, izvršena je alokacija. Protok saobraćaja polazi od svakog ne-hab čvora i ide do njemu pridruženog haba, što je označeno tankom linijom, zatim se vrši transfer do drugog haba (deblja linija) i, na kraju, distribucija od drugog haba do ne-hab čvora (tanka linija).

2.1.1 Matematička formulacija problema

Potrebno je odrediti skup habova $H \subseteq I$ i odrediti alokaciju $i \in I \rightarrow hab(i) \in H$ tako da suma troškova u jednakosti (2.1) bude minimalna. Ukoliko bismo koristili binarne promenljive da odredimo koji čvorovi su habovi i koji čvor je pridružen kom habu, poslednja dvostruka suma u (2.1) bi zahtevala n^4 binarnih promenljivih.



Slika 2.2: Lokacija, alokacija i protok saobraćaja u mreži od 30 čvorova

Upravo takav je bio prvi model koji je definisao O'Keli [53]. Ernst i Krišnamurti su 1999. godine [30] predložili formulaciju koja koristi n^2 binarnih promenljivih za prve dve sume u poslednjoj jednakosti (2.1) i n^3 realnih promenljivih za poslednju sumu.

MILP koji će sada biti predstavljen zasniva se na modelu Ernsta i Krišnamurtija, ali sadrži jedan novi skup uslova koje je dodao Koreja 2010. godine [18]. Ti uslovi se od tada nazivaju „uslovima koji su nedostajali", jer su autori pokazali da bez tih uslova model nije kompletan i nije ekvivalentan O'Kelijevom modelu i da bi rešenja, dobijena bez tih uslova, mogla biti nedopustiva.

U MILP problemu postoji n^2 binarnih promenljivih $Z_{ij}, i, j \in I$, takvih da $Z_{ij} = 1 \Leftrightarrow i \vdash j \Leftrightarrow hab(i) = j$, što znači da je j hab i da je čvor i alociran habu j. Takođe, $Z_{kk} = 1 \Leftrightarrow k \in H \Leftrightarrow k$ je hab, jer habovi moraju biti alocirani svaki samom sebi.

Korišćenjem samo Z promenljivih, Ernst je prilagodio jednakost (2.1) i definisao funkciju cilja, što je rezultiralo kvadratnim problemom:

$$\min \sum_{i \in I} F_i Z_{ii} + \sum_i \sum_k \sum_j \sum_l W_{ij}(\chi C_{ik} + \alpha C_{kl} + \delta C_{lj}) Z_{ik} Z_{jl} \tag{2.2}$$

Da bi se dobio linearan problem, svaki proizvod $Z_{ik}Z_{jl}$ bi trebalo zameniti jednom binarnom promenljivom, što znači da bi ih bilo n^4. Umesto toga, koristi se n^3 realnih, nenegativnih promenljivih Y^i_{kl} koje služe za računanje troškova transfera. Iskorišćena je činjenica da je sav saobraćaj iz jednog čvora uvek usmeren preko istog haba, bez

obzira na odredište. Time je omogućeno smanjenje broja promenljivih za faktor n. Svaka od tih nenegativnih Y_{kl}^i promenljivih predstavlja ukupnu količinu saobraćaja koja polazi iz čvora i, prikupljenu u habu k i distribuiranu preko haba l.

Matematički model problema USAHLP je sledeći MILP problem, definisan kao u [18]:

$$\min \sum_{i\in I}\sum_{k\in I} C_{ik}Z_{ik}(\chi O_i + \delta D_i) + \sum_{i\in I}\sum_{k\in I}\sum_{l\in I} \alpha C_{kl}Y_{kl}^i + \sum_{k\in I} f_k Z_{kk} \tag{2.3}$$

tako da:

$$\sum_{k\in I} Z_{ik} = 1 \quad \forall i \in I \tag{2.4}$$

$$Z_{ik} \leq Z_{kk} \quad \forall i,k \in I \tag{2.5}$$

$$\sum_{l\in I} Y_{kl}^i - \sum_{l\in I} Y_{lk}^i = O_i Z_{ik} - \sum_{j\in I} W_{ij}Z_{jk} \quad \forall i,k \in I \tag{2.6}$$

$$\sum_{l\in I, l\neq k} Y_{kl}^i \leq O_i Z_{ik} \quad \forall i,k \in I \tag{2.7}$$

$$Z_{ik} \in \{0,1\} \quad \forall i,k \in I. \tag{2.8}$$

$$Y_{kl}^i \geq 0 \quad \forall i,k,l \in I \tag{2.9}$$

Funkcija cilja (2.3) se razlikuje od (2.2) u načinu na koji se računaju troškovi transfera, ali obe funkcije daju istu vrednost za određenu lokaciju i alokaciju ([30], [18]). Skup uslova (2.4) obezbeđuje jednostruku alokaciju: svaki čvor je alociran tačno jednom habu. Uslovi (2.5) obezbeđuju da je svaki hab alociran samom sebi i takođe obezbeđuju da ne-hab čvor može biti alociran samo uspostavljenom habu. Skupom uslova (2.6) su date jednakosti održanja protoka u mreži i on služi da odredi vrednost Y promenljivih. Pomenuti „uslovi koji su nedostajali“ a koje je definisao Koreja dati su nejednakostima (2.7). Konacno, (2.8) i (2.9) definišu promenljive ovog problema.

Problem mešovitog celobrojnog linearnog programiranja za USAHLP sadrži

ukupno $n^3 + n^2$ promenljivih i $O(n^2)$ uslova.

2.1.2 Pregled metoda za rešavanje problema

Mnogi autori su se bavili rešavanjem USAHLP problema korišćenjem različitih heurističkih i metaheurističkih metoda:

- U istom radu u kom je definisao problem, O'Keli ga je rešavao heuristikom koja se zasniva na totalnoj enumeraciji.
- Zatim je Klincewicz 1991. [39] razvio heuristiku koja se sastoji od grupisanja i razmene (eng. clustering and exchange). Isti autor je 1992. godine razvio dve metaheurističke metode: GRASP i tabu pretragu [40].
- Tabu pretragom je problem ponovo rešavao O'Keli 1992. godine [54].
- 1994. godine Skorin-Kapov i ostali su takođe koristili tabu pretragu [60].
- Abdinnour-Helm je 1998. godine je prvo razvio BnB metodu koja nije bila dovoljno efikasna, pored nje i GA [3] a zatim i hibridnu metodu koja se sastoji iz GA i tabu pretrage [2].
- Genetski algoritam za USAHLP je razvio i Topkuoglu 2005. godine. [67]
- 2007. godine je Čen primenio hibridnu metodu koja se sastoji od simuliranog kaljenja, tabu pretrage i lokalne pretrage. [13]
- 2009. godine su Silva i Kunja [59] generisali nove instance do dimenzije 400 i uspešno ih primenili u rešavanju problema pomoću dve metode: multi start tabu pretragom i dvofaznom tabu pretragom.
- Filipović i ostali su takođe 2009. godine problem USAHLP rešavali pomoću GA i totalne enumeracije za alokaciju [33]. Ovaj i prethodni rad su postigli najbolje rezultate i smatra se da su te metode u tom trentuku bile najefikasnije poznate metode za rešavanje USAHLP.
- U radu [58] autori su predstavili metodu za određivanje donje granice ciljne funkcije zasnovanu na Lagranžeovoj relaksaciji.

- Diskretna verzija optimizacije pomoću roja čestica za USAHLP je predstavljena u radu [6].
- USAHLP i srodne probleme autori rada [49] su rešavali linearizacijom, koristeći činjenicu da su u test primerima troškovi transporta zapravo euklidske udaljenosti u ravni.
- Jedan od najnovijih radova na temu USAHLP je rad [4] u kome je predstavljena još jedna verzija tabu pretrage za ovaj problem.

2.2 Lokacijski problem snabdevača neograničenog kapaciteta u više nivoa

Jedan od osnovnih lokacijskih problema u kojem se razmatra pitanje snabdevanja klijenata od strane snabdevača jeste problem lokacije snabdevača (eng. simple plant location problem - SPLP). Njegovo uopštenje (videti [24]) je lokacijski problem snabdevača neograničenog kapaciteta u više nivoa (eng. Multi Level Uncapacitated Facility Location Problem-MLUFLP), koji se od SPLP razlikuje po tome što se snabdevači nalaze na nekoliko nivoa. U pitanju je hijerarhijski problem - data je mreža snabdevača i između njih je utvrđena određena hijerarhija: snabdevači na različitim nivoima nisu potpuno nezavisni već postoji neka saradnja koja se odvija među njima. S obzirom da je SPLP $\mathbb{NP}$-težak [41], jasno je da i MLUFLP pripada istoj klasi.

MLUFLP modeluje stvarne probleme infrastrukturnog tipa koji se sreću u javnom sektoru. Pod pojmom snabdevača podrazumevaju se, između ostalog: elektrodistribucije, zdravstvene ili obrazovne ustanove, banke, kanali isporuke, poštanske mreže, skladišta, fabrike i sl. Svi snabdevači imaju neke zajedničke karakteristike. Međutim, snabdevači na različitim nivoima imaju određena specifična svojstva koja snabdevači na drugim nivoima nemaju. Upravo po tim svojstvima su snabdevači i grupisani u nivoe. Na primer, u slučaju zdravstvenih ustanova, „snabdevači“ jednog nivoa su ambulante, drugog nivoa bolnice a trećeg klinički centri. Pored snabdevača, drugi važan deo ovog problema predstavljaju korisnici usluga koje snabdevači pružaju. Cilj optimizacije problema MLUFLP jeste da se odrede lokacije snabdevača na svim nivoima i da se odredi hijerarhijska mreža lociranih snabdevača, tako da postoji putanja od svakog korisnika do nekog lociranog snabdevača na svakom nivou i da pri tom ukupni troškovi lociranja snabdevača i snabdevanja korisnika budu minimalni.

Ulazni parametri svake instance MLUFLP su sledeći:

- Skup snabdevača F ($|F| = m$),
- Podela skupa F na k podskupova (nivoa) $F_1, \ldots, F_k$,
- Skup klijenata D ($|D| = n$),
- Troškovi transporta c_{ij} između svaka dva snabdevača na susednim nivoima i

svakog snabdevača na nivou k i svakog klijenta. Dakle, (i, j) uzima vrednost iz skupa $\bigcup_{l=1}^{k-1}(F_{l+1} \times F_l) \bigcup (D \times F_k)$,

- Fiksni troškovi za uspostavljanje svakog od snabdevača f_i, $i \in F$.

Svako dopustivo rešenje se sastoji od uspostavljanja snabdevača na svim nivoima (lokacija) i utvrđivanja putanje snabdevanja za svakog klijenta (alokacija). Ta putanja polazi od klijenta i ide do jednog od uspostavljenih snabdevača na nivou k, pa do jednog od uspostavljenih snabdevača na nivou $k-1$ i tako do nivoa 1.

Za dato dopustivo rešenje utvrđuje se suma troškova koja se sastoji od fiksnih troškova i troškova transporta. Troškovi transporta jednaki su zbiru svih troškova transporta u putanji svakog od klijenata. Cilj optimizacije MLUFLP je naći ono dopustivo rešenje koje ima najmanje ukupne troškove.

Na primeru to izgleda ovako. Neka se MLUFLP mreža sastoji od $n = 10$ klijenata i $m = 8$ potencijalnih snabdevača na $k = 2$ nivoa. Prvi nivo F_1 sadrži lokacije 3 potencijalna snabdevača a drugi nivo F_2 čini 5 snabdevača. Fiksni troškovi uspostavljanja lokacija svakog od potencijalnih snabdevača su: $f_1 = 3, f_2 = 2, f_3 = 2, f_4 = 4, f_5 = 4, f_6 = 1, f_7 = 1, f_8 = 1$. Bitan faktor svake MLUFLP mreže čine troškovi transporta između snabdevača na različitim nivoima i troškovi između snabdevača na poslednjem nivou (u ovom slučaju nivou 2) i klijenata. Ti troškovi su dati u tabelama 2.1 i 2.2.

Tabela 2.1: Troškovi transporta između snabdevača prvog i drugog nivoa

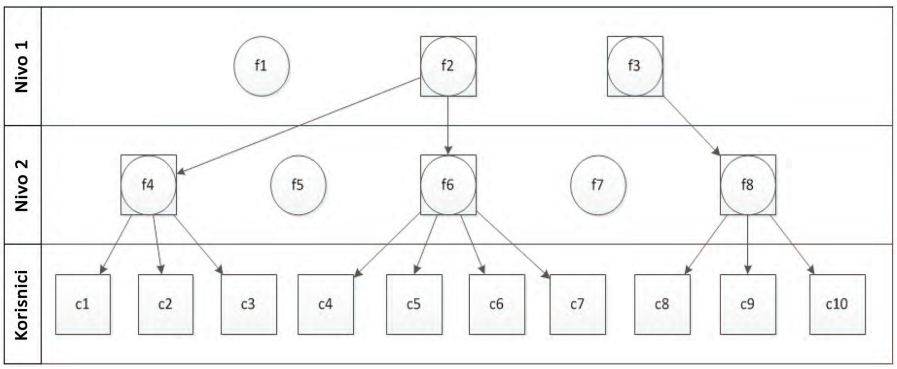
	f1	**f2**	**f3**
f4	9	6	8
f5	8	6	5
f6	9	5	9
f7	7	7	7
f8	8	9	5

Optimalno rešenje ovog problema dato je na slici 2.3. Snabdevači su uspostavljeni na lokacijama $f2, f3, f4, f6$ i $f8$, a putanje između klijenata i snabdevača su označene. Ukupni troškovi ovog, optimalnog, rešenja su 119.

Teoretskim aspektima rešavanja MLUFLP problema, zatim razvojem i modifikacijama matematičkog modela i metaheurističkim metodama za rešavanje ovog i njemu srodnih problema bavili su se mnogi autori u sledećim radovima, knjigama i doktorskim tezama:

Tabela 2.2: Troškovi transporta između snabdevača drugog nivoa i korisnika

	f4	**f5**	**f6**	**f7**	**f8**
c1	5	6	6	8	7
c2	5	9	9	7	9
c3	5	6	8	9	7
c4	7	6	5	6	8
c5	6	5	6	7	7
c6	7	8	5	8	8
c7	9	6	6	7	7
c8	6	7	7	8	5
c9	9	5	8	8	6
c10	8	8	9	9	8



Slika 2.3: Optimalno rešenje u mreži sa 2 nivoa, 8 snabdevača i 10 korisnika

- Aardal i ostali su predložili aproksimativni algoritam sa konstantnim faktorom za problem snabdevača na k-nivoa [1].
- Chudak i Williamson su na sličan način rešavali verziju problema sa kapacitetima [15], dok su Chudak i Shmoys na taj način rešavali verziju bez kapaciteta [14].
- Charikar i Guha su kombinatornim algoritmom rešavali problem snabdevača i problem k-medijane [12].
- Edwards je 2001. godine aproksimativnim algoritmom rešavao MLUFLP [27] a 2010. godine su novi aproksimativni algoritam predložili Gabor i van Ommeren.
- Galvao i ostali su predstavili hijerarhijski model za zdravstvene ustanove u Brazilu [34].
- Marić je 2012. godine MLUFLP rešavao pomoću genetskog algoritma [44].

2.2.1 Matematička formulacija problema

U novijim radovima iz ove oblasti koriste se tri matematička modela MLUFLP, koji se međusobno razlikuju po broju i tipu promenljivih i po broju uslova, ali sva tri su međusobno ekvivalentna. Dokaz ekvivalencije prikazan je u radu [47], a ovde će biti predstavljen model koji je nastao tokom istraživanja koje je rezultiralo tim radom.

Model je napravljen nad dva skupa promenljivih. Prvo, binarne promenljive $y_i \in \{0,1\}, i \in F$ služe da odrede da li je na lokaciji i uspostavljen snabdevač ili ne. Drugo, koriste se i celobrojne, nenegativne promenljive $z_{is}^l \geq 0$ koje zavise od promenljivih $y_i, i \in F$. Zarad jednostavnijeg pisanja formula kojima se računaju troškovi transporta, skup klijenata D se posmatra kao nov nivo, odnosno definiše se $F_{k+1} \equiv D$. Svaka promenljiva $z_{is}^l \geq 0$ predstavlja broj klijenata iz skupa $F_{k+1} \equiv D$ koji se snabdevaju putanjom između dva susedna nivoa i i s, gde $i \in F_l$ i $s \in F_{l-1}$, i pri tom $l = 2, .., k+1$. Dakle, gornji indeks l svake promenljive z određuje nivo na kome se utvrđuje koliko klijenata se snabdeva putanjom (i, s) koja određuje donji indeks promenljive z, gde je i neki od elemenata skupa F upravo sa tog nivoa l, a s je element sa prethodnog nivoa. Primetimo da l može imati i vrednost $k+1$, što znači da su time obuhvaćeni i elementi skupa $F_{k+1} \equiv D$, odnosno klijenti.

MLUFLP problem mešovitog celobrojnog linearnog programiranja definisan je kao u [47]:

$$\min \sum_{i=1}^{m} f_i y_i + \sum_{l=2}^{k+1} \sum_{i \in F_l} \sum_{s \in F_{l-1}} c_{is} z_{is}^{l} \tag{2.10}$$

tako da:

$$\sum_{i \in F_k} z_{ji}^{k} = 1 \ , \qquad \forall j \in D \equiv F_{k+1} \tag{2.11}$$

$$\sum_{s \in F_{l-1}} z_{is}^{l} = \sum_{r \in F_{l+1}} z_{ri}^{l+1} \ , \qquad \forall i \in F_l, \ l = 2, ..., k, \tag{2.12}$$

$$z_{ri}^{l+1} \leq n y_i \ , \qquad \forall i \in F_l, \ r \in F_{l+1}, \ l = 1, ..., k, \tag{2.13}$$

$$z_{is}^{l} \in \mathbb{N} \cup \{0\} \ , \qquad \forall l = 2, ..., k+1, \ i \in F_l, \ s \in F_{l-1}, \tag{2.14}$$

$$y_i \in \{0, 1\} \ , \qquad \forall i \in F. \tag{2.15}$$

Funkcija cilja (2.10) predstavlja zbir fiksnih troškova za uspostavljanje snabdevača i ukupnih troškova transporta. Uslovi (2.11) znače da je svaki klijent snabdeven od jednog snabdevača sa nivoa k. Uslovi (2.12) predstavljaju „tok“ snabdevanja, odnosno, oni obezbeđuju da je za svakog snabdevača i na nivou l, ulazni broj klijenata jednak izlaznom. Klijenti mogu biti snabdevani samo od strane uspostavljenih snabdevača što obezbeđuju uslovi (2.13). Konačno, uslovi (2.14) i (2.15) definišu promenljive ovog problema.

2.3 Hab lokacijski problem sa jednostrukom alokacijom i sa ograničenjem kapaciteta

Varijanta USAHLP problema, u kojoj postoji ograničenje kapaciteta saobraćaja za svaki potencijalni hab, naziva se hab lokacijski problem sa jednostrukom alokacijom i sa ograničenjem kapaciteta (eng. Capacitated Single Allocation Hub Location Problem-CSAHLP).

Funkcija cilja problema CSAHLP je ista kao kod USAHLP. Ulazni parametri su prošireni jednim dodatnim vektorom G, kojim se određuje kapacitet saobraćaja G_i koji može biti preusmeren preko potencijalnog haba $i \in I$.

2.3.1 Matematička formulacija problema

Matematički model za problem CSAHLP jednak je modelu za USAHLP u koji su autori rada [30] dodali sledeći skup uslova:

$$\sum_{i \in I} O_i Z_{ik} \leq G_k Z_{kk} \quad \forall k \in I \tag{2.16}$$

S obzirom da se USAHLP i CSAHLP razlikuju po dodatnom skupu uslova koje CSAHLP ima i koji dodatno otežavaju problem, jasno je da je i CSAHLP $\mathbb{NP}$-težak problem kao i USAHLP.

2.3.2 Pregled metoda za rešavanje problema

Istorijat razvoja matematičkog modela CSAHLP prati razvoj modela za USAHLP. U literaturi su poznate sledeće metode za rešavanje CSAHLP:

- Ernst i Krišnamurti su u [30] koristili dve metode: simulirano kaljenje (eng. simulated annealing - SA) i slučajni spust (eng. random descent heuristics - RDH), pomoću kojih su dobili gornje granice rešenja, a zatim ih koristili u BnB algoritmu za odsecanje, poredivši ih sa LP rešenjem.

- 2008. godine je Randall koristio 4 varijante metode mravlje kolonije (eng. ant colony optimization) [57].

- Iste godine, Kosta i drugi autori su rešavali CSAHLP pomoću modela sa dve funkcije cilja (eng. bi-objective) [21]. U tom pristupu, umesto ograničenja

kapaciteta, nova funkcija cilja služi da minimizuje protok kroz svaki hab.

- Stanimirović je 2007. godine pomoću genetskog algoritma dobila odlične rezultate [61].

- Kontreras i drugi su 2009. godine koristili Lagranževu relaksaciju (LR) i dekompoziciju na manje potprobleme i na taj način efikasno došli do kvalitetnih rešenja [16]. Međutim, ispostavilo se da je na jednom test primeru prikazano rešenje pogrešno. O tome će biti više reči u glavama 3 i 4.

Treba primetiti da nijedna poznata metoda nije uspešno primenjena na rešavanje oba problema, USAHLP i CSAHLP, premda se oni razlikuju samo po ograničenju kapaciteta.

U daljem tekstu, ukoliko se izloženo odnosi i na USAHLP i na CSAHLP biće korišćena oznaka SAHLP.

2.4 Razvoj potproblema

Kada se fiksira vrednost neke promenljive MILP problema M, verovatno će neki uslovi tog problema postati trivijalni pa ih je moguće ukloniti. Takođe, u zavisnosti od vrednosti fiksirane promenljive, moguće je odrediti vrednost nekih drugih, od nje zavisnih, promenljivih, te je i njih moguće eliminisati. Na primer, ako su x, y i z vektori nekih nenegativnih promenljivih, a a i b vektori nenegativnih koeficijenata i ako je vrednost neke promenljive z_m fiksirana i jednaka 0, tada:

- svi uslovi oblika $\sum a_j x_j + \sum b_k y_k \geq z_m$ postaju trivijalni i brišu se;
- za svaki uslov oblika $\sum a_j x_j + \sum b_k y_k \leq z_m$ vrednosti promenljivih x_j i y_k moraju biti $\forall j, x_j = 0$ i $\forall k, y_k = 0$, pa je te promenljive moguće zameniti konstantama i eliminisati iz modela.

Za dati MILP problem M i skup H, kojim su određene vrednosti nekih promenljivih problema M, definiše se potproblem $M(H)$ tako što se u problemu M fiksiraju vrednosti tih promenljivih a zatim uklone redundantni uslovi i promenljive.

Potproblem problema SAHLP sastoji se u tome da je potrebno odrediti alokaciju čvorova habovima kada je lokacija habova unapred data i fiksirana. Prelaskom na MILP problem 2.1.1 za SAHLP, gde je iz konteksta jasno da li se radi o USAHLP ili CSAHLP modelu, odnosno da li je uključen skup uslova (2.16) kojim se uvodi ograničenje kapaciteta, za dati skup $H \subseteq I$ neka $SAHLP(H)$ označava potproblem kada je skup habova fiksiran i jednak H. To znači da se neke promenljive Z mogu fiksirati: $Z_{ii} = 1, i \in H$ i $Z_{ii} = 0, i \in Z \setminus H$. Zbog toga, kao i u prethodnom razmatranju, neke promenljive i neki uslovi u modelu SAHLP postaju suvišni i mogu se ukloniti. U radu [65] je primenjen ovaj postupak, čime je dobijen sledeći potproblem $SAHLP(H)$:

$$\begin{aligned} \min \sum_{i \in I \setminus H} \sum_{k \in H} C_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{k \in H} C_{kk} (\chi O_k + \delta D_k) + \\ \sum_{i \in I \setminus H} \sum_{k \in H} \sum_{l \in H} \alpha C_{kl} Y_{kl}^i + \sum_{i \in I \setminus H} \sum_{k \in H} \sum_{l \in H} \alpha C_{lk} W_{li} Z_{ik} + \sum_{k \in H} \sum_{l \in H} \alpha C_{kl} W_{kl} + \sum_{k \in H} f_k \end{aligned} \tag{2.17}$$

tako da:

$$\sum_{k\in H} Z_{ik} = 1 \quad \forall i \in I\backslash H \tag{2.18}$$

$$\sum_{l\in H} Y_{kl}^{i} - \sum_{l\in H} Y_{lk}^{i} = (O_i - W_{ii})Z_{ik} - W_{ik} - \sum_{j\in I\backslash H, j\neq i} W_{ij}Z_{jk} \quad \forall i \in I\backslash H, k \in H \tag{2.19}$$

$$O_k + \sum_{i\in I\backslash H} O_i Z_{ik} \leq G_k \quad \forall k \in H, \text{ samo za CSAHLP(H)} \tag{2.20}$$

$$\sum_{l\in H, l\neq k} Y_{kl}^{i} \leq O_i Z_{ik} \quad \forall i \in I\backslash H, k \in H \tag{2.21}$$

$$Y_{kl}^{i} \geq 0 \quad \forall i \in I\backslash H, k, l \in H \tag{2.22}$$

$$Z_{ik} \in \{0, 1\} \quad \forall i \in I\backslash H, k \in H. \tag{2.23}$$

Funkcija cilja (2.17) deluje komplikovanija od funkcije cilja (2.3) ali treba primetiti da su drugi, peti i šesti sabirak u (2.17) zapravo konstante, kada je dato H. Prvi sabirak iz (2.3) je podeljen na dva: jedan za prikupljanje i distribuciju od ne-hab čvora i do hab čvora k i jedan za prikupljanje i distribuciju od haba k do samog sebe, što je konstanta jednaka 0 jer $C_{kk} = 0$, ali je, zbog potpunosti, i dalje prikazana. Drugi sabirak iz (2.3) je podeljen na tri nova sabirka: jedan za transfer saobraćaja koji polazi iz ne-hab čvora i preko habova k i l; drugi za transfer saobraćaja koji polazi iz haba l a za odredište ima ne-hab i koji je pridružen habu k; i treći, konstantan, koji predstavlja transfer saobraćaja koji polazi is haba k i za odredište ima hab l. Poslednji sabirak u (2.17) je konstantan jer predstavlja sumu fiksnih troškova za uspostavljanje zadatih habova H.

Svi uslovi problema $SAHLP$ prelaze u odgovarajuće uslove problema $SAHLP(H)$, s tim da je razlika što kod promenljivih Z_{ik}, prvi indeks i uzima vrednosti iz $I\backslash H$ a drugi indeks k uzima vrednosti iz H; za Y_{kl}^{i} promenljive, indeksi uzimaju vrednosti iz sledećih skupova: $i \in I\backslash H$ i $k, l \in H$.

Skup uslova (2.5) bi postao $Z_{ik} \leq 1$, što je trivijalno jer su Z_{ik} binarne promenljive, tako da je taj skup uslova izbačen iz $SAHLP(H)$.

Ukoliko je broj uspostavljenih habova mnogo manji od ukupnog broja čvorova

($h = |H| << |I| = n$), tada je broj promenljivih i uslova problema $SAHLP(H)$ značajno manji nego u problemu $SAHLP$. Umesto $n^3 + n^2$ promenljivih, sada ih ima $(n-h) * h^2 + (n-h) * h$. U najboljim poznatim rešenjima test primera važi da je $|H| << |I|$, a može se reći i da važi $|H| \approx ln|I|$. Na primer, za instance sa 200 čvorova, broj habova u najboljim poznatim rešenjima je 2, 3 ili 4.

Poglavlje 3

Memetski i hibridni algoritmi

U ovom poglavlju biće prikazane tri metaheurističke metode za rešavanje lokacijskih problema, kao i rezultati koji su njihovom primenom dobijeni. Efikasnost tih metoda će biti upoređena sa prethodno poznatim metodama iz literature koje se smatraju najkvalitetnijim.

Za potrebe rešavanja tri različita lokacijska problema razvijene su sledeće metode:

- Memetski algoritam za rešavanje USAHLP, koji se sastoji od genetskog algoritma sa dve lokalne pretrage;
- Memetski algoritam za rešavanje MLUFLP, koji se sastoji od genetskog algoritma i jedne lokalne pretrage;
- Hibridni algoritam za rešavanje CSAHLP, koji se sastoji od genetskog algoritma i metode grananja i ograničavanja.

Sledi detaljan prikaz sve tri predložene metode.

3.1 Memetski algoritam za rešavanje hab lokacijskog problema sa jednostrukom alokacijom bez ograničenja kapaciteta

U poglavlju 2.1 definisan je i predstavljen hab lokacijski problem sa jednostrukom alokacijom bez ograničenja kapaciteta (eng. Uncapacitated Single Allocation Hub Location Problem, skr. USAHLP). Matematički model, kao i sve oznake i pretpostavke iz tog poglavlja, koriste se i u tekstu koji sledi.

Memetski algoritam 3 za rešavanje USAHLP [46] zasniva se na genetskom algoritmu 1 prikazanom u poglavlju 1.3.1. Razlika je u tome što se u MA, nakon računanja funkcije cilja, pozivaju dve lokalne pretrage, *Lokacija* i *Alokacija*.

Algoritam 3: Memetski algoritam (MA) za rešavanje USAHLP

```
UlazniPodaci();
PseudoslučajnoGenerisanjePopulacije();
while not KrajRadaGA() do
    RačunanjeFunkcijeCilja();
    Lokalna pretraga Lokacija;
    Lokalna pretraga Alokacija;
    primena operatora:
        Prilagođenost;
        Selekcija;
        Ukrštanje;
        Mutacija;
    NovaGeneracija();
end
Rezultati();
```

Sledi opis zapisa svake jedinke (kodiranje), način generisanja početne populacije, opis svih genetskih operatora kao i objašnjenje algoritama lokalne pretrage.

3.1.1 Kodiranje

Zapis svake jedinke sastoji se od $n = |I|$ gena, gde je I mreža čvorova. Svi čvorovi i svaki gen neke jedinke su numerisani od 0 do $n-1$. Svaki gen neke jedinke se odnosi na odgovarajući čvor u mreži: čvoru sa indeksom i odgovara i-ti gen u genetskom kodu. Svi geni svake jedinke se sastoje od istog broja bitova, označenog sa *nbit* i sastoje se iz dva dela:

- Prvim delom svakog gena određuje se da li je u čvoru kome odgovara taj gen uspostavljen hab ili ne. Za to se koristi jedan bit. Ukoliko je vrednost tog bita 1, odgovarajući čvor je hab, u suprotnom nije hab;
- Drugi deo svakog gena se odnosi na alokaciju, odnosno određuje kom habu je alociran čvor kome taj gen odgovara. Mogućih alokacija jednog čvora može biti n, te se za kodiranje alokacije koristi binarno kodirani ceo broj između 0 i $n-1$. Za to je potrebno $\lceil log_2 n \rceil$ bitova.

Za binarno kodirani zapis svake jedinke potrebno je ukupno $n \cdot (1 + \lceil log_2 n \rceil)$ bitova.

Prilikom čitanja zapisa date jedinke primenjuju se sledeća pravila:

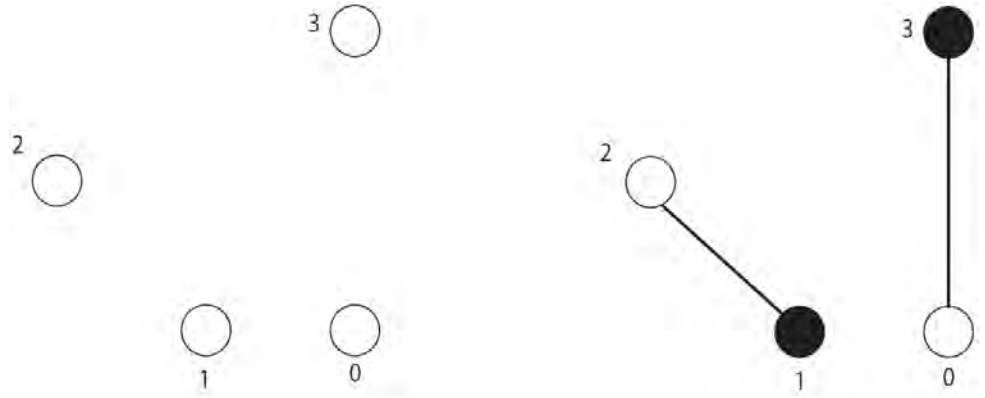
- Prvo se pročitaju svi habovi H i odredi se njihov broj $nhab = |H|$;
- Ukoliko je $nhab = 0$, jedinka se odbacuje jer ne može predstavljati dopustivo rešenje; kako nema drugih ograničenja, svaka jedinka sa $nhab > 0$ predstavlja neko dopustivo rešenje;
- Problem je definisan tako da su habovi uvek alocirani sami sebi, te zbog toga za one gene kod kojih je prvi bit jednak 1, dakle, koji su habovi, drugi deo gena se ignoriše;
- Intuitivno je jasno da će u najboljim rešenjima ne-hab čvorovi biti alocirani „bližim“ habovima, odnosno onim habovima do kojih su troškovi transporta manji. Zbog toga, kada se pročita broj j iz drugog dela gena, on ne označava indeks haba iz I ili H, već indeks iz niza nh koji se pravi od skupa H sortiranog po rastućem redosledu troškova transporta od čvora na koji se taj gen odnosi. S obzirom da j može biti najviše $n - 1$, a habova ima $nhab$, za određivanje haba kome je čvor alociran koristi se vrednost $k = j\%nhab$. Ako je $k = 0$, čvor se alocira najbližem habu; za $k = 1$ sledećem itd. Ovakva vrsta alociranja i kodiranja je predstavljena u [61] i naziva se „uređivanje po najbližem susedu“ (eng. "nearest neighbor ordering").

Zbog računanja $k = j\%nhab$ i zbog ignorisanja drugog dela gena kada prvi deo gena određuje hab, moguće je da dve jedinke sa različitim binarnim zapisom predstavljaju identično rešenje. To nije mana, već prednost, jer doprinosi bogatstvu genetskog materijala.

Kada je na osnovu binarnog koda neke jedinke određena lokacija i alokacija, time je direktno određeno pridruživanje $i \rightarrow hab(i)$, te se vrednost funkcije cilja rešenja koje ta jedinka predstavlja direktno računa pomoću formule (2.1). Složenost te formule je $O(n^2)$.

Primer 1.

Na slici 3.1 data je mreža sa 4 čvora. Pretpostavimo da su dati i ulazni parametri te mreže:



Slika 3.1: Primer USAHLP mreže sa 4 čvora i jedno moguće rešenje lokacije i alokacije

$$W = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 6 & 2 & 1 & 3 \\ 4 & 3 & 2 & 2 \\ 1 & 10 & 5 & 1 \end{pmatrix} \text{ i } C = \begin{pmatrix} 0 & 1 & 4 & 3 \\ 1 & 0 & 2 & 4 \\ 4 & 3 & 0 & 4 \\ 3 & 4 & 4 & 0 \end{pmatrix}.$$

Fiksni troškovi su $f = (10, 20, 30, 40)$ a težinski faktori za prikupljanje, transfer i distribuciju su $\chi = 1$, $\alpha = 0.5$ i $\delta = 1.5$. Svaki gen ima 1 bit za lokaciju i 2 za alokaciju, što znači da svaka jedinka ukupno ima 12 bita. Neka je kôd neke jedinke dat sa: |011|110|000|101|. Posmatrajući prve bitove svakog gena, vidi se da su habovi uspostavljeni u čvorovima 1 i 3, a da su ne-hab čvorovi 0 i 2. Za ne-hab čvorove niz habova se poređa u rastućem poretku troškova transporta. Ovde je taj niz isti za oba čvora: (1, 3). Za ne-hab čvor 0, drugi deo gena je 11, što znači da je $j = 3$, odnosno $k = 3\%2 = 1$. Dakle, čvor 0 nije pridružen sebi najbližem habu, već narednom, odnosno habu 3. Za ne-hab čvor 2, drugi deo gena je 00, te je $j = 0$ i $k = 0$, pa se on alocira sebi najbližem habu 1. Time su direktno određene vrednosti promenljivih Z_{ik}, odnosno pridruživanje $i \to hab(i)$.

Nakon toga, računa se vrednost funkcije cilja prema formuli (2.1). Fiksni troškovi su 60, ukupni troškovi transporta su 236.5 te je vrednost funkcije cilja ovog rešenja 296.5.

3.1.2 Generisanje početne populacije

Korišćena je populacija od 150 jedinki. Bitovi svake jedinke se postavljaju pomoću generatora pseudoslučajnih brojeva. Svi čvorovi, bez pristrasnosti, imaju jednaku verovatnoću da u njima budu uspostavljeni habovi. Međutim, kod drugog dela

svakog gena, alokacije, uvedena je pristrasnost na sledeći način: bit najmanje težine ima najveću verovatnoću da bude postavljen na 1 i ona iznosi $\frac{1}{2 \cdot n}$. Sledeći bit ima verovatnoću $\frac{1}{4 \cdot n}$ itd. Na taj način, u drugom delu svakog gena će biti jako mali broj jedinica, a i kad budu postavljeni bitovi na vrednost 1, to će češće biti slučaj na nižim bitovima. Drugim rečima, celi brojevi koji su na ovaj način generisani će biti jako mali, bliski 0, što znači da će ne-hab čvorovi najčešće biti alocirani sebi najbližem habu.

3.1.3 Genetski operatori

Prilikom implementacije memetskog algoritma 3 primenjeni su sledeći genetski operatori:

- prilagođenost neke jedinke definisana je tako da bude jednaka normalizovanoj vrednosti funkcije cilja: $p_i = \frac{c_i - min(c)}{max(c) - min(c)}$, gde su c su vrednosti funkcije cilja svake jedinke, p su vrednosti funkcije prilagođenosti i p_i pripada intervalu [0,1];
- korišćena je fino gradirana turnirska selekcija [32] sa veličinom turnira 5 u 60% slučajeva a veličinom turnira 6 u preostalih 40% slučajeva;
- dvopoziciono ukrštanje u kome učestvuje 85% jedinki koje su prošle selekciju;
- mutacija sa zaleđenim bitovima [62]; verovatnoća mutacije svakog bita je jednaka $p_{mut} = 0.4/l$, gde je l ukupan broj bitova u genetskom kodu. Ukoliko sve jedinke imaju identičnu vrednost bita na nekoj poziciji, verovatnoća mutacije na toj poziciji se množi faktorom 3.5;
- politika zamene generacija u kojoj 2/3 elitnih jedinki prelazi u novu generaciju, a samo jedna trećina biva zamenjena novim jedinkama. Pre toga, ukoliko veći broj jedinki ima identičan genetski kôd, ostavlja se samo jedna jedinka. Ukoliko veći broj jedinki ima različit kôd, ali istu vrednost funkcije cilja, ostavlja se maksimalno 40 takvih jedinki. To je takozvana stabilna elitistička strategija [50];

Memetski algoritam 3 prestaje sa radom kada ukupan broj generacija dostigne G_{max} ili kada nema promene najbolje jedinke u R_{max} generacija. Vrednosti ta dva parametra određene su preliminarnim testiranjem. Rezultati dobijeni sa različitim vrednostima tih parametara prikazani su u sekciji 3.1.8.

Tabela 3.1: Oznake koje se koriste u algoritmima lokalne pretrage *Lokacija* i *Alokacija*

Oznaka	Opis
s	Lokalna pretraga *Lokacija* se primenjuje na 1 od s jedinki, s je parametar.
q	Lokalna pretraga *Lokacija* ispituje q najbližih čvorova kada pokušava da zameni čvor i hab, q je parametar.
r	Lokalna pretraga *Alokacija* se primenjuje na 1 od r jedinki, r je parametar.
N, G	Jedinke. Ista oznaka se koristi i za genetski kôd jedinke kada se kôd modifikuje.
$MA\_cilj(N)$	Vrednost funkcije cilja za jedinku N.
$H(G)$	Skup habova određenih jedinkom G.
$NH(G)$	Skup ne-hab čvorova određenih jedinkom G.
$\{nh\}$	Pomoćni niz čvorova koji se formira za neki hab $h \in H(G)$ neke jedinke G. Taj niz sadrži sve ne-hab čvorove iz $NH(G)$, poređane u rastućem redosledu troškova transporta od tog haba h za koji je niz generisan.
$Suma(G, k)$	Za dato G, suma svih troškova transporta od ne-hab čvora $k \in NH(G)$ preko haba kojem je k alociran.

3.1.4 Pregled korišćenih oznaka

Pre predstavljanja algoritama lokalne pretrage, radi lakšeg praćenja algoritama, u tabeli 3.1 je dat pregled oznaka koje se u njima koriste.

3.1.5 Lokalna pretraga za promenu lokacije habova

Lokalna pretraga *Lokacija* služi da promeni lokaciju habova u nekoj jedinki i na taj način poboljša vrednost funkcije cilja. Ideja je da se u genetskom kodu jedinke izvrši zamena dva čvora: jedan čvor u kome nije uspostavljen hab postaje hab, a drugi čvor koji jeste hab, postaje ne-hab čvor. Pri tom, sve alokacije se preusmeravaju sa starog haba na novi hab. Dakle, algoritam lokalne pretrage *Lokacija* ne menja broj habova. Ukoliko se na taj način postigne bolje rešenje, nova jedinka se čuva u populaciji a stara briše. Čim se postigne poboljšanje, lokalna pretraga za tu jedinku se prekida.

Algoritam 4 ima dva ulazna parametra celobrojnog tipa, s i q.

Parametar s određuje za koje jedinke će biti izvršena lokalna pretraga. Da bi se izbegla preuranjena konvergencija, lokalna pretraga se ne vrši za sve jedinke, već za svaku s-tu jedinku u populaciji. Propratni efekat je da se na taj način izbegava i preveliko usporavanje celokupnog memetskog algoritma.

Kada se odabere jedinka G, na osnovu njenog zapisa se formiraju skupovi habova

Algoritam 4: Lokalna pretraga Lokacija

```
ulaz    : s, q celobrojni parametri
foreach s-tu jedinku G u populaciji do
    napravi skup NH(G) ← ne-hab čvorovi iz G;
    napravi skup H(G) ← hab čvorovi iz G;
    foreach hab k iz skupa habova H(G) do
        /*napravi niz {nh} za dato k                                  */
        {nh} ← sortiraj NH(G) (rastući redosled troškova transporta od k);
        for i=1:q do
            N ← G;
            /*U jedinki N zameni hab k i ne-hab čvor nh(i)           */
            N ← označi nh(i) kao hab;
            N ← označi k kao ne-hab;
            N ← alociraj novi hab nh(i) samom sebi;
            N ← alociraj stari hab k novom habu nh(i);
            N ← promeni alokaciju ne-hab čvorova alociranih starom habu k
             tako da budu alocirani novom habu nh(i);
            if MA_cilj(G) > MA_cilj(N) then
                G ← N;
                /*kraj lokalne pretrage za jedinku G              */
                izlaz iz (foreach hab) petlje;
            end
        end
    end
end
```

$H(G)$ i ne-hab čvorova $NH(G)$. Za svaki hab k iz $H(G)$ formira se pomoćni niz {nh} koji sadrži sve čvorove iz G koji nisu habovi, poređane po rastućem redosledu troškova transporta od haba k. Zatim se prolazi kroz prvih q elemenata tog sortiranog niza ne-hab čvorova {nh}, dakle, prvih q čvorova najbližih habu k. Konačno, pokušava se zamena haba k čvorom $nh(i)$, gde je i vrednost indeksa od 1 do q. Formira se nova jedinka N koja je kopija jedinke G i u njoj se čvor $nh(i)$ označi kao hab, zatim se hab k označi kao ne-hab i sva alokacija se preusmeri sa k na $nh(i)$.

Konačno, ukoliko nova jedinka N predstavlja bolje rešenje od jedinke G, vrši se zamena tih jedinki u populaciji i lokalna pretraga za tu jedinku prekida sa radom.

Složenost izračunavanja $MA_{cilj}(N)$ je $O(n^2)$ i ponavlja se q puta za svaki hab k, što je svakako veće od složenosti sortiranja niza $NH(G)$ koja je $O(|G| \cdot ln|G|)$, pogotovo zato što se očekuje da $|G| \ll n$. Takođe, realokacija je složenosti $O(n)$, te zbog toga, prilikom računanja složenosti ovog algoritma, sortiranje i realokacija se mogu ignorisati, pa je složenost algoritma lokalne pretrage *Lokacija* jednaka $O(h * q * n^2 * p/s)$, gde je p veličina populacije, a h najveći broj habova koje neka jedinka u populaciji sadrži. Prilikom određivanja parametara q i s treba imati u vidu da linearno utiču na složenost celog memetskog algoritma.

3.1.6 Lokalna pretraga za promenu alokacije čvorova habovima

Za razliku od prve lokalne pretrage, druga ima zadatak da promeni alokaciju čvorova habovima. Algoritam *Alokacija* (algoritam 5) ima jedan ulazni parametar r. Svaka r-ta jedinka u populaciji prolazi kroz pretragu.

Za izabranu jedinku, pokušava se realokacija svakog ne-hab čvora nekom drugom habu. Za izabranu jedinku G, formiraju se skupovi habova $H(G)$ i ne-habova $NH(G)$. Za svaki ne-hab čvor k menja se alokacija sa čvora $i = hab(k)$ na hab $j, j \neq i$. Zatim je potrebno uporediti da li je tako obavljena alokacija efikasnija od prethodne. U opštem slučaju, promena alokacije jednog čvora može uticati na svaki sabirak sume kojom se računaju ukupni troškovi, jer transferi više ne idu preko istog para habova. Zbog toga je neophodno računati celu funkciju cilja $MA_{cilj}(N)$ da bi se znalo da li je nova alokacija bolja ili ne.

Da bi se povećala efikasnost lokalne pretrage *Alokacija*, prvo se računa suma troškova saobraćaja koji polazi iz ne-hab čvora k, za jednu i za drugu alokaciju.

Algoritam 5: Lokalna pretraga Alokacija

```
input : r - celobrojni parametar
foreach r-tu jedinku G u populaciji do
    napravi skup NH(G) ← ne-hab čvorovi iz G;
    napravi skup H(G) ← hab čvorovi iz G;
    foreach k ∈ NH(G) do
        Suma(G, k) ← suma transportnih troškova čvora k preko haba j kome
         je alociran;
        foreach i ∈ H(G), i ≠ j do
            N ← G;
            N ← preusmeri alokaciju k na i;
            Suma(N, k) ← suma transportnih troškova čvora k preko haba i
             kome je sada alociran;
            if Suma(G, k) > Suma(N, k) then
                if MA_cilj(G) > MA_cilj(N) then
                    G ← N;
                    /*kraj lokalne pretrage za jedinku G          */
                    izlaz iz (foreach k) petlje;
                end
            end
        end
    end
end
```

To se obavlja u vremenu $O(n)$ i tek ako je ta suma bolja, računa se cela funkcija cilja. Moguće je da bi nova alokacija bila bolja čak i ako suma troškova odlaznog saobraćaja iz k nije bolja u novoj alokaciji, ali se zarad povećanja efikasnosti takvi slučajevi ne ispituju.

Složenost algoritma i dalje zavisi od računanja funkcije cilja, te je jednaka $O(p/r*(n-h)*h*n^2)$, p je veličina populacije. Međutim, imajući u vidu prethodno razmatranje, može se očekivati da gotovo uvek važi: $Suma(G,k) > Suma(N,k) \Rightarrow MA_{cilj}(G) > MA_{cilj}(N)$. S obzirom da do prekida lokalne pretrage za neku jedinku dolazi čim se nađe prva efikasnija alokacija, onda se zarad određivanja očekivane složenosti ovog algoritma, složenost računanja funkcije cilja može ignorisati, jer će izračunavanje funkcije cilja gotovo uvek biti obavljeno samo jednom i umesto nje koristiti složenost izračunavanja $Suma(N,k)$. Drugim rečima, očekivana složenost ovog algoritma je za faktor n manja od teoretske složenosti, odnosno jednaka je $O(p/r*(n-h)*h*n/r) \approx O(n^2*h*p/r)$, jer $h \ll n$. Svakako, parametar r linearno utiče na složenost ovog algoritma a time i celog memetskog algoritma.

3.1.7 Standardne test instance

Kako su napredovale metode za rešavanje SAHLP problema, javljala se potreba za sve većim i težim test primerima. Na žalost, velikih test primera iz stvarnog života nije bilo pa su autori generisali i teže i veće test primere, zasnovane na postojećim.

U tabeli 3.2 je dat pregled poznatih test primera iz literature, sa brojem čvorova i opisom.

Sve u svemu, za USAHLP postoji 28 test primera koji sadrže od 10 do 200 čvorova, i koji se testiraju sa faktorom fiksnih troškova λ sa vrednostima $1, 0.9$ i 0.8. Postoje još 4 test primera sa 300 i 400 čvorova, što sve zajedno čini 88 test primera za USAHLP.

3.1.8 Rezultati

Memetski algoritam 3 je implementiran u programskom jeziku C. Sva testiranja su obavljena na računaru sa procesorom Intel Core i7-860 2.8 GHz sa 8GB RAM memorije i pod Windows 7 Professional operativnim sistemom. Testiranja su obavljena na pomenutom standardnom skupu USAHLP instanci.

Prilikom testiranja MA, test svake instance ponavlja se 20 puta sa različitim

Tabela 3.2: USAHLP i CSAHLP test primeri

Skup instanci	Dimenzije problema	Tip fiksnih troškova	Opis instanci
CAB	$10 \leq n \leq 25$	konstantan (100, 150 200, 250)	Standardne ORLIB instance [8]; Zasnovane na protoku vazdušnog saobraćaja između 25 američkih gradova; Prvi put korišćene u [53]
AP	$10 \leq n \leq 200$	Teški "T" Laki "L"	Standardne ORLIB instances [8]; Zasnovane na podacima Australijske Pošte za 200 poštanskih okruga u Australiji; Prvi put korišćene u [29]
modifikovane AP	$10 \leq n \leq 200$	Teške "T" Lake "L" $\lambda = 0.8$ $\lambda = 0.9$	Zasnovane na standardnim AP instancama; Fiksni troškovi se množe faktorom λ; Teže su za rešavanje od standardnih AP; Prvi put korišćene u [59]
velike mod. AP	$n = 300, 400$	Teške "T" Lake "L"	Zasnovane na standardnim AP instancama sa 200 čvorova; Ne koriste faktor λ; Prvi put korišćene u [59]

vrednostima za inicijalizaciju generatora pseudoslučajnih brojeva. Na taj način se utvrđuje da li je algoritam stabilan bez obzira na jedinke generisane u početnoj populaciji. Vrednosti celobrojnih parametara s, q i r, koji se koriste u lokalnim pretragama, određene su preliminarnim testiranjem. Korišćene su sledeće vrednosti:

- $s = 4$ - lokalna pretraga Lokacija se primenjuje na svaku četvrtu jedinku u populaciji;
- $q = 8$ - u lokalnoj pretrazi Lokacija pokušava se zamena svakog haba sa jednim od osam njemu najbližih ne-hab čvorova;
- $r = 4$ - lokalna pretraga Alokacija se primenjuje na svaku četvrtu jedinku u populaciji;

Rezultati koje je MA postigao upoređeni su sa rezultatima objavljenim u [59] i [33].

Kolone u tabelama sa rezultatima označavaju:

- Test inst. - naziv test instance koji sadrži broj čvorova i tip fiksnih troškova (teški - T, laki - L);

- CPLEX 10.1 - Prvo je prikazano optimalno rešenje dobijeno pomoću CPLEX 10.1 rešavača. Pri tom, oznaka „$*$“ znači da optimalnost nije dokazana, dok „$-$“ znači da CPLEX nije pronašao nijedno rešenje. Zatim je navedeno vreme izvršavanja u sekundama, gde „m.l.“ označava da je CPLEX prekinuo rad zbog dostizanja memorijskog limita a „v.l.“ zbog dostizanja vremenskog limita (24 časa);

- MA - za MA metodu je prvo prikazano najbolje pronađeno rešenje, a zatim:
  t(s) - vreme potrebno da MA pronađe najbolje rešenje;
  $t_{kraj}(s)$ - ukupno vreme izvršavanja MA;
  agap - prosečno odstupanje MA rešenja od najboljeg (u procentima);
  Gen. - prosečan broj generacija MA;

- $HubTS$ - najbolje rešenje i vreme izvršavanja HubTS tabu pretrage iz [59];

- $MSTS$ - najbolje rešenje i vreme izvršavanja MSTS tabu pretrage iz [59];

- GA - najbolje rešenje i vreme izvršavanja genetskog algoritma GA iz [33];

U kolonama sa rešenjima koriste se sledeće oznake:

- *opt* - rešenje je identično optimalnom rešenju koje je CPLEX pronašao;

- *cpl* - rešenje je identično sub-optimalnom rešenju koje je CPLEX pronašao;

- opt/naj - rešenje je ili optimalno ili najbolje poznato;

Na kraju svake tabele su date prosečne vrednosti po kolonama.

MA je takođe testiran na standardnim CAB instancama maksimalne dimenzije do 25 i dostigao je optimalna rešenja u vremenu koje se meri hiljaditim delovima sekunde. Slične rezultate na ovom skupu testova postižu i druge poznate metode. Zbog toga, CAB test više nije merodavan za poređenje različitih metoda, pa ti rezultati ovde nisu prikazani.

Genetski algoritam GA iz [33] i tabu pretrage HubTS i MSTS iz [59] su testirani na računarima slabijih performansi od računara na kome je testiran MA. Poređenjem performansi tih računara, korišćenjem podataka sa internet stranice *www.spec.org*, zaključuje se da je računar na kome je testiran MA približno 1.5 puta brži. Da bi poređenje bilo korektnije, vreme koje je potrebno za izvršavanje MA je pomnoženo koeficijentom 1.5.

Tabela 3.3: Rezultati MA i poređenje na AP instancama do 90 čvorova

Test	CPLEX 10.1		MA					GA		MSTS		HubTS	
inst.	Rešenje	t(s)	Reš.	t(s)	$t_{kraj}(s)$	agap	Gen.	Reš.	t(s)	Reš.	t(s)	Reš.	t(s)
10L	224250.05	0.078	opt	0.001	0.011	0.000	24.4	opt	0.101	opt	0.016	opt	0.015
10T	263399.94	0.125	opt	0.001	0.011	0.000	23.8	opt	0.113	opt	0.015	opt	0.015
20L	234690.62	0.220	opt	0.010	0.030	0.000	25.2	opt	0.206	opt	0.062	opt	0.046
20T	271128.18	0.250	opt	0.009	0.030	0.000	25.4	opt	0.213	opt	0.062	opt	0.062
25L	236650.62	0.563	opt	0.013	0.040	0.000	27.0	opt	0.275	opt	0.125	opt	0.125
25T	295667.84	12.364	opt	0.008	0.031	0.000	21.0	opt	0.233	opt	0.094	opt	0.031
40L	240986.23	1.002	opt	0.034	0.082	0.011	29.9	opt	0.554	opt	0.375	opt	0.781
40T	293164.84	15.512	opt	0.010	0.060	0.000	21.0	opt	0.500	opt	0.344	opt	0.188
50L	237421.98	22.771	opt	0.053	0.122	0.002	30.9	opt	0.904	opt	0.703	opt	1.891
50T	300420.98	275.976	opt	0.034	0.106	0.022	24.8	opt	0.592	opt	0.719	opt	0.469
prosek	-	32.886	opt	0.02	0.05	0.004	2.34	opt	0.369	opt	0.251	opt	0.362
60L	228007.900	11.18	opt	0.093	0.191	0.014	34.3	opt	1.205	-	-	-	-
60T	246285.034	9.53	opt	0.063	0.169	0.000	27.6	opt	1.016	-	-	-	-
70L	233154.289	90.30	opt	0.095	0.235	0.000	28.6	opt	1.489	-	-	-	-
70T	252882.633	22.42	opt	0.120	0.279	0.000	31.5	opt	1.397	-	-	-	-
80L	229240.373	143.39	opt	0.193	0.363	0.020	36.0	opt	2.397	-	-	-	-
80T	274921.572	140.58	opt	0.160	0.344	0.000	32.9	opt	1.818	-	-	-	-
90L	231236.235	423.14	opt	0.243	0.473	0.009	35.0	opt	2.463	-	-	-	-
90T	280755.459	153.72	opt	0.223	0.458	0.375	34.1	opt	1.934	-	-	-	-
prosek	-	-	opt	0.076	0.169	0.025	28.5	opt	1.715	-	-	-	-

Na instancama do dimenzije 200, MA je testiran sa parametrima zaustavljanja $G_{max} = 50$ i $R_{max} = 20$ (poglavlje 3.1.3), dok su za testiranje instanci dimenzija 300 i 400 korišćene vrednosti $G_{max} = 150$ i $R_{max} = 50$, jer je prostor rešenja znatno veći.

Rezultati testova na AP instancama do 90 čvorova su dati u Tabeli 3.3. Slično kao i na CAB skupu instanci, sve metode postižu optimalna rešenja u jako kratkom vremenskom periodu, te ih je teško porediti.

Na instancama sa 100 i više čvorova poređenje MA sa GA, MSTS i HubTS je moguće. Imajući u vidu razlike u performansama računara, zaključuje se da je MA oko 4 puta brži od GA, a da istovremeno ima manje prosečno odstupanje nego GA. Treba imati u vidu da je GA testiran samo na instancama do 200 čvorova. MA je brži i od MSTS i od HubTS tabu pretraga između 15 i 100 puta. Pri tom, MA je pronašao ukupno 8 boljih rešenja od onih koje su pronašle tabu pretrage. Razlika u brzini i kvalitetu rešenja u korist MA raste sa porastom dimenzije problema.

Tabela 3.4: Rezultati MA i poređenje na AP instancama od 100 do 200 čvorova

Test	CPLEX 10.1		MA					GA		MSTS		HubTS	
inst.	Rešenje	t(s)	Rešenje	t(s)	$t_{kraj}(s)$	agap	Gen.	Rešenje	t(s)	Reš.	t(s)	Reš.	t(s)
100L	238016.28	275.976	opt	0.124	0.394	0.000	24.4	opt	3.221	opt	7.95	opt	28.5
100T	305097.96	23.139	opt	0.193	0.478	0.000	28.9	opt	2.180	opt	5.19	opt	7.1
110L	222704.770	661.33	opt	0.369	0.761	0.000	34.5	opt	5.205	-	-	-	-
110T	227934.627	383.96	opt	0.407	0.813	0.000	35.0	opt	4.761	-	-	-	-
120L	-	m.l.	225801.362	0.319	0.728	0.002	30.5	225801.362	4.775	-	-	-	-
120T	-	m.l.	232460.818	0.455	0.919	0.657	35.0	232460.818	5.913	-	-	-	-
130L	-	m.l.	227884.626	0.479	1.025	0.000	33.7	227884.626	6.661	-	-	-	-
130T	-	m.l.	234935.968	0.625	1.152	0.022	37.5	234935.968	6.181	-	-	-	-
200L	233803.02*	v.l.	233802.976	1.563	2.889	0.015	37.1	233802.976	19.63	cpl	678.5	cpl	310.9
200T	272237.78*	v.l.	272188.113	1.737	3.232	0.208	38.0	272188.113	19.22	cpl	693.9	cpl	536.7
prosek	-	-	opt/naj	0.627	1.239	0.090	33.5	opt/naj	7.775	-	-	-	-

Tabela 3.5: Rezultati MA i poređenje na modifikovanim AP instancama ($n < 100, \lambda = 0.8$)

Test	CPLEX 10.1		MA					MSTS		HubTS	
inst.	Rešenje	t(s)	Reš.	t(s)	$t_{kraj}(s)$	agap	Gen.	Reš.	t(s)	Reš.	t(s)
10L	206555.04	0.110	opt	0.002	0.015	0.000	24.4	opt	0.015	opt	0.000
10T	242936.29	0.078	opt	0.001	0.012	0.000	23.8	opt	0.016	opt	0.015
20L	222085.76	0.313	opt	0.010	0.031	0.000	24.6	opt	0.078	opt	0.140
20T	255571.86	0.265	opt	0.010	0.031	0.000	25.9	opt	0.062	opt	0.062
25L	224428.91	1.000	opt	0.016	0.045	0.000	27.6	opt	0.141	opt	0.125
25T	279548.44	0.765	opt	0.011	0.039	0.000	24.4	opt	0.110	opt	0.156
40L	226032.08	6.703	opt	0.029	0.081	0.000	27.5	opt	0.486	opt	2.016
40T	285599.31	7.000	opt	0.010	0.060	0.000	21.0	opt	0.343	opt	0.187
50L	225895.98	27.141	opt	0.051	0.119	0.023	29.1	opt	1.081	opt	1.843
50T	281803.93	7.234	opt	0.070	0.151	0.003	34.6	opt	0.750	opt	2.281
prosek	-	5.061	opt	0.021	0.058	0.002	26.3	opt	0.308	opt	0.682
60L	218573.387	24.81	opt	0.088	0.190	0.028	32.5	-	-	-	-
60T	235211.393	8.8	opt	0.065	0.168	0.000	28.2	-	-	-	-
70L	221899.153	162.76	opt	0.118	0.268	0.001	31.1	-	-	-	-
70T	240737.734	23.17	opt	0.122	0.277	0.000	30.9	-	-	-	-
80L	217929.646	338.54	opt	0.223	0.383	0.016	39.4	-	-	-	-
80T	257751.027	138.65	opt	0.172	0.362	0.131	33.6	-	-	-	-
90L	220106.769	823.82	opt	0.334	0.538	0.006	41.2	-	-	-	-
90T	268373.837	343.69	opt	0.247	0.493	0.179	35.5	-	-	-	-
prosek	-	233.03	opt	0.171	0.335	0.004	34.1	-	-	-	-

Tabela 3.6: Rezultati MA i poređenje na modifikovanim AP instancama ($n < 100, \lambda = 0.9$)

Test	CPLEX 10.1		MA					MSTS		HubTS	
inst.	Rešenje	t(s)	Reš.	t(s)	$t_{kraj}(s)$	agap	Gen.	Reš.	t(s)	Reš.	t(s)
10L	215425.88	0.062	opt	0.001	0.011	0.000	23.7	opt	0.015	opt	0.015
10T	253798.41	0.125	opt	0.003	0.012	0.000	24.4	opt	0.031	opt	0.000
20L	228785.69	0.390	opt	0.010	0.031	0.000	25.4	opt	0.078	opt	0.046
20T	263350.03	0.297	opt	0.010	0.031	0.000	24.7	opt	0.094	opt	0.047
25L	230539.77	0.922	opt	0.013	0.044	0.000	27.9	opt	0.140	opt	0.109
25T	288066.69	0.812	opt	0.011	0.041	0.000	24.1	opt	0.187	opt	0.140
40L	234013.75	10.469	opt	0.031	0.082	0.000	28.1	opt	0.484	opt	2.047
40T	289382.06	1.422	opt	0.010	0.060	0.000	21.0	opt	0.562	opt	0.188
50L	231658.98	17.954	opt	0.053	0.125	0.072	30.4	opt	1.078	opt	1.860
50T	291435.494	8.63	opt	0.064	0.142	0.005	31.8	opt	1.141	opt	2.266
prosek	-	4.962	opt	0.021	0.058	0.008	26.1	opt	0.381	opt	0.672
60L	223712.162	19.73	opt	0.089	0.183	0.019	33.0	-	-	-	-
60T	240748.213	10.55	opt	0.058	0.158	0.000	26.8	-	-	-	-
70L	228337.777	186.28	opt	0.128	0.270	0.000	32.5	-	-	-	-
70T	246810.184	24.41	opt	0.114	0.270	0.000	30.3	-	-	-	-
80L	224334.542	488.48	opt	0.204	0.371	0.051	37.4	-	-	-	-
80T	266036.299	128.14	opt	0.185	0.364	0.000	34.4	-	-	-	-
90L	226769.050	1138.99	opt	0.265	0.494	0.004	37.8	-	-	-	-
90T	275248.468	201.69	opt	0.251	0.487	0.480	35.6	-	-	-	-
prosek	-	274.78	opt	0.162	0.325	0.069	33.5	-	-	-	-

Tabela 3.7: Rezultati MA i poređenje na modifikovanim AP instancama ($100 \leq n \leq 200, \lambda = 0.8$)

Test	CPLEX 10.1		MA					HubTS		MSTS	
inst.	Rešenje	t(s)	Rešenje	t(s)	$t_{kraj}(s)$	agap	Gen.	Reš.	t(s)	gap	t(s)
100L	227409.66	890.090	opt	0.136	0.404	0.000	24.9	opt	27.83	1.39 %	13.31
100T	297646.26	88.345	opt	0.310	0.593	0.000	36.6	opt	36.03	0.23 %	5.02
110L	214536.059	677.45	opt	0.372	0.768	0.000	34.1	-	-	-	-
110T	218719.945	554.79	opt	0.440	0.845	0.000	36.3	-	-	-	-
120L	-	m.l	215865.553	0.420	0.862	0.148	34.4	-	-	-	-
120T	-	m.l.	222680.294	0.552	1.010	0.451	37.7	-	-	-	-
130L	-	m.l.	218676.703	0.554	1.115	0.015	35.6	-	-	-	-
130T	-	m.l.	224317.777	0.618	1.187	0.000	36.9	-	-	-	-
200L	223704.44*	v.l.	223704.422	1.810	3.095	0.015	39.1	cpl	463.05	0%	679.71
200T	260312.67*	v.l.	258823.126	1.994	3.498	0.874	40.6	cpl	623.30	1.62%	39.63
prosek	-	-	opt/naj	0.721	1.338	0.150	35.6	-	-	-	-

Tabela 3.8: Rezultati MA i poređenje na modifikovanim AP instancama ($100 \leq n \leq 200, \lambda = 0.9$)

Test	CPLEX 10.1		MA					HubTS		MSTS	
inst.	Rešenje	t(s)	Rešenje	t(s)	$t_{kraj}(s)$	agap	Gen.	Reš.	t(s)	gap	t(s)
100L	232712.97	29,73	opt	0.127	0.393	0.000	24.5	opt	28.09	opt	43.86
100T	301719.69	249.777	opt	0.260	0.534	0.000	31.6	opt	7.14	opt	6.94
110L	218620.42	461.70	opt	0.346	0.736	0.001	33.0	-	-	-	-
110T	223327.29	362.79	opt	0.451	0.860	0.000	37.1	-	-	-	-
120L	-	m.l	221693.722	0.404	0.813	0.002	32.9	-	-	-	-
120T	-	m.l.	227687.232	0.458	0.925	0.878	34.7	-	-	-	-
130L	-	m.l.	223280.664	0.516	1.068	0.000	34.8	-	-	-	-
130T	-	m.l.	229626.873	0.697	1.225	0.028	38.9	-	-	-	-
200L	228753.70*	v.l.	228753.699	1.449	2.792	0.017	35.6	cpl	467.8	0%	690.73
200T	266275.22*	v.l.	266134.108	1.590	3.111	0.303	36.6	cpl	625.1	0.23%	50.11
prosek	-	-	opt/naj	0.630	1.246	0.123	34.0	-	-	-	-

Tabela 3.9: Rezultati MA i poređenje na velikim instancama ($n = 300, 400$)

Test	MA					HubTS		MSTS	
inst.	Rešenje	t(s)	$t_{kraj}(s)$	agap	Gen.	Rešenje	t(s)	gap	t(s)
300L	263964.001	15.492	29.887	0.464	94.7	264837.88	4605.078	2.79%	6221.969
300T	276023.348	20.961	31.825	2.539	112.1	276047.75	3483.375	5.67%	6266 .391
400L	267921.707	87.512	131.245	0.796	127.7	268164.13	8447.437	3.84%	15116.031
400T	284037.245	56.574	94.525	1.569	111.6	284212.47	9530.828	3.08%	15225.438
prosek	-	45.135	71.870	1.342	111.5	-	6516.6795	3.84%	12187.813

3.2 Memetski algoritam za rešavanje lokacijskog problema snabdevača neograničenog kapaciteta u više nivoa

U poglavlju 2.2 definisan je i predstavljen lokacijski problem snabdevača neograničenog kapaciteta u više nivoa (eng. Multi Level Uncapacitated Facility Location Problem, skr. MLUFLP). Matematički model, kao i sve oznake i pretpostavke iz tog poglavlja koriste se i u tekstu koji sledi.

Ono što razlikuje MLUFLP od sličnih problema jeste činjenica da potproblem problema MLUFLP, takozvani FixMLUFLP, koji se odnosi na alokaciju klijenata habovima za zadati skup habova, nije NP-težak. To potkrepljuje sledeća teorema.

Teorema 1. *FixMLUFLP se može polinomski svesti na problem najkraćih puteva u direktnom acikličnom grafu (DAG).*

Formulacija Teoreme 1 je preuzeta iz [43] gde se može pronaći i dokaz teoreme, kao i algoritam dinamičkog programiranja pomoću koga se FixMLUFLP optimalno rešava uz vremensku složenost $O(n^2)$.

Memetski algoritam 6 za rešavanje MLUFLP zasniva se na genetskom algoritmu 1. Za razliku od MA za USAHLP (Algoritam 3), primenjena je nešto drugačija strategija. Nakon računanja funkcije cilja, poziva se funkcija lokalne pretrage za određene jedinke, ali samo u slučaju da u prethodnih N generacija nije bilo poboljšanja najboljeg rešenja. Lokalna pretraga za alokaciju nije potrebna jer se FixMLUFLP rešava na prethodno opisan način. Zbog toga, u ovom memetskom algoritmu postoji jedna lokalna pretraga koja se odnosi na lokaciju habova.

Sledi opis zapisa svake jedinke u populaciji (kodiranje), način generisanja početne populacije, opis genetskih operatora kao i opis algoritma lokalne pretrage.

3.2.1 Kodiranje

S obzirom da se alokacija rešava pomoću FixMLUFLP, zapisom jedinke se određuju habovi, ali ne i pridruživanje klijenata habovima. Zbog toga se zapis svake jedinke sastoji od $m = |F|$ bitova, gde svaki bit označava da li je na odgovarajućoj lokaciji uspostavljen hab ili ne. Da bi zapis jedinke predstavljao dopustivo rešenje, neophodno je da na svakom nivou bude uspostavljen bar jedan hab. Jedinka bez haba na nekom

Algoritam 6: Memetski algoritam (MA) za rešavanje MLUFLP

```
ulaz    : celobrojni parametar N
UlazniPodaci();
PseudoslučajnoGenerisanjePopulacije();
while not KrajRadaGA() do
    RačunanjeFunkcijeCilja();
    if BrojGeneracijaBezPoboljšanja ≥ N then
        Lokalna pretraga;
    end
    primena operatora:
        Prilagođenost;
        Selekcija;
        Ukrštanje;
        Mutacija;
    NovaGeneracija();
end
Rezultati();
```

nivou može nastati prilikom generisanja početne populacije ili nakon primene genetskih operatora ukrštanja ili mutacije. Zbog toga se prilikom zapisivanja jedinke vodi računa o tome i ukoliko ne postoji hab na nekom nivou, jedinka se dopunjuje tako što se na pseudoslučajan način doda po jedan hab na svakom nivou na kome nedostaje. Na taj način je eliminisana mogućnost da u populaciji postoje jedinke koje ne predstavljaju dopustivo rešenje.

Nakon što se pročita zapis neke jedinke i time odrede habovi, na prethodno opisan način se dinamičkim programiranjem određuje alokacija klijenata habovima, u vremenu izvršavanja $O(n^2)$, gde je n broj klijenata. Tom prilikom se računaju i troškovi tako uspostavljene alokacije. Da bi se izračunala funkcija cilja potrebno je još dodati fiksne troškove za uspostavljanje habova, za šta je potrebno još $O(m)$ sabiranja. Ukupna složenost računanja vrednosti funkcije cilja je $O(n^2 + m)$.

3.2.2 Generisanje početne populacije

Populacija od 150 jedinki se generiše na pseudoslučajan način. Jedinke koje ne sadrže habove na svim nivoima se dopunjuju potrebnim habovima, takođe pseudoslučajno. Lokalna pretraga se ne primenjuje na početnu populaciju.

3.2.3 Genetski operatori

Korišćeni su sledeći genetski operatori:

- Prilagođenost jedinke jednaka je normalizovanoj vrednosti funkcije cilja;
- Korišćena je fino gradirana turnirska selekcija [32]: održava se 50 turnira različite veličine - 30 turnira sa 6 jedinki i 20 turnira sa 5 jedinki;
- Dvopoziciono ukrštanje u kojem učestvuje 80% jedinki koje su prošle selekciju;
- Osnovna verovatnoća mutacije svakog bita je jednaka $p_{mut} = 0.5/m$. Ukoliko sve jedinke imaju identičnu vrednost bita na nekoj poziciji, primenjuje se mutacija sa zaleđenim bitovima [62] i 4 puta većom verovatnoćom mutacije: $p_{zal} = 2/m$;
- Primenjena je stabilna elitistička strategija [50] prilikom zamene generacija: jedna trećina jedinki sa najlošijom vrednošću funkcije cilja biva zamenjena novim jedinkama, a preostale dve trećine jedinki prelaze u novu generaciju. Ukoliko ima jedinki sa identičnim genetskim kodom, ostavlja se jedna a ostale se brišu.

Memetski algoritam 3 prestaje sa radom kada ukupan broj generacija dostigne 5000 ili kada nema promene najbolje jedinke u 2000 generacija ili kada vreme izvršavanja algoritma pređe 60 minuta. Te vrednosti su određene preliminarnim testiranjem.

3.2.4 Lokalna pretraga

Ideja lokalne pretrage za MLUFLP jeste da se pojedinim jedinkama iterativno dodaje ili oduzima po jedan hab i da se svaka promena, koja dovodi do poboljšanja vrednosti funkcije cilja, pamti. Iteracije se nastavljaju sve dok ima poboljšanja. Ako oduzimanje ili dodavanje jednog haba ne daje bolje rešenje, proces se prekida. Algoritmom 7 se realizuje ova ideja.

Lokalna pretraga se primenjuje na svaku R-tu jedinku u populaciji, gde je R parametar. U svakoj iteraciji za neku jedinku G vrši se negacija jednog po jednog bita, redom počev od prvog. Ukoliko negacija jednog bita dovede do poboljšanja vrednosti funkcije cilja, ona se pamti i proces se ponavlja, ponovo od prvog bita, ali

Algoritam 7: Lokalna pretraga za MLUFLP

```
ulaz    : celobrojni parametar R
foreach R-tu jedinku G u populaciji do
    popravka ← true;
    while popravka do
        popravka ← false;
        N ← G;
        /*m = |F|, što je ujedno i broj bitova u kodu jedinke    */
        for k = 1..m do
            N ← negacija bita k u kodu N;
            if N je dopustivo rešenje then
                if MA_cilj(G) > MA_cilj(N) then
                    G ← N;
                    popravka ← true;
                    break;
                end
            end
            N ← negacija bita k u kodu N;
        end
    end
end
```

sa tom novodobijenom jedinkom. Ukoliko ne dođe do poboljšanja, bit se vraća na prethodnu vrednost i nastavlja se dalje sa pokušajem negacije narednog bita.

Nakon svake negacije, potrebno je izračunati vrednost funkcije cilja nove jedinke, složenosti $O(n^2 + m)$. Kada ne dođe do popravke jedinke, to će se ponoviti m puta, te je složenost $O(m(n^2 + m))$. Kada dođe do popravke na bitu k, ukupno je k puta računata vrednost funkcije cilja, $k \leq m$. Zbog toga, ukupna složenost lokalne pretrage za jednu jedinku je $O((p + 1)m(n^2 + m))$, gde je p broj izvršenih popravki.

3.2.5 Rezultati

Memetski algoritam je implementiran u programskom jeziku C# korišćenjem Microsoft .NET platforme. Prilikom implementacije, primenjene su tehnike paralelizacije na sve delove algoritma u kojima se jedinke obrađuju nezavisno jedna od druge, a to su redom: računanje vrednosti funkcije cilja, lokalna pretraga, prilagođenost, ukrštanje i mutacija.

Svi testovi su obavljeni na računaru sa Intel Core i7-860 2.8 GHz quad-core procesorom (sa 4 jezgra) i 8GB RAM memorije, pod Windows 7 Professional opera-

Tabela 3.10: Skupovi test primera za MLUFLP

Naziv	Broj nivoa	Dimenzije	Opis
ORLIB instance	$k = 1$	$50 \leq n \leq 1000$ $16 \leq m \leq 100$	Standardne instance [8] malih i srednjih dimenzija.
M* instance	$k = 1$	$300 \leq n \leq 2000$ $300 \leq m \leq 2000$	Instance velikih dimenzija; Prvi put korišćene u [56].
Modifikovane ORLIB instance	$2 \leq k \leq 4$	$50 \leq n \leq 1000$ $16 \leq m \leq 100$	Generisane iz ORLIB instanci; Prvi put korišćene u [44].
Velike MLUFLP instance	$2 \leq k \leq 5$	$300 \leq n \leq 2000$ $300 \leq m \leq 2000$	Generisane iz M* instanci [56]; Prvi put korišćene u [44].

tivnim sistemom. Svaka instanca je testirana po 20 puta, sa različitim vrednostima za inicijalizaciju generatora pseudoslučajnih brojeva. Na taj način se utvrđuje da li je algoritam stabilan, bez obzira na jedinke generisane u početnoj populaciji.

Opis instanci koje su testirana, njihovo poreklo i dimenzije dati su u tabeli 3.2.5.

Naziv svake instance ujedno sadrži i podatke o dimenzijama problema na koji se odnosi. Na primer, naziv instance *capb*_3*L*_12_25_63.1000 označava da instanca potiče od *capb* instance (ORLIB instanca), da sadrži 3 nivoa sa, redom, 12, 25 i 63 potencijalna snabdevača i 1000 klijenata.

Parametar N određuje posle koliko generacija bez poboljšanja najboljeg rešenja se poziva lokalna pretraga dok parametar R određuje za koje jedinke će biti izvršena lokalna pretraga (1 od R jedinki u populaciji). Prilikom svih testova korišćene su vrednosti $N = 150$ i $R = 15$.

Izvršeno je i poređenje sa prethodno najboljom poznatom metodom za rešavanje MLUFLP, a to je genetski algoritam GA iz [44]. Implementacija tog algoritma je testirana na računaru sa procesorom Intel 1.8 GHz sa 512 MB memorije i to na istom skupu testova.

Rezultati testova i poređenje MA i GA [44] dati su u tabelama 3.11 i 3.12. U tabeli 3.11 pored imena instance prikazano je i optimalno rešenje dobijeno pomoću CPLEX 12.1 rešavača. Obe metode, MA i GA, su pronašle rešenja identična optimalnom za svih 27 test instanci prikazanih u tabeli 3.11. U tabeli 3.12 su prikazana rešenja za test instance za koje optimalno rešenje nije poznato, već je u kolonama označenim sa „Rešenje“, za svaku od metoda GA i MA prikazana vrednost funkcije cilja najboljeg pronađenog rešenja. U obe tabele prikazane su i sledeće kolone za svaku od metoda:

Tabela 3.11: Rezultati MA i poređenje sa GA - optimalna rešenja

Test instanca	CPLEX 12.1	GA			MA		
		$t(s)$	*agap*	σ	$t(s)$	*agap*	σ
cap71_1L_16.50	932615.75	0.012	0.000	0.000	0	0.000	0.000
cap71_2L_6_10.50	1813375.51	0.006	0.000	0.000	0	0.000	0.000
cap71_3L_2_5_9.50	4703216.31	0.005	0.000	0.000	0	0.000	0.000
cap101_1L_25.50	796648.44	0.030	0.000	0.000	0.01	0.000	0.000
cap101_2L_8_17.50	1581551.40	0.018	0.000	0.000	0.01	0.000	0.000
cap101_3L_3_7_15.50	3227179.81	0.019	0.000	0.000	0.01	0.000	0.000
cap131_1L_50.50	793439.56	0.199	0.000	0.000	0.03	0.000	0.000
cap131_2L_13_37.50	1592548.45	0.118	0.000	0.000	0.03	0.000	0.000
cap131_3L_6_14_30.50	3201970.46	0.105	0.125	0.084	0.03	0.071	0.087
cap131_4L_3_7_15_25.50	3630297.67	0.071	0.000	0.000	0.03	0.000	0.000
capa_1L_100.1000	17156454.48	13.409	0.000	0.000	0.97	0.000	0.000
capa_2L_30_70.1000	31524957.41	8.380	0.106	0.063	1.07	0.021	0.050
capa_3L_15_30_55.1000	40725103.25	3.076	0.000	0.000	0.92	0.000	0.000
capa_4L_6_12_24_58.1000	54643362.80	4.688	0.881	0.904	1	0.617	0.840
capb_1L_100.1000	12979071.58	43.642	0.060	0.186	10.14	0.170	0.286
capb_2L_35_65.1000	25224163.28	18.414	0.860	1.406	5.26	0.631	1.297
capb_3L_12_25_63.1000	34978486.51	3.137	0.000	0.000	0.57	0.000	0.000
capb_4L_6_13_31_50.1000	53034149.83	4.652	0.110	0.057	1.1	0.117	0.049
capc_1L_100.1000	11505594.33	34.542	0.073	0.135	15.08	0.026	0.013
capc_2L_32_68.1000	22762468.84	22.625	0.770	0.717	2.56	0.290	0.691
capc_3L_13_27_60.1000	35540649.43	8.539	0.675	1.176	0.74	0.182	0.793
capc_4L_4_9_27_60.1000	57017358.04	4.601	0.150	0.210	1.55	0.279	0.205
mq1_1L_300.300	3591.27	14.288	0.000	0.000	2.22	0.000	0.000
mq1_2L_100_200.300	8341.29	19.313	0.000	0.000	0.61	0.000	0.000
mq1_3L_30_80_190.300	12994.87	16.980	2.273	1.369	0.87	0.389	0.953
mq1_4L_20_40_80_160.300	17648.01	17.423	1.764	2.124	1.92	0.871	1.523
mq1_4L_18_39_81_162.300	18048.03	14.876	0.736	0.876	3.68	0.440	0.553
prosek	-	9.377	0.318	0.345	1.867	0.152	0.272

- $t(s)$ - prosečno vreme izvršavanja algoritma, u sekundama;
- *agap* - prosečno odstupanje od *naj* rešenja, u procentima;
- σ - standardna devijacija od najboljeg rešenja, u procentima.

MA je pronašao bolje rešenje od GA na 6 od 14 instanci za koje optimalno rešenje nije poznato, dok su na 8 instanci pronašli ista rešenja. Ukupna prosečna vrednost *agap* za GA je 0.6 dok je za MA ta vrednost 0.3. Prosečna vrednost σ za GA je 0.48 dok je za MA 0.33. Zaključuje se da je MA stabilniji algoritam po kvalitetu rešenja od GA. Konačno, vreme izvršavanja MA je, u proseku, preko 4 puta manje

Tabela 3.12: Rezultati MA i poređenje sa GA - suboptimalna rešenja

Test instanca	GA				MA			
	Rešenje	$t(s)$	*agap*	σ	Rešenje	$t(s)$	*agap*	σ
mr1_1L_500.500	2349.86	74.85	0.000	0.000	2349.86	11.73	0.000	0.000
mr1_2L_160_340.500	6707.51	83.12	0.611	0.988	6707.51	5.73	0.000	0.000
mr1_3L_55_120_325.500	10911.32	76.01	1.341	0.814	10911.32	7.25	0.734	0.788
mr1_4L_30_65_140_265.500	15311.47	61.23	1.544	0.879	**15237.26**	23	0.871	0.418
ms1_1L_1000.1000	4378.63	534.89	0.000	0.000	4378.63	29.4	0.000	0.000
ms1_2L_320_680.1000	13416.81	540.53	0.510	0.485	**13361.39**	22.86	0.303	0.346
ms1_3L_120_250_630.1000	21881.38	501.03	2.260	1.312	21881.38	117.23	1.128	0.871
ms1_4L_64_128_256_552.1000	30936.59	418.52	2.258	1.019	**30902.74**	119.21	1.355	1.024
ms1_5L_25_55_120_250_550.1000	40191.23	396.69	1.738	1.118	**40070.04**	136.01	0.543	0.514
mt1_1L_2000.2000	9176.51	4134.33	0.000	0.000	9176.51	350.32	0.000	0.000
mt1_2L_650_1350.2000	27733.06	3949.57	0.331	0.704	27733.06	819.98	0.012	0.021
mt1_3L_256_600_1144.2000	46095.09	3247.06	2.021	1.105	46095.09	858.48	1.266	0.770
mt1_4L_120_250_520_1110.2000	65044.00	3084.89	1.126	0.649	**64953.25**	1108.5	0.667	0.400
mt1_5L_60_120_250_500_1070.2000	83523.75	2944.08	1.678	0.830	**83404.33**	916.77	0.997	0.692
prosek	-	1431.91	1.101	0.707	-	323.32	0.562	0.417

od vremena izvršavanja GA. Uzimajući u obzir da je veći deo MA paralelizovan a GA nije, kao i razliku u procesorskoj snazi računara na kojima su testirani, to ukazuje da su ove dve metode ravnopravne po brzini izvršavanja.

3.3 Hibridna metoda za rešavanje hab lokacijskog problema sa jednostrukom alokacijom i sa ograničenjem kapaciteta

U poglavlju 2.3 definisan je i predstavljen hab lokacijski problem sa jednostrukom alokacijom i sa ograničenjem kapaciteta (eng. Capacitated Single Allocation Hub Location Problem, skr. CSAHLP). Matematički model, kao i sve oznake i pretpostavke iz tog poglavlja koriste se i u tekstu koji sledi.

S obzirom da su uvedeni maksimalni dozvoljeni kapaciteti saobraćaja koji može biti usmeren na neki hab, postoji mogućnost da neka kombinacija habova $H \subseteq I$ ne može predstavljati dopustivo rešenje, bez obzira kako je alokacija urađena. To ograničenje čini problem znatno složenijim i zahteva nov pristup pri rešavanju.

Hibridni evolutivni algoritam sa metodom grananja i ograničavanja (u daljem tekstu EA-BnB) za rešavanje CSAHLP sastoji se od genetskog algoritma i metode grananja i ograničavanja koja je primenjena na specifičan način. Osnovna ideja jeste da se pomoću genetskih operatora stvaraju jedinke kojima se rešava pitanje lokacije habova a da se zatim BnB metodom vrši alokacija. Radi povećanja efikasnosti algoritma, alokacija se pomoću BnB ne vrši za sve jedinke, već samo za pojedine. Za većinu jedinki alokacija se obavlja algoritmom $ProstaAlokacija$ koji na jednostavan način u polinomskom vremenu izvršavanja nalazi dobro, dopustivo rešenje.

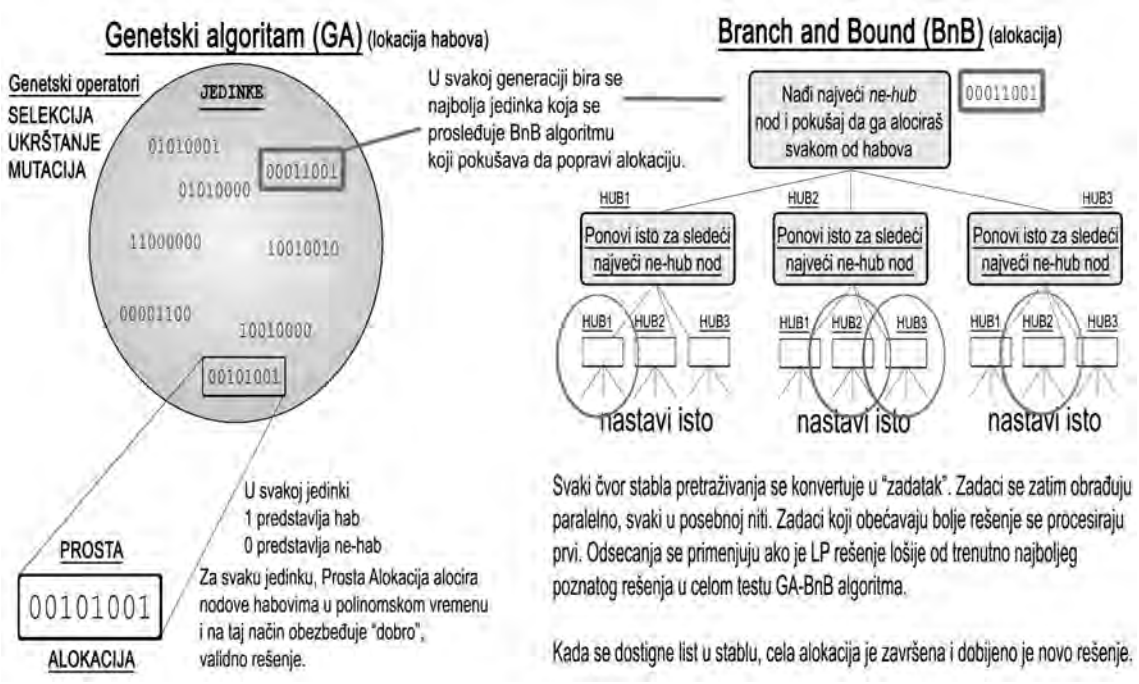
Na slici 3.2 je dat shematski prikaz ove ideje, dok je algoritmom 8 ta ideja precizno zapisana.

Nakon unošenja ulaznih podataka, vrši se određeno preprocesiranje podataka koje će u daljem tekstu biti objašnjeno. Početna populacija se generiše na pseudoslučajan način. Zatim se ulazi u petlju u kojoj se koriste dve pomoćne promenljive: $NajboljaJedinka$ čuva vrednost globalno najbolje jedinke u celom izvršavanju algoritma, a *kandidat* pamti koja jedinka ima najbolju vrednost funkcije cilja dobijenu pomoću algoritma $ProstaAlokacija$ u tekućoj generaciji EA-BnB.

Keširanje se koristi da se algoritmi BnB i $ProstaAlokacija$ ne bi pozivali više puta za istu jedinku. Ukoliko je jedinka J nova i nema je u kešu, poziva se algoritam $ProstaAlokacija(J)$. Taj algoritam u polinomskom vremenu nalazi dobro rešenje alokacije. Tom prilikom, ažurira se vrednost promenljive *kandidat*, ukoliko je vrednost funkcije cilja jedinke J manja od vrednosti jedinke *kandidat*. U slučaju

Algoritam 8: Hibridni algoritam EA-BnB

```
UlazniPodaci();
Preprocesiranje();
PseudoslučajnoGenerisanjePopulacije();
NajboljaJedinka = ∅;
repeat
    kandidat = ∅;
    foreach jedinka J ∈ populacija P do
        resenje = Cache(J);
        if resenje == ∅ then
            /*ProstaAlokacija menja kandidat                    */
            resenje = ProstaAlokacija(J);
            /*BnB menja NajboljaJedinka                         */
            if resenje == ∅ then
                resenje = BnB(J, POPRAVKA)
            end
            if resenje! = ∅ then DodajCache(J, resenje);
            else izbaci J iz P;
        end
    end
    resenje =BnB(kandidat, POPRAVKA);
    if resenje! = ∅ then PromeniCache(kandidat, resenje);
    primena operatora:
        Prilagođenost;
        Selekcija;
        Ukrštanje;
        Mutacija;
    NovaGeneracija();
until KrajRadaEA();
resenje = BnB(NajboljaJedinka, ZAVRSNO);
```



Slika 3.2: Shematski prikaz EA-BnB algoritma

da $ProstaAlokacija$ nije uspela da nađe dopustivo rešenje za jedinku J, poziva se algoritam $BnB(J, POPRAVKA)$. Drugi parametar u pozivu BnB određuje tip pretrage, što će biti objašnjeno u poglavlju 3.3.4. Ukoliko ni BnB nije pronašao rešenje, jedinka J se izbacuje iz populacije.

Nakon što je alokacija izvršena za sve jedinke, jedinka koja je najbolji *kandidat* u ovoj generaciji se prosleđuje algoritmu BnB, kojim se popravlja alokacija te jedinke. BnB uvek traži alokaciju koja daje vrednost funkcije cilja bolju od vrednosti jedinke $NajboljaJedinka$. Ukoliko to nije moguće, jedinka *kandidat* ostaje u populaciji, ali se njena alokacija ne menja. Ovaj korak se može posmatrati kao lokalna pretraga, kojom se u svakoj generaciji pretražuje okolina jedne jedinke i to one koja ima najbolju vrednost funkcije. BnB se poziva samo za jedinke za koje je inicijalna alokacija dobijena pomoću algoritma $ProstaAlokacija$. BnB se u toku svih generacija nikada ne poziva za istu jedinku više od jedanput.

Posle alokacije i lokalne pretrage algoritmom BnB, primenjuju se standardni genetski operatori: prilagođenost, selekcija, ukrštanje i mutacija, a zatim se pravi nova generacija jedinki.

Konačno, kada se završi evolutivni deo algoritma, ponovo se poziva BnB za jedinku $NajboljaJedinka$, sa drugim tipom pretrage. Taj tip pretrage dopušta

algoritmu BnB da bolje istraži prostor rešenja i nađe bolju alokaciju. Ukoliko do toga dođe, jedinka $NajboljaJedinka$ se ažurira i vrednost funkcije cilja te jedinke postaje rešenje problema.

3.3.1 Kodiranje

Jedna od ideja hibridne EA-BnB metode jeste da se alokacija vrši pomoću algoritma $ProstaAlokacija$ kad god je to moguće. U suprotnom, alokacija se vrši pomoću algoritma BnB. U svakom slučaju, genetski kôd i genetski operatori se bave lokacijom habova, ali ne i alokacijom. Zbog toga, genetski kôd sadrži niz od $n = |I|$ bitova, od kojih svaki određuje da li je neki čvor hab ili ne.

Kada se pročita genetski kôd jedinke i odrede habovi, pod uslovom da je bar jedan hab uspostavljen, alokacija se određuje algoritmom $ProstaAlokacija$ ili, ako to nije moguće, algoritmom BnB. Istovremeno se računaju i ukupni troškovi, odnosno funkcija cilja. Ukoliko ni jedan ni drugi algoritam ne mogu da odrede alokaciju čvorova habovima, jedinka se proglašava za nevažeću i izbacuje se iz populacije.

3.3.2 Genetski operatori

Genetski operatori prilagođenosti, selekcije, ukrštanja i mutacije, koji su korišćeni u memetskom algoritmu za rešavanje USAHLP (sekcija 3.1.3), primenjeni su i u hibridnom evolutivnom algoritmu EA-BnB (algoritam 8) sa istim vrednostima parametara.

3.3.3 Algoritam za nalaženje alokacije čvorova habovima

Funkcija cilja (2.1) je takva da prikupljanje i distribucija, koji su dati sumom $\sum_i(\chi O_i + \delta D_i)C_{ik}$ (i je čvor, k je hab) znatno više utiču na ukupnu vrednost nego transfer $\alpha \sum_i \sum_j W_{ij}C_{kl}$. Za to postoje dva razloga: prvo, $\alpha \leq \chi, \delta$, što znači da se ukupan transfer množi manjim koeficijentom nego prikupljanje i distribucija; drugo, ako su dva čvora alocirana istom habu ($k = l$), trošak transfera je 0, jer $\forall k, C_{kk} = 0$. Zbog toga je intuitivno jasno da će u najboljim rešenjima alokacije čvor i biti alociran onom čvoru k za koje je vrednost C_{ik} što manja, jer to i sumu $\sum_i(\chi O_i + \delta D_i)C_{ik}$ čini malom. Drugim rečima, čvorovi će često biti alocirani sebi najbližim habovima, ukoliko je to moguće.

Ideja algoritma *ProstaAlokacija* se zasniva upravo na tome: alocirati čvor njemu najbližem habu, ukoliko kapacitet tog haba to dopušta; u suprotnom, pokušati sa sledećim habom po udaljenosti itd.

Algoritam 9: ProstaAlokacija

ulaz : jedinka J
izlaz: rešenje za J: (alokacije Z_{ik}, vrednost funkcije cilja)
$H = Habovi(J)$;
/*Niz T pamti iskorišćene kapacitete habova; */
/*Zbog ograničenja kapaciteta, uvek mora važiti $T_k \leq G_k$. */
foreach $k \in H$ **do**
  /*Hab je alociran samom sebi. */
  $Z_{kk} = 1$;
  $T_k = O_k$;
end
foreach $i \in SortiranI, i \notin H$ **do**
  /*Poređaj H u rastućem poretku udaljenosti $C_{ik}, \forall k \in H$ */
  $SortiranH = Sort(H)$;
  /*Nađi najbliži hab kome mogu da alociram i */
  foreach $k \in SortiranH$ **do**
    /*Ako alociranjem čvora i kapacitet haba k nije premašen */
    if $T_k + O_i \leq G_k$ **then**
      $Z_{ik} = 1$; /*alociraj i habu k */
      $T_k += O_i$; /*ažuriraj iskorišćen kapacitet haba k */
      break;
    end
  end
  if *i nije alociran* **then**
    return $\emptyset$;
  end
end
/*Ažuriraj jedinku sa najboljom alokacijom dobijenom pomoću algoritma ProstaAlokacija */
if $Cilj(J) < Cilj(kandidat)$ **then** $kandidat = J$;
return (alokacije Z_{ik}, $Cilj(J)$);

Iz jedinke J se pročita koji su habovi uspostavljeni. Prvo se izvrši alokacija habova, svaki hab se alocira sam sebi. Pri tom se postave početne vrednosti pomoćnog niza $T_k, k \in H$ pomoću koga se vodi računa da se nikada ne premaše kapaciteti habova.

Problem alokacije, kada su dati maksimalni kapaciteti habova, podseća na problem ranca [48]. Često primenjivana strategija pri rešavanju problema ranca jeste

gramziva (eng. greedy) strategija, koja je i ovde primenjena. Vrednost O_i određuje koliko hab i opterećuje saobraćajem hab kome je alociran, odnosno koliko njegovog kapaciteta zauzima. Alokacija čvorova $i \in I \setminus H$ se zbog toga vrši po opadajućem redosledu vrednosti O_i. Korist od primene te strategije je dvostruka: prvo, ne-hab čvorovi sa velikim tokom izlaznog saobraćaja O_i će prvi biti alocirani, dok su habovi još uvek relativno „prazni“, te je veća verovatnoća da će svi čvorovi uspešno biti alocirani; drugo, očekuje se da vrednost funkcije cilja dobijena tim postupkom bude manja, jer će velike vrednosti O_i biti pomnožene najmanjim mogućim vrednostima C_{ik}.

U Algoritmu 8, prilikom preprocesiranja se formira niz $SortiranI$ koji sadrži indekse svih čvorova $i \in I$ poređane po opadajućem redosledu vrednosti O_i. Alokacija se obavlja redom za svaki element i niza $SortiranI$, osim za habove. Za svaki čvor i formira se niz $SortiranH$ koji sadrži indekse svih habova $k \in H$ poređane po rastućem redosledu udaljenosti C_{ik}. Pokušava se sa alokacijom čvora i habu $k \in SortiranH$: ukoliko dodavanje izlaznog saobraćaja O_i na vrednost T_k ne prelazi kapacitet haba k, vrši se alokacija i ažurira vrednost T_k. Ako i zbog ograničenja kapaciteta ne može biti alociran nijednom habu, proces alokacije se prekida i označava se da algoritam $ProstaAlokacija$ ne može da nađe dopustivo rešenje za jedinku J.

Kada su sve alokacije uspešno obavljene, proverava se da li je vrednost funkcije cilja jedinke J, sa ovako izvršenom alokacijom, bolja od vrednosti prethodno najbolje jedinke iz tekuće generacije. Ukoliko jeste, najbolja jedinka *kandidat* se ažurira.

Ako je $h = |H|$, tada je, zbog sortiranja $Sort(H)$ u svakom koraku, složenost ovog algoritma $O((n-h) \cdot h \cdot ln(h))$.

3.3.4 Paralelni BnB algoritam za rešavanje potproblema

Algoritam grananja i ograničavanja (BnB) služi da izvrši alokaciju čvorova habovima, odnosno, nalazi dopustiva rešenja potproblema CSAHLP(H) opisanog u sekciji 2.4. Način na koji je BnB ovde implementiran je specifičan iz nekoliko razloga:

- BnB ne pretražuje ceo prostor rešenja, već prekida sa radom posle određenog broja čvorova koje je obišao.
- Kao gornja granica se uvek koristi vrednost funkcije cilja najbolje poznate jedinke u celom izvršavanju EA-BnB. Drugim rečima, algoritam BnB uvek

traži alokaciju koja bi rezultirala vrednošću funkcije cilja boljom od najbolje poznate. Ukoliko ne može da pronađe takvu alokaciju, BnB prekida sa radom.

- BnB je paralelizovan, pri čemu je prvenstveni cilj paralelizacije mogućnost pronalaženja kvalitetnijeg rešenja, jer sa porastom broja niti koje se paralelno izvršavaju, povećava se i deo prostora rešenja koji je pretražen. Cilj paralelizacije koja je primenjena nije ubrzanje.

- BnB radi u dva režima: prvi, koji prekida sa radom čim nađe prvo rešenje bolje od zadate gornje granice (parametar $POPRAVKA$) i drugi, koji se poziva na kraju rada EA-BnB i služi da dublje ispita prostor rešenja najbolje jedinke i nađe što bolju alokaciju (parametar $ZAVRSNO$).

BnB je prikazan algoritmom 10.

Algoritam 10: Paralelizovani algoritam grananja i ograničavanja - BnB

```
ulaz : jedinka J
ulaz : parametar TipPretrage
izlaz: rešenje za J
Inicijalizacija();
resenje = ∅;
zadatak = ∅;
Push(zadatak);
parallel while not KrajRadaBnB() do
    if (PostojiZadatak()) then
        zadatak = Pop();
        /*Grananje menja resenje                    */
        Grananje(J, zadatak);
    end
end
return resenje;
```

BnB je implementiran kao paralelan proces sa više niti. Svaka nit obilazi jedan deo stabla pretraživanja na način koji će biti opisan. Niti se razdvajaju blizu korena stabla i u svakoj niti se obilazi različita grana. Za određivanje donje granice se koristi LP relaksacija potproblema CSAHLP(H).

Prilikom pretraživanja, od određenih čvorova stabla se napravi „zadatak“, koji se stavlja (eng. push) na listu koja je implementirana kao stek (eng. stack). Sve niti koriste istu listu zadataka. „Zadatak“ je struktura koja sadrži alokacije čvorova habovima koje su do tog trenutka urađene: to je skup uređenih parova $(i, k), i \in$

$I \setminus H, k \in H$. Tih alokacija može biti proizvoljan broj, 0 ili više. Na taj način, svaki „zadatak“ sadrži sve informacije koje su potrebne da se nastavi obilaženje stabla od onog čvora u kome je „zadatak“ kreiran.

Rutina $Inicijalizacija()$ napravi određen broj niti, inicijalizuje ih i stavi u stanje čekanja na zadatak. Takođe, tom prilikom se pripremi okruženje za rešavanje LP relaksacije problema $CSAHLP(Habovi(J))$ kako to ne bi moralo da se radi pri svakom pozivu LP rešavača. Zatim se na listu zadataka stavlja prazan zadatak, sa 0 obavljenih alokacija. Svaka nit, koja čeka (ne obrađuje stablo), prati listu zadataka (funkcija $PostojiZadatak()$ u Algoritmu 10) i čim se na listi pojavi neki zadatak, nit preuzme zadatak sa liste (eng. pop) i nastavi da obrađuje stablo od čvora koji tom zadatku odgovara (funkcija $Grananje(J, zadatak)$ u Algoritmu 10). Prilikom implementacije se vodi računa o potencijalnim problemima konkurentnosti: jedan zadatak može preuzeti samo jedna nit, tako da prvi, prazan zadatak, preuzima jedna nit dok ostale nastavljaju da čekaju.

Funkcijom $Grananje(J, zadatak)$ se stvaraju novi zadaci, a niti koje čekaju ih preuzimaju i proces se nastavlja, sve dok funkcija $KrajRadaBnB()$ ne signalizira prekid rada.

Algoritam BnB (algoritam 10) predstavlja osnovni mehanizam, pomoću koga se vodi računa kada treba prekinuti sa radom, zatim, da niti preuzimaju zadatke na željeni način i da ne dođe do problema konkurentnosti. Pretraživanje stabla (grananje i ograničavanje), alokacija i stvaranje novih zadataka se vrše procedurom $Grananje$ koja je prikazana Algoritmom 11.

Funkcija $KrajRadaBnB()$ se koristi i u algoritmu 10 i u algoritmu 11. Ona služi da jedna ili više niti prekine sa daljim pretraživanjem kada se ispuni neki od sledećih uslova po navedenom redosledu:

1. Ako je broj već izvršenih alokacija u rutini $Grananje$ veći od 95% (dakle, veći od $.95n$), pretraživanje se uvek nastavlja. Očekuje se da će novo rešenje vrlo brzo biti pronađeno, uz samo nekoliko ($0.05n$) poziva LP rešavača, te se pretraživanje ne prekida.

2. Ako je vrednost parametra $TipPretrage$ jednaka $POPRAVKA$, pretraživanje se prekida čim se nađe prvo dopustivo rešenje. Ako je $TipPretrage == ZAVRSNO$, pretraživanje se nastavlja.

Algoritam 11: Rekurzivna procedura - Grananje

ulaz : jedinka J
ulaz : skup alokacija *zadatak*
if *KrajRadaBnB()* **then**
  return;
end
$(vrednost, rezultati) = LPResenje(Habovi(J), zadatak)$;
if $Cilj(NajboljaJedinka) < vrednost$ **then**
  return;
end
if *AlokacijaGotova(zadatak)* **then**
  $resenje = (zadatak, vrednost)$;
  $NajboljaJedinka = J$;
  $ObrisiZadatke()$;
  return;
end
i =prvi čvor koji nije alociran, po redosledu sortiranih čvorova $SortiranI$;
`/*Odredi podskup rezultata iz LP rešenja */`
`/*koja odgovaraju čvoru` i`: */`
$HabRezultati = \{(k, Z_{ik}) \in rezultati \mid k \in Habovi(J)\}$;
`/*sortiraj` $HabRezultati$ `u opadajućem redosledu vrednosti` Z_{ik} `*/`
$SortiraniHabRezultati = Sort(HabRezultati)$;
foreach $(k, Z_{ik}) \in SortiraniHabRezultati$ **do**
  if $ImaMestaZaZadatak(zadatak, k, Z_{ik})$ **then**
    $novi = Prepisi(zadatak)$;
    $novi \leftarrow (i, k)$;
    $Push(novi)$;
    označi k *zavrsen*;
  end
end
foreach $(k, Z_{ik}) \in SortiraniHabRezultati$ **do**
  if k *nije zavrsen* **then**
    $novi = Prepisi(zadatak)$;
    $novi \leftarrow (i, k)$;
    $Grananje(J, novi)$;
  end
end

3. Ukoliko je ukupan broj čvorova stabla pretraživanja, koje su obišle sve niti zajedno, veći od zadate vrednosti, prekida se sa pretraživanjem. Ta vrednost je postavljena na $2 \cdot n \cdot nt$ kada je $TipPretrage == POPRAVKA$, odnosno $2 \cdot n \cdot nt^2$ kada je $TipPretrage == ZAVRSNO$. Međutim, ukoliko je $TipPretrage == ZAVRSNO$, svaki put kada se pronađe novo rešenje, brojač pretraženih čvorova se postavi na 0, što kao posledicu ima da se u završnom pozivu BnB, pretraživanje nastavlja sve dok algoritam ne obiđe $2 \cdot n \cdot nt^2$ čvorova bez popravke najboljeg rezultata.

4. Ukoliko je $TipPretrage == POPRAVKA$, proverava se da li je list stabla pretraživanja udaljen od trenutnog čvora stabla više nego što dopušta ograničenje postavljeno stavkom 3. U tom slučaju, prekida se sa radom, jer će uslov 3. svakako biti ispunjen u nekom od narednih koraka. Treba primetiti da neke niti mogu ispuniti ovaj uslov, a neke ne i samo one koje ga ispune prekidaju sa radom. Ako je $TipPretrage == ZAVRSNO$, ovaj uslov se ne koristi, jer neka druga nit može pronaći novo rešenje i postaviti brojač na 0.

Nakon provere da li treba prekinuti sa radom, rutina *Grananje* poziva LP rešavač problema CSAHLP(Habovi(J)). Prethodno se fiksiraju vrednosti promenljivih Z_{ik} tog problema, koje su već izvršene i nalaze se u promenljivoj *zadatak*. LP rešavač nalazi donju granicu vrednosti funkcije cilja, sa habovima $Habovi(J)$ i alokacijama $Z_{ik} \in zadatak$, označenu promenljivom *vrednost*. Ukoliko ne postoji dopustivo LP rešenje, koristi se oznaka $vrednost = \infty$. Zatim se proverava da li je gornja granica manja od donje granice *vrednost*. Ako jeste, dolazi do ograničavanja, odnosno pretraživanje ne ide dalje, već se vraća unazad (eng. backtracking). Ovo je jedna od ključnih strategija implementiranih u celom EA-BnB algoritmu, zato što se za gornju granicu uzima vrednost funkcije cilja najbolje jedinke iz cele populacije. Dakle, BnB uvek traži dopustivo rešenje za jedinku J, takvo da ono ima vrednost funkcije cilja bolju od vrednosti funkcije cilja jedinke $NajboljaJedinka$, ako postoji. BnB ne traži bilo koje dopustivo rešenje, za to je zadužen algoritam $ProstaAlokacija$. Ova strategija dramatično poboljšava performanse celog algoritma: jako često BnB prekida sa radom vrlo brzo, na samom početku ili nakon nekoliko pretraženih čvorova, jer utvrdi da jedinka J ne može imati bolje rešenje od jedinke $NajboljaJedinka$. Međutim, u onim slučajevima kada BnB nađe novo rešenje, to znači da jedinka J postaje nova $NajboljaJedinka$.

Ukoliko ne dođe do ograničavanja, proverava se da li je trenutni čvor list stabla, odnosno da li su sve alokacije izvršene. Ako jesu, to znači da je trenutno LP rešenje ujedno i rešenje MILP problema, odnosno, da trenutni *zadatak* predstavlja novo rešenje jedinke J koje ne samo da je dopustivo već, zbog prethodno izloženog, ima i najbolju *vrednost* u celoj populaciji. Pre povratka unazad, poziva se rutina $ObrisiZadatke()$ koja sa liste zadataka briše sve zadatke čija je gornja granica lošija od novodobijenog najboljeg rešenja.

Za čvorove stabla koji nisu listovi, grananje se nastavlja. Od svih čvorova koji nisu alocirani, bira se sledeći čvor i po već opisanom redosledu $SortiranI$. Pored vrednosti funkcije cilja, LP rešavač kao rezultat vraća i vrednosti svih promenljivih $0 \leq Z_{ik} \leq 1$, $i \in I \setminus H$, $k \in H = Habovi(J)$. Iz tih rezultata izdvaja se podskup koji odgovara izabranom čvoru i a zatim se sortira u opadajućem redosledu vrednosti $0 \leq Z_{ik} \leq 1$. Dakle, pokušava se alokacija izabranog čvora i habovima k po opadajućem redosledu vrednosti realnih promenljivih Z_{ik}. Prva će biti obrađena promenljiva koja ima vrednost najbližu broju 1.

U prvom prolazu kroz sortirane rezultate, proverava se da li ima mesta za nove zadatke na listi zadataka. Ukoliko ima, za neko (k, Z_{ik}) se formira novi zadatak i stavlja se na listu zadataka. Pri tom se obeleži da je alokacija čvora i habu k u ovom pozivu rutine $Grananje$ završena. Stavljanje zadataka na listu će aktivirati niti koje u rutini BnB čekaju na nove zadatke.

U drugom prolazu kroz sortirane rezultate formiraju se novi zadaci, tako što se za sve habove k, za koje nije obeleženo da je alokacija čvora i tom habu završena u prvom prolazu, novi zadatak formira kopiranjem trenutnog zadatka i dodavanjem alokacije (i, k). Novi zadaci će biti obrađeni rekurzivnim pozivanjem rutine $Grananje$. Ovo je „klasičan“ deo metode grananja i ograničavanja, kao u algoritmu 2.

Rutine $ImaMestaZaZadatak$ i $Push$ određuju redosled kojim će zadaci biti obrađivani. Samim tim, one značajno, ako ne i presudno, utiču na performanse celog BnB algoritma.

Način rada rutine $ImaMestaZaZadatak$ je ilustrovan slikom 3.2. Za koren stabla (označen kao nivo 1) i za sve njegove direktne potomke (nivo 2) svi zadaci se stavljaju na listu zadataka. U opštem slučaju, na taj način će biti napravljeno h zadataka na nivou 2 i h^2 zadataka na nivou 3, gde je $h = |H|$. Počev od nivoa 4,

rutina $ImaMestaZaZadatak$ prati sledeća pravila, navedenim redosledom:

1. Novi zadatak, koji odgovara paru (i, k) za prvu vrednost k (dakle, za hab k za koje je Z_{ik} najveće), se nikada ne stavlja na listu zadataka. LP rešenje ukazuje da je taj hab k potencijalno najbolji za alokaciju čvora i i zato će biti odmah obrađen od strane trenutne niti, bez stavljanja na listu zadataka.

2. Ako je $Z_{ik} = 0$, odgovarajući zadatak se ne stavlja na listu zadataka, on čeka da vrednosti k za koje je $Z_{ik} \neq 0$ budu obrađene. On će biti obrađen kada se trenutna nit vrati unazad. Međutim, verovatno će neki od kriterijuma zaustavljanja rada BnB pre toga biti ispunjen. U opštem slučaju, moguće je da u optimalnom MILP rešenju bude $Z_{ik} = 1$ i ako je $Z_{ik} = 0$ u LP rešenju. Zato se ovaj čvor ne ignoriše potpuno, ali se obrađuje poslednji, ako kriterijumi zaustavljanja to dopuste.

3. Ako je broj zadataka na listi zadataka manji od $2 \cdot nt$, gde je nt broj niti, onda se zadatak stavlja na listu zadataka. Ovim se obezbeđuje da na listi zadataka uvek bude dovoljno zadataka za sve niti, odnosno da niti nikada bespotrebno ne čekaju.

Već je rečeno da rutina Pop uzima prvi zadatak sa liste zadataka, što znači da redosled izvršavanja zadataka zavisi od redosleda u kojem ih rutina $Push$ postavlja na listu zadataka. Pravila koja rutina $Push$ koristi pri postavljanju zadataka na listu su sledeća:

- Zadaci sa nižih nivoa (bližih korenu stabla) se uvek stavljaju pre svih zadataka sa viših nivoa. Na taj način se obezbeđuje da će niti obrađivati grane koje počinju blizu korena.

- Zadaci sa istog nivoa se ređaju na listu zadataka u rastućem poretku vrednosti so (eng. sort order) koja se računa za svaki čvor posebno na osnovu jednog od parametara algoritma, koji se označava sa tso (eng. task sort order) i koji može imati vrednosti 0, 1 i 2. Ako je $tso = 0$, to znači da je ređanje zadataka isključeno, odnosno, svi zadaci imaju istu vrednost $so = 0$, što znači da će biti obrađivani onim redosledom kojim su kreirani. Ako je parametar $tso = 1$, tada je $so = vrednost \cdot (1 - Z_{ik})$, gde $vrednost$ označava vrednost funkcije cilja dobijenu od LP rešavača. Ova formula favorizuje zadatke za koje je

istovremeno i vrednost funkcije cilja mala i vrednost kontinualne promenljive Z_{ik} bliska jedinici. Konačno, ako je parametar $tso = 2$, onda je $so = 1 - Z_{ik}$, dakle, favorizuju se zadaci za koje je Z_{ik} bliska jedinici. Rezultati sa različitim vrednostima parametra tso, kao i drugih parametara cele metode, prikazani su u poglavlju 3.3.5.

Za ilustraciju rada algoritma BnB može poslužiti primer kada je broj niti $nt = 4$ (slika 3.2). Zbog načina na koji rutine $ImaMestaZaZadatak$ i $Push$ stavljaju zadatke na listu zadataka, celo stablo za prva 3 nivoa će biti kreirano, sa h^2 zadataka, $h = |H|$. Svaka od 4 niti počne da obrađuje po jedan od tih zadataka sa nivoa 3. Koje će zadatke uzeti zavisi od parametra tso i redosleda kojim ih je rutina $Push$ stavila na listu. Ukoliko bi sve niti mogle da stignu do lista u podstablu koje obrađuju, ukupno bi obišle $(|I| - |H| - 3) * nt$ čvorova. S obzirom da je $|I| \gg |H|$ i $|I| \gg 3$, taj broj je blizak vrednosti $|I| \cdot nt$. To je razlog zbog koga je u rutini $KrajRadaGa$ stavljeno ograničenje na $|I| \cdot nt \cdot 2$ čvorova kada je $TipPretrage == POPRAVKA$: faktor 2 omogućava da ipak bude nekog pretraživanja unazad, ako dođe do ograničavanja. Ako je $TipPretrage == ZAVRSNO$, faktor koji omogućava pretraživanje unazad je $2 \cdot nt$, što svim nitima omogućava da obiđu još veći deo stabla. Redosled po kome se zadaci stavljaju na listu zavisi od parametra tso i predviđeno je da taj redosled favorizuje one zadatke koji potencijalno vode ka boljem rešenju.

3.3.5 Rezultati

Hibridni evolutivni algoritam sa metodom grananja i ograničavanja implementiran je u programskom jeziku C, pri čemu je korišćen OpenMP za paralelizovanje BnB algoritma. Univerzalni rešavač CPLEX 12.1 je korišćen za rešavanje LP relaksacije problema CSAHLP(H). Svi testovi na instancama, navedenim u poglavlju 3.1.7, su obavljeni na računaru zasnovanom na procesoru Intel Core i5-3470 3.2 GHz sa 8GB RAM pod Windows 7 Professional operativnim sistemom. U literaturi ne postoji praksa testiranja CSAHLP instanci sa različitim faktorima fiksnih troškova, tako da se skup testova za CSAHLP sastoji od 30 test instanci u kojima se broj čvorova kreće od 10 do 300.

Evolutivni algoritam je implementiran sa sledećim kriterijumima zaustavljanja: maksimalni broj generacija $G_{max} = 200$ i maksimalni broj generacija bez poboljšanja najboljeg rešenja: $R_{max} = 50$. Korišćen je promenljivi broj jedinki u populaciji, u

zavisnosti od dimenzije problema: 100 jedinki za $n \leq 50$, 150 jedinki za $n = 100$ i 200 jedinki za $n = 200$ i $n = 300$. Ovi parametri su određeni preliminarnim testiranjem. Kako navedeni procesor ima 4 fizička jezgra, korišćena je vrednost $nt = 4$ kao parametar BnB algoritma.

Obavljeno je i preliminarno testiranje algoritma sa različitim vrednostima sledećih parametara:

- Parametar *tso* (vrednosti 0, 1, 2), kojim se određuje redosled zadataka kojim rutina $Push$ stavlja zadatke na listu;
- Parametar *sortI* (vrednosti *uklj.*, *isklj.*), kojim se određuje da li se čvorovi $i \in I$ obrađuju po opadajućem redosledu vrednosti O_i u rutinama $ProstaAlokacija$ i $Grananje$;
- Parametar *sortH* (vrednosti *uklj.*, *isklj.*), kojim se određuje da li se rezultati realnih promenljivih Z_{ik} sortiraju u opadajućem redosledu u rutini $Grananje$.

Svih 12 kombinacija parametara je testirano na podskupu od 16 test instanci i korišćenjem 10 različitih vrednosti za inicijalizaciju generatora pseudoslučajnih brojeva. Rezultati su prikazani u tabeli 3.13. Pored vrednosti parametara, u koloni BrReš je dat broj instanci na kojima je postignuto najbolje rešenje (maksimum 16), zatim u koloni PrReš je dat prosečan broj postignutih najboljih rešenja (maksimum 10), zatim prosečno vreme izvršavanja u sekundama (t(s)) i prosečna vrednost prosečne udaljenosti *agap* (eng. average gap).

Tabela 3.13: Podešavanje parametara *sortI*, *sortH* i *tso*

sortI	*sortH*	*tso*	BrReš	PrReš	t(s)	agap (%)
isklj.	isklj.	0	6	2.88	15.507	0.607
isklj.	isklj.	1	6	2.88	14.224	0.555
isklj.	isklj.	2	6	2.88	15.260	0.574
isklj.	uklj.	0	8	3.81	13.339	0.652
isklj.	uklj.	1	9	4.31	13.922	0.639
isklj.	uklj.	2	9	4.19	13.041	0.694
uklj.	isklj.	0	11	5.38	8.138	0.475
uklj.	isklj.	1	12	5.63	8.315	0.530
uklj.	isklj.	2	12	5.50	7.900	0.514
uklj.	uklj.	0	14	6.69	6.737	0.480
uklj.	uklj.	1	**16**	7.31	6.908	0.470
uklj.	uklj.	2	15	6.81	7.006	0.469

Parametri $sortI$ i $sortH$ značajno utiču na performanse, jer se njima određuje da li će neka strategija biti primenjena ili ne. Korišćenje ili nekorišćenje gramzive strategije u rutini $ProstaAlokacija$, određeno parametrom $sortI$, imalo je najveći uticaj, kako na kvalitet i stabilnost rešenja tako i na vreme izvršavanja. Kada je isključeno sortiranje čvorova po veličini izlaznog saobraćaja O_i, rutina $ProstaAlokacija$ ređe pronalazi dopustiva rešenja, što znači da se BnB češće poziva. To ne samo što povećava vreme izvršavanja, već rezultira manjim brojem kvalitetnih jedinki, jer BnB uvek traži rešenje bolje od najboljeg poznatog. Zaključak je da sortiranje niza I mora biti uključeno.

Parametar $sortH$ više utiče na kvalitet i stabilnost rešenja, a manje na brzinu izvršavanja. Sortiranje niza $HabRezultati$ u rutini $Grananje$ znači da će vrednost promenljive Z_{ik} biti fiksirana na 1 prvo za ono k za koje realna promenljiva Z_{ik} ima najveću vrednost, blisku ili jednaku 1. Ovi rezultati pokazuju da ta strategija češće dovodi do rešenja, a pri tom i povećava brzinu izvršavanja.

Kada je $tso = 0$, odnosno kada rutina $Push$ stavlja zadatke na listu onim redom kojim se zadaci kreiraju, rezultati su znatno slabiji nego kada je vrednost parametra $tso = 1$ ili $tso = 2$. Razlika u rezultatima kada je $tso = 1$ ili $tso = 2$ je mala, ali je ipak samo kombinacija parametara sa uključenim sortiranjem oba niza i vrednošću $tso = 1$ dovela do svih 16 najboljih rešenja. Zato je za sve ostale testove izabrana vrednost parametra $tso = 1$, odnosno, stavljanje zadataka na listu u rastućem redosledu vrednosti formule $vrednost \cdot (1 - Z_{ik})$. Posle određivanja vrednosti parametara, izvršen je test na svih 30 instanci. Pri tom je svaka instanca testirana 20 puta, svaki put sa različitom vrednošću parametra inicijalizacije generatora pseudoslučajnih brojeva.

U tabeli 3.14 su prikazani rezultati na AP instancama sa $10 \leq n \leq 50$ čvorova. Kolone u tabeli označavaju:

- Ime test instance, koje sadrži broj čvorova i dvoslovnu oznaku za tip fiksnih troškova i tip kapaciteta: „L“ znači „lako“, „T“ znači „teško“; na primer, 50LT označava instancu sa 50 čvorova, „lakim“ fiksnim troškovima i „teškim“ ograničenjima kapaciteta;
- Optimalno rešenje koje je pronašao univerzalni rešavač CPLEX;
- Vreme potrebno da CPLEX pronađe i dokaže optimalnost rešenja (u sekun-

dama);

- Najbolje rešenje koje je pronašao EA-BnB; *opt* označava isto kao CPLEX;
- Prosečno vreme potrebno da EA-BnB dostigne najbolje rešenje - $t(s)$ (u sekundama);
- Prosečno vreme potrebno da EA-BnB završi sa radom - $t_{kraj}(s)$ (u sekundama);
- Prosečan broj generacija potreban da EA-BnB nađe najbolje rešenje - Gen_{start};
- Prosečan broj generacija potreban da EA-BnB završi sa radom - Gen_{kraj};
- Prosečno odstupanje od optimalnog rešenja - agap (u procentima);

Tabela 3.14: Rezultati za AP instance do 50 čvorova

Test	CPLEX 12.1		EA-BnB					
inst.	Rešenje	$t(s)$	Reš.	$t(s)$	$t_{kraj}(s)$	Gen_{start}	Gen_{kraj}	agap
10LL	224250.0548	0.09	opt	0.007	0.085	9.8	59.8	0
10LT	250992.2617	0.09	opt	0.034	0.103	5.5	55.5	0
10TL	263399.9433	0.08	opt	0.001	0.097	6.2	56.2	0
10TT	263399.9433	0.07	opt	0.012	0.076	4.8	54.8	0
20LL	234690.9629	0.18	opt	0.004	0.146	6.6	56.6	0
20LT	253517.3953	0.86	opt	0.268	0.526	7.7	57.6	0
20TL	271128.1758	0.28	opt	0.006	0.170	11.9	61.9	0
20TT	296035.4020	0.74	opt	0.210	0.409	12.3	62.3	0
25LL	238977.9516	1.27	opt	0.013	0.158	11.7	61.7	0
25LT	276372.4982	2.21	opt	0.918	1.370	21.9	71.8	0.196
25TL	310317.6372	0.65	opt	0.009	0.155	14.6	64.5	0
25TT	348369.1478	1.45	opt	0.244	0.613	24.7	74.7	0.284
40LL	241955.7064	4.96	opt	0.190	0.426	13.4	63.5	0
40LT	272218.3248	6.96	opt	1.098	1.770	17.2	67.2	0
40TL	298919.0110	4.68	opt	0.020	0.214	17.7	67.7	0
40TT	354874.1000	15.89	opt	0.453	1.117	28.7	78.7	0
50LL	238520.5865	9.71	opt	0.020	0.256	19.4	69.5	0
50LT	272897.4853	24.02	opt	0.991	1.795	17.3	67.3	0.527
50TL	319015.7711	21.87	opt	0.033	0.235	18.5	68.5	0
50TT	417440.9925	1068.3	opt	0.912	1.835	42.1	92.2	0.33
prosek	-	58.22	opt	0.272	0.578	15.6	65.6	0.067

Iz rezultata na instancama do 50 čvorova se vidi da je hibridni algoritam pronašao sva optimalna rešenja za manje od 1 sekunde, što ukazuje na izuzetnu efikasnost

Tabela 3.15: Rezultati za AP instance sa 100, 200 ili 300 čvorova

Test	Najbolje	EA-BnB						
inst.	rešenje	Rešenje	$t(s)$	$t_{kraj}(s)$	Gen_{start}	Gen_{kraj}	agap	Komentar
100LL	246713.97	*naj*	0.363	1.221	33.5	83.5	0.961	najbolje
100LT	256155.33	*naj*	1.713	4.041	38.3	88.3	0.648	najbolje
100TL	362950.09	*naj*	0.231	0.881	37.6	87.6	0.096	najbolje
100TT	474186.73	*naj*	2.586	5.039	62.6	112.7	0.197	poboljšanje
200LL	**231069.50**	**241992.97**	2.818	6.755	52.3	102.3	0.771	**nejasno**
200LT	268820.57	267236.99	49.651	72.923	98.3	142.6	1.422	poboljšanje
200TL	273443.81	*naj*	1.186	5.812	57	107.1	0.447	najbolje
200TT	290841.84	290743.23	12.498	25.333	74	124	1.016	poboljšanje
300LL	-	227667.00	5.875	17.565	56.1	106.1	1.309	novo rešenje
300TT	-	396177.01	86.579	146.105	89	138.1	0.783	novo rešenje
prosek	-	-	16.35	28.57	59.87	109.23	0.765	

implementacije algoritma. Sa druge strane, univerzalni rešavač je najtežu instancu, 50TT izračunavao preko 1000 sekundi.

Za veće instance sa 100, 200 i 300 čvorova, optimalna rešenja u trenutku testiranja nisu bila poznata. Na tim instancama, CPLEX 12.1 rešavač prekida sa radom zbog memorijskog ograničenja. U tabeli 3.15 su, zbog toga, date vrednosti najboljih poznatih rešenja iz literature, u trenutku testiranja. Te vrednosti su preuzete iz rada [16]. U koloni EA-BnB oznaka *naj* označava da je EA-BnB našao rešenje iste vrednosti kao najbolje poznato. Na 3 test instance sa 100 i 200 čvorova i „lakim“ ograničenjima kapaciteta, EA-BnB je pronašao najbolja poznata rešenja. Na 3 test instance sa „teškim“ ograničenjem kapaciteta, EA-BnB je našao rešenja bolja od do tada najboljih poznatih. Na kraju, rešenje koje je EA-BnB pronašao za instancu 200LL, je 241992.97 i ono ima veću vrednost od rešenja 231069.50 koje je prikazano u radu [16]. Rešenje 241992.97 je takođe prikazano u radovima [30] i [61]. U poglavlju 4 će biti dokazano da je to rešenje ujedno i optimalno, što znači da je u radu [16] greškom objavljeno rešenje sa vrednošću 231069.50.

Poglavlje 4

Egzaktna metoda dekompozicije

U ovom poglavlju biće predstavljena dva algoritma koja se zasnivaju na dekompoziciji matematičkog modela i koja koriste univerzalni rešavač za rešavanje problema manjih dimenzija. Prvi je iterativni algoritam kojim se efikasno pronalaze dopustiva rešenja problema. Drugi, kombinatorni algoritam polazi od nekog dopustivog rešenja, a zatim pokušava da nađe bolje rešenje i, konačno, dokazuje optimalnost rešenja.

Nakon definisanja klase problema na koju se algoritmi mogu primeniti i predstavljanja samih algoritama, biće prikazana implementacija oba algoritma za rešavanje problema USAHLP i CSAHLP, kao i rezultati dobijeni njihovom primenom.

4.1 Polazne pretpostavke

Matematički model na koji se primenjuje metoda dekompozicije mora ispunjavati određene uslove. Neka je matematički model zadat kao problem mešovitog celobrojnog linearnog programiranja $M = M(V, cilj, C)$, gde je V skup promenljivih, $cilj = cilj(V)$ funkcija cilja i $C = C(V)$ skup uslova. Polazne pretpostavke su:

- cilj optimizacije je minimizovati vrednost funkcije cilja;
- $V = X \cup Y \cup Z$, gde su $Z = (z_1, ..., z_n)$ binarne promenljive, a za promenljive $X = (x_1, ..., x_l)$ i $Y = (y_1, ..., y_m)$ ne postoji ograničenje tipa podataka (mogu biti realne, celobrojne ili binarne);
- funkcija cilja je oblika $cilj = A * X + B * Y + F * Z$, gde je $A = (a_1, ..., a_l)$, $B = (b_1, ..., b_m)$, $F = (f_1, ..., f_n)$ i A je takvo da je $A * X >= 0$ za svako X

koje je dopustivo u smislu uslova C.

Stavljajući ove pretpostavke u kontekst hab lokacijskih problema, binarne promenljive Z mogu da odgovaraju činjenicama da li su habovi uspostavljeni na odgovarajućim lokacijama ili ne, dok promenljive F mogu predstavljati fiksne troškove uspostavljanja habova, koji su često dominantan sabirak u funkciji cilja.

Ako je $S(P)$ prostor rešenja nekog MILP problema P i ako je $r \in S(P)$ neko dopustivo rešenje P, što znači: uređeni skup vrednosti svih promenljivih problema P koje se slažu sa svim uslovima tog problema, tada će funkcija evaluacije $val = val(cilj, r)$ biti korišćena da označi vrednost funkcije cilja datog rešenja r.

4.2 Dekompozicija modela

Prvi korak u dekompoziciji modela jeste pravljenje nadproblema N za MILP problem $M = M(V, cilj, C)$. Nadproblem N je problem koji je opštiji od M, ima manje promenljivih i uslova i čijim rešavanjem se može ograničiti pretraživanje prostora rešenja problema M. Zatim se definiše potproblem za fiksirane vrednosti nekog podskupa promenljivih problema M i njegovim rešavanjem se dobija dopustivo rešenje problema M. Potproblem je, takođe, manje kompleksnosti od celog problema. Nakon toga, u iterativnom algoritmu i nadproblem i potproblem se rešavaju više puta, čime se dobijaju rešenja koja postepeno konvergiraju ka optimalnom rešenju problema M.

Nadproblem N

Polazeći od $M(V, cilj, C)$, koji zadovoljava prethodno navedene uslove, definiše se nadproblem $N = N(Y \cup Z, cilj_{YZ}, C\mid_{Y \cup Z})$. Pri tome je $cilj_{YZ} = cilj(Y \cup Z) = B * Y + F * Z$, a $C\mid_{Y \cup Z} \subseteq C$ restrikcija skupa uslova C koja sadrži samo one uslove u kojima se koriste promenljive Y i Z, ali ne i promenljive X. Drugim rečima, skup promenljivih X je eliminisan iz problema zajedno sa svim uslovima u kojima se promenljive X koriste, a funkcija cilja je promenjena tako što je izbačen sabirak $A * X \geq 0$.

Narednim tvrđenjima uspostavlja se teorijska osnova algoritma proste dekompozicije modela.

Tvrđenje 1. *Ako je r dopustivo rešenje problema M, tada je restrikcija r na promenljive Y i Z, u oznaci $r\mid_{Y\cup Z}$, dopustivo rešenje nadproblema N.* $r \in S(M) \Rightarrow r\mid_{Y\cup Z} \in S(N)$.

Dokaz. Ako je $r \in S(M)$, tada se r slaže sa svim uslovima C, što znači da se slaže i sa uslovima $C\mid_{Y\cup Z} \subseteq C$. Dakle, restrikcija na promenljive $Y \cup Z$ je dopustivo rešenje nadproblema N, $r\mid_{Y\cup Z} \in S(N)$. □

Tvrđenje 2. $r \in S(M) \Rightarrow val(cilj_{YZ}, r\mid_{Y\cup Z}) \leq val(cilj, r)$.

Dokaz. Iz tvrđenja 1 $\Rightarrow$ $r\mid_{Y\cup Z} \in S(N)$. Kako je $cilj = A * X + B * Y + F * Z$ i $cilj_{YZ} = B * Y + F * Z$ i $A * X >= 0$, tada $val(cilj, r) - val(cilj_{YZ}, r\mid_{Y\cup Z}) = val(A * X, r\mid_X) \geq 0 \Rightarrow val(cilj, r) \geq val(cilj_{YZ}, r\mid_{Y\cup Z})$. □

Tvrđenje 3. *Ako je M problem, N nadproblem problema M i o je realan broj, tada, ako ne postoji rešenje $r' \in S(N)$ takvo da $val(cilj_{YZ}, r') \leq o$, onda ne postoji ni rešenje $r \in S(M)$ takvo da $val(cilj, r) \leq o$.*

Dokaz. Pretpostavimo suprotno: neka je $r \in S(M)$ rešenje takvo da $val(cilj, r) \leq o$. Iz tvrđenja 1 sledi da $r\mid_{Y\cup Z} \in S(N)$, a iz tvrđenja 2 sledi da $val(cilj_{YZ}, r\mid_{Y\cup Z}) \leq val(cilj, r) \leq o$. Dakle, $r\mid_{Y\cup Z}$ je rešenje nadproblema N takvo da $val(cilj_{YZ}, r\mid_{Y\cup Z}) \leq o$, što je suprotno uslovu da ne postoji rešenje nadproblema N sa vrednošću funkcije cilja manjom od o. □

Prethodna tvrđenja važe za svaki MILP problem M u kome je $A * X \geq 0$. Nadproblem N se može definisati na više načina, jer skup X može biti izabran na različite načine. U daljem će biti podrazumevano da je skup promenljivih X određen i uvek isti.

Potproblem $M(H)$

Neka su vrednosti promenljivih iz skupa Z problema M unapred određene i fiksirane: dat je skup $H \subseteq Z$ takav da je $z_i = 1, i \in H$ i $z_i = 0, i \in Z \setminus H$. Postoje bar dva načina za pronalaženje dopustivog rešenja problema M koje sadrži tako zadate vrednosti promenljivih iz skupa Z:

1. Implementirati ceo MILP problem M, a zatim obaviti preprocesiranje kojim se, za dato H, brišu neke promenljive i uslovi. Dakle, potrebno je napraviti skup

matematičkih pravila po kojima je moguće izvršiti eliminaciju promenljivih i uslova. Neki primeri takvih pravila dati su u poglavlju 2.4. Ovaj pristup se može primeniti na svaki MILP problem. Algoritam preprocesiranja se direktno implementira kada je određen skup pravila za eliminaciju, ali su vreme izvršavanja tog algoritma i potrebna memorija proporcionalni broju promenljivih i uslova problema M.

2. Napraviti nov matematički model i odgovarajući MILP problem kada je dat skup H. Ovo je eksplicitan pristup u kome se kreiraju nove matematičke formule u kojima su neke promenljive polaznog problema pretvorene u konstante i, posledično, trivijalni uslovi su izbačeni. Ukoliko su promenljive iz skupova X i Y jako zavisne od vrednosti promenljivih skupa Z, to će rezultirati uklanjanjem velikog broja promenljivih i uslova. Tako se dobija potproblem polaznog problema i dobijeni potproblem se označava sa $M(H)$. Formule MILP potproblema $M(H)$ koje važe za svako H se mogu definisati, ali se potproblem mora implementirati pojedinačno, za svako konkretno H. Kada se potproblem $M(H)$ implementira, prednost je što ni u jednom trenutku on neće sadržati trivijalne uslove i fiksirane promenljive. To znači da će manje memorije biti potrebno za njegovu implementaciju i faza preprocesiranja iz prvog pristupa neće biti potrebna.

Ponekad je korisno napraviti i nadproblem potproblema, u oznaci $N(H)$.

4.3 Algoritam proste dekompozicije modela (APDM)

Ako je $|X| \gg |Y|$ i $|X| \gg |Z|$ tada je dimenzija nadproblema N značajno manja od dimenzije problema M, jer su promenljive X i odgovarajući uslovi izbačeni, te stoga rešavanje N može biti moguće i kada rešavanje M nije moguće, zbog memorijskog ili vremenskog ograničenja.

Iterativni algoritam proste dekompozicije modela (APDM - algoritam 12) koristi nadproblem N i potprobleme $M(H)$ za nalaženje dopustivih rešenja problema M sa dobrim vrednostima funkcije cilja.

U svakoj iteraciji, rešava se nadproblem N i na taj način se pronalaze konfiguracije $G \subseteq Z$ koje mogu voditi ka dobrom rešenju problema M. Zatim se traži

Algoritam 12: APDM

ulaz : Problem M
izlaz: Dopustivo rešenje problema M
$najbolje = \infty$;
$resenja = \emptyset$;
$N = Nadproblem(M)$;
while *not KrajRada()* **do**
  $ax =$ izračunaj donju granicu $A * X$ (ili postavi $ax = 0$);
  $r = Resenje(N, najbolje - ax)$;
  if $r == \emptyset$ **then**
    break;
  end
  /*G `je restrikcija rešenja` r `na promenljive` $z = 1, z \in Z$ */
  $G = \{z \in Z | \{z = 1\} \in r\}$;
  foreach $H, H \subseteq G$ **do**
    $novo = Resenje(M(H), najbolje)$;
    if $novo \neq \emptyset$ **then**
      $najbolje = Vrednost(novo)$;
      /*`dodaj` H `na listu` $resenja$ */
      $resenja \leftarrow H$;
    end
  end
  /*`u nadproblem` N `dodaj uslov koji sprečava da se` G `ponovi` */
  $N \leftarrow \{ \sum_{z \in Z \setminus G} z >= 1 \}$
end
foreach $H \in resenja$ **do**
  $novo = Optimizuj(M(H), najbolje)$;
  if $novo \neq \emptyset$ **then**
    $najbolje = Vrednost(novo)$;
  end
end

dopustivo rešenje potproblema $M(H), \forall H \subseteq G$. Zbog toga je poželjno da važi $|G| \ll |Z|$. Pomoćna funkcija $Resenje(P, vrednost)$ poziva univerzalni rešavač za nalaženje dopustivog rešenja nekog MILP problema P (u algoritmu APDM P je ili N ili $M(H)$) i pri tom postavlja *vrednost* kao gornju granicu potencijalnog rešenja. Važno je naglasiti da se univerzalni rešavač poziva tako da nađe bilo koje dopustivo rešenje, koje ne mora obavezno biti optimalno. Ukoliko postoji rešenje potproblema $M(H)$ sa vrednošću funkcije cilja boljom od *najbolje* vrednosti, skup H se stavlja na listu rešenja sa dobrim vrednostima funkcije cilja, a vrednost najboljeg rešenja se ažurira.

Razlika između vrednosti funkcije cilja problema M i nadproblema N je $A * X$. Ukoliko je na neki način moguće odrediti donju granicu ax sabirka $A * X$, onda je prilikom rešavanja N korisno oduzeti tu vrednost od gornje granice i tako ubrzati konvergenciju; ukoliko to nije moguće uraditi na efikasan način, postavlja se $ax = 0$. Vrednost ax se ne mora računati u svakoj iteraciji, već se to može uraditi samo jednom, pre ulaska u *while* petlju.

Na kraju svake iteracije, nadproblemu N se dodaje uslov $\{ \sum_{z \in Z \setminus G} z >= 1 \}$. Zahvaljujući tom uslovu, prilikom rešavanja N u narednoj iteraciji biće dobijen skup G koji je različit od svih dobijenih G u prethodnim iteracijama, zato što ovaj uslov zahteva da bar jedna od binarnih promenljivih skupa Z koje nisu iz G bude jednaka 1. Uvođenje ovog uslova je opravdano time što se u svakoj iteraciji rešavaju potproblemi $M(H), \forall H \subseteq G$, odnosno, ovim uslovom nijedno $H \subseteq Z$, koje već nije analizirano, neće biti isključeno kao potencijalno rešenje.

Iako je G različito u svakoj iteraciji, podskupovi $H \subseteq G$ mogu biti ponovljeni, pa treba implementirati i pomoćnu strukturu podataka koja pamti koje su konfiguracije G analizirane u prethodnim iteracijama, te ako je H podskup bilo koje prethodne konfiguracije G, nema potrebe ponovo rešavati $M(H)$.

Ako rešenje nadproblema N sa vrednošću funkcije cilja boljom od *najbolje* vrednosti ne postoji, iz tvrđenja 3 sledi da ne postoji ni takvo rešenje problema M i u tom slučaju se prekida sa iteracijama. To je jedan od kriterijuma zaustavljanja, a mogu postojati i drugi, kao što su: vremensko ograničenje, ograničenje broja iteracija ili neko drugo.

Kada se iteracije završe, za svaku konfiguraciju H, koja se nalaze na listi *resenja*, ponovo se rešava potproblem $M(H)$, sa gornjom granicom *najbolje* vrednosti.

Ovoga puta, univerzalni rešavač se poziva tako da nađe optimalno rešenje potproblema $M(H)$ (bolje od $najbolje$ vrednosti), ili se odredi neki kriterijum zaustavljanja. Svaki put kada se nađe novo rešenje, promenljiva $najbolje$ se ažurira. Ponovo rešavanje svih $resenja$ je neophodno jer je prilikom pozivanja $Resenje(M(H), najbolje)$ traženo bilo koje rešenje. Zbog toga, svako rešenje na listi $resenja$ i dalje može imati vrednost funkcije cilja bolju od $najbolje$ vrednosti.

Prilikom pozivanja $Resenje(N, najbolje)$, može se tražiti ili LP ili MILP rešenje, u zavisnosti od implementacije i težine rešavanja nadproblema N. Ukoliko se traži LP rešenje, prilikom određivanja skupa G uzimaju se sve promenljive $z \in Z$, takve da je $z > \epsilon > 0$. Prilikom pozivanja $Resenje(M(H), najbolje)$, uvek se traži dopustivo MILP rešenje, koje, ako postoji, ujedno predstavlja i dopustivo rešenje problema M.

Algoritam 12 je moguće primeniti na bilo koji MILP problem za koji je moguće definisati nadproblem i potprobleme i za koji važi da je funkcija cilja aditivna po promenljivama koje su uklonjene iz MILP problema da bi se dobio nadproblem ($A * X \geq 0$). Na prikazan način se dobija kvalitetno rešenje polaznog MILP problema M, a da pri tom ceo MILP problem nije rešavan, već problemi mnogo manjih dimenzija. Rešavanje problema manje dimenzije više puta je često brže od rešavanja celog problema; osim toga, često je moguće rešavati nadproblem i potprobleme iako to nije moguće uraditi sa celim MILP problemom.

Ukoliko se iteracije nastavljaju sve dok postoji rešenje nadproblema N i ako se nakon iteracija, u završnom rešavanju konfiguracija na listi potencijalnih $resenja$ uvek traži optimalno rešenje, ovim algoritmom će biti pronađeno optimalno rešenje i biće dokazana njegova optimalnost. Međutim, za probleme velikih dimenzija ne očekuje se da do toga dođe, već je neophodno u nekom trenutku prekinuti sa iteracijama. Ideja algoritma APDM i jeste da se on koristi kao metaheuristika; shodno tome, u sam algoritam moguće je uvesti dodatna ograničenja, kako bi se on brže izvršavao. Na primer, ukoliko $|G|$ nije mali broj, ne moraju se rešavati potproblemi $M(H)$ za svako $H \subseteq G$, već samo za neke od tih podskupova, recimo, takve da je $|G| - |H| \leq 3$. Dalje, ako nije moguće tačno odrediti donju granicu ax vrednosti $A * X$, moguće je, poznavanjem konkretnog problema, proceniti vrednost ax i koristiti je za ubrzanje konvergencije. Može se ceo algoritam APDM pozivati više puta, s tim da se ax prosleđuje kao parametar u pozivu APDM; u prvom pozivu se koristi velika vrednost ax, da bi se brzo dobilo neko rešenje, a zatim u narednim pozivima

APDM se svaki put koristi sve manja vrednost ax.

Konačno, ako je na opisan način moguće definisati i nadproblem nadproblema i odgovarajuće potprobleme nadproblema, prilikom poziva $Resenje(N, najbolje)$ može se rekurzivno pozvati sam algoritam APDM, sa obaveznim izlaskom iz rekurzije, koji se realizuje pozivanjem univerzalnog rešavača, u trenutku kada nadproblemi više nisu definisani. Time bi se dobilo rešenje polaznog problema M putem rešavanja drugih problema vrlo malih dimenzija.

4.4 Kombinatorni algoritam dekompozicije modela (KADM)

Ideja kombinatornog algoritma dekompozicije modela (KADM) jeste da se sabirak $A * X$ u funkciji cilja ne računa, da se zatim odredi donja granica sabirka $B * Y$ i da se analiziraju sve konfiguracije $H \subseteq Z$ i za njih u polinomskom vremenu odrede neke donje granice cele funkcije cilja. Ako je za neko H ta donja granica manja od date gornje granice, rešava se potproblem $M(H)$.

Pre izlaganja algoritma, biće izvršena dalja dekompozicija modela.

4.4.1 Particionisanje modela

Za svako $k = 0, ..., n$, gde je $n = |Z|$, definiše se $M_k = M_k(V, cilj, C_k)$, gde je $C_k = C \cup \{\sum Z = k\}$. MILP problem M_k je nastao od problema M dodavanjem uslova da je suma vrednosti binarnih promenljivih Z jednaka k. Slično, definiše se problem $M_k^+ = M_k^+(V, cilj, C_k^+)$, gde je $C_k^+ = C \cup \{\sum Z \geq k\}$. Ovi problemi se zovu particijama polaznog problema M.

Tvrđenje 4. *Optimalno rešenje MILP problema M jednako je minimumu svih optimalnih rešenja problema M_k. $opt(M) = \min_k opt(M_k)$*

Tvrđenje 5. *Optimalno rešenje problema M jednako je minimumu svih optimalnih rešenja prvih l particija $M_k, k = 0, ..., l-1$ i rešenja M_l^+. $opt(M) = \min(opt(M_0), .., opt(M_{l-1}), opt(M_l^+))$.*

Dokaz. S obzirom da su promenljive Z binarne i ima ih n, optimalno rešenje $opt(M)$ zadovoljava tačno jedan od uslova $\{\sum Z = k\}$ ($\{\sum Z \geq k\}$), za neko $k = 0, ..., n$,

što znači da će isto rešenje biti dobijeno ili za M_k, ako je $k < l$ ili za M_k^+, ako je $k \geq l$. □

Tvrđenja iz poglavlja 4.2 takođe važe i za nadproblem N_k problema M_k i za nadproblem N_k^+ problema M_k^+, koji se dobijaju na isti način kao nadproblem N, tako što se uklone sve promenljive X i odgovarajući uslovi. Ako je M_k particija problema M a N je nadproblem problema M, onda je N_k nadproblem problema M_k. Obrnuto takođe važi. Drugim rečima, particija nadproblema je isto što i nadproblem particije, jer nije bitno da li se prvo uklone promenljive X ili se doda uslov da je suma svih promenljivih skupa Z jednaka k.

Teorema 2. *Ako je neko dopustivo rešenje r MILP problema M poznato, sa vrednošću funkcije cilja o i ako je N odgovarajući nadproblem, a M_k, M_k^+ i N_k, N_k^+ odgovarajuće particije, tada iz tvrđenja 2 sledi da:*

- *Ako N_k nema dopustivo rešenje sa vrednošću funkcije cilja manjom od o, onda ne postoji ni rešenje M_k sa vrednošću funkcije cilja manjom od o.*
- *Ako N_k^+ nema dopustivo rešenje sa vrednošću funkcije cilja manjom od o, onda ne postoji ni rešenje M_k^+ sa vrednošću funkcije cilja manjom od o.*

Teorema 2 pokazuje da se rešavanjem N_k ili N_k^+ može smanjiti prostor rešenja MILP problema M koji je potrebno pretraživati. Ako postoji dopustivo rešenje $r \in S(M)$ sa vrednošću funkcije cilja o, i ako je optimalno rešenje N_k (ili N_k^+) lošije od o, onda optimalno rešenje problema M ne treba tražiti u particijama M_k (odnosno M_k^+). Može se desiti da je već i LP rešenje N_k lošije od o, pa se i u tom slučaju particija M_k može ignorisati.

Fragment nadproblema

Fragment nadproblema je MILP problem, u oznaci $N' = N'(granica)$, koji se definiše za dati problem $M = M(V, cilj, C)$, odgovarajući nadproblem $N = N(Y \cup Z, cilj_{YZ}, C \mid_{Y \cup Z})$ i neku realnu vrednost označenu sa $granica$, tako što je $N'(granica) = N'(Y \cup Z, cilj_Y, C')$, pri čemu je $C' = C \mid_{Y \cup Z} \cup \{B * Y + F * Z \leq granica\}$ i $cilj_Y = B * Y$. Dakle, razlika između N i N' je što je funkcija cilja nadproblema N pretvorena u uslov u N' sa gornjom granicom postavljenom na neku vrednost $granica$ i što funkcija cilja u N' sadrži samo sabirak $B * Y$.

Za particiju N_k definiše se fragment $N_k'(granica)$ na isti način.

Tvrđenje 6. *Neka su k i granica date vrednosti i neka optimalno rešenje fragmenta $N'_k(granica)$ postoji, i označeno je sa opt i ima vrednost funkcije cilja $v_{opt} = val(cilj_Y, opt)$. Neka je $H \subseteq Z, |H| = k$ bilo koji podskup. Ako postoji dopustivo rešenje r particije nadproblema N_k, $r \in S(N_k)$, takvo da $\forall h \in H, r_h = 1$, sa vrednošću funkcije cilja $v = val(cilj_{YZ}, r)$ i ako je $v \leq granica$, tada $\sum_{h \in H} F_h + v_{opt} \leq v$.*

Dokaz. Ako je $v = val(cilj_{YZ}, r) \leq granica$ onda je $r \in S(N_k)$ ujedno i dopustivo rešenje problema $N'_k(granica)$. Vrednost $v' = val(cilj_Y, r)$ je $v' = val(cilj_{YZ}, r) - \sum_{h \in H} F_h = v - \sum_{h \in H} F_h$. Međutim, kako je v_{opt} vrednost optimalnog rešenja problema $N'_k(granica)$ sledi da je $v_{opt} \leq v'$, što znači da $v_{opt} \leq v - \sum_{h \in H} F_h$. □

Tvrđenjem 6 se formalno konstatuje sledeća činjenica: ako optimalno rešenje problema $N'_k(granica)$ postoji i ima vrednost v_{opt}, onda ako za bilo koju konfiguraciju $H \subseteq Z, |H| = k$ postoji rešenje problema N_k koje sadrži tu konfiguraciju, ono je ili lošije od vrednosti *granica* ili lošije od vrednosti $v_{opt} + \sum_{h \in H} F_h$.

Drugim rečima, fragment nadproblema određuje donju granicu sabirka $B * Y$ funkcije cilja, bez obzira na vrednost promenljivih iz skupa Z.

Parcijalni fragment nadproblema

Za fragment nadproblema $N'_k(granica)$ i za $Z_0 \subseteq Z$ i $Z_1 \subseteq Z$, definiše se parcijalni fragment nadproblema $N'_k(granica, Z_0, Z_1)$ tako što se fiksiraju vrednosti promenljivih iz skupa Z: $\forall z \in Z_0$ fiksira se $z = 0$ i $\forall z \in Z_1$ fiksira se $z = 1$. Očigledno mora važiti $Z_0 \cap Z_1 = \emptyset$ i $|Z_1| \leq k$, inače ovako definisan parcijalni fragment ne bi imao rešenja.

Treba primetiti da važi $N'_k(granica) \equiv N'_k(granica, \emptyset, \emptyset)$.

Tvrđenje 7. *Neka je $Z_0 \subseteq Z'_0 \subseteq Z$, $Z_1 \subseteq Z'_1 \subseteq Z$ i neka je za datu vrednost granica, opt optimalno rešenje problema $N'_k(granica, Z_0, Z_1)$ koje ima vrednost v_{opt}. Ako je r dopustivo rešenje problema $N'_k(granica, Z'_0, Z'_1)$ sa vrednošću v_r onda je $v_{opt} \leq v_r$.*

Dokaz. Pretpostavimo suprotno: neka je $v_r < v_{opt}$. Međutim, r je rešenje problema $N'_k(granica, Z'_0, Z'_1)$ i $Z_0 \subseteq Z'_0$ i $Z_1 \subseteq Z'_1$, što znači da je r takođe i rešenje problema $N'_k(granica, Z_0, Z_1)$ sa vrednošću $v_r < v_{opt}$, što je suprotno uslovu tvrđenja da je *opt* optimalno rešenje problema $N'_k(granica, Z_0, Z_1)$. □

Parcijalni fragment nadproblema određuje donju granicu $B * Y$ sabirka funkcije cilja, pri čemu su neke promenljive iz skupa Z fiksirane na 1, a neke na 0. Pomoću njega se određuje jedna donja granica cele funkcije cilja: ako je v optimalno rešenje za $N'_k(granica, Z_0, Z_1)$ i H je takvo da $Z_1 \subseteq H \subseteq Z$ i $H \cap Z_0 = \emptyset$, onda je $v + \sum_{h \in H} F_h$ jedna donja granica vrednosti funkcije cilja nadproblema (a samim tim i celog problema), za dato H.

Udeo promenljivih iz skupa X u funkciji cilja se može zanemariti (što je opravdano uslovom $A * X \geq 0$); donja granica udela promenljivih iz skupa Y se određuje rešavanjem fragmenta nadproblema; za dato $H \subseteq Z$, $F * H$ se lako računa. Time je, za dato H, određena jedna donja granica funkcije cilja problema M.

4.4.2 Pregled korišćenih oznaka

U algoritmima koji slede koriste se sledeće oznake koje su prethodno definisane:

- M - MILP problem za koji se traži minimum vrednosti $A * X + B * Y + F * Z$, Z su binarne promenljive i $A * X \geq 0$;
- M_k - particija, odnosno problem M sa dodatim uslovom $\sum Z = k$;
- M_k^+ - particija, odnosno problem M sa dodatim uslovom $\sum Z \geq k$;
- N - nadproblem M u kome su X promenljive i odgovarajući uslovi izbačeni;
- N_k, N_k^+ - particije nadproblema (ili, ekvivalentno, nadproblemi particija);
- $N'(granica), N'_k(granica)$ - fragment nadproblema sa novom funkcijom cilja $cilj_Y = B * Y$ u kojima je funkcija cilja nadproblema pretvorena u uslov $B * Y + F * Z \leq granica$;
- $N'_k(granica, Z_0, Z_1)$ - donja granica vrednosti sabirka $B * Y$ kada su vrednosti promenljivih $Z_0 \subseteq Z$ postavljene na 0 i $Z_1 \subseteq Z$ postavljene na 1.
- $M(H)$ - potproblem problema M u kome su promenljive $H \subseteq Z$ postavljene na 1 i promenljive $Z \setminus H$ postavljene na 0.
- ax - donja granica $A * X$ sabirka, ako je moguće odrediti je; ako nije, onda $ax = 0$.

4.4.3 Algoritam za rešavanje jedne particije

Sledećim tvrđenjem se konstatuje činjenica da je optimalno rešenje particije M_k jednako minimumu optimalnih rešenja svih $\binom{n}{k}$ potproblema $M(H)$.

Tvrđenje 8. *Optimalno rešenje particije M_k jednako je najmanjem optimalnom rešenju svih potproblema $M(H)$, gde su H podskupovi sa k elemenata skupa Z.* $opt(M_k) = \min_{H \subseteq Z, |H|=k} opt(M(H))$.

Međutim, kada je poznato neko dopustivo rešenje, nije neophodno rešavati svih $\binom{n}{k}$ potproblema. Umesto toga, implementira se niz algoritama koji u polinomskom vremenu izvršavanja određuju neke donje granice vrednosti funkcije cilja. Taj niz algoritama će u daljem tekstu biti označen kao „sito“. Potproblem $M(H)$ se rešava samo za one konfiguracije H koje prođu kroz sito.

Algoritam 13 prikazuje rutinu $Particija$ za rešavanje jedne particije M_k. Ova rutina je jedna specifična primena metode grananja i ograničavanja. U osnovi je rekurzivni kombinatorni algoritam za generisanje svih podskupova sa k od n elemenata. Polazi se od praznog skupa i zatim se u svakom rekurzivnom koraku dodaje jedan element, sa indeksom većim od poslednjeg dodatog elementa. U rutini se koristi globalna promenljiva $najbolje$ koja sadrži vrednost najboljeg poznatog rešenja i koju algoritam ažurira.

Rutina $Particija$ ima 4 ulazna parametra i poziva se sa $Particija(\emptyset, k, 1, -\infty)$. Vrednosti parametara u pozivu rutine su, redom: skup H, u koji se dodaju elementi, je na početku prazan, $H = \emptyset$; u H treba dodati k elemenata, a to je ujedno i dubina rekurzije; prvi sledeći element koji treba dodati ima indeks 1, odnosno na početku to je z_1; donja granica parcijalnog fragmenta nadproblema je $-\infty$ (ili neka druga donja granica ako je poznata).

Prvo se proverava da li trenutna konfiguracija H prolazi kroz sito ili ne. Neka je $fragment$ vrednost ili optimalnog rešenja ili LP rešenja $N'_k(resenje, Z_0, Z_1)$, za neko $Z_1 \subseteq H$ i $Z_0 \cap H = \emptyset$. Ako je ta vrednost poznata, pomoću nje se definiše sledeće sito: ako je $fragment + ax + \sum_{h \in H} F_h \geq resenje$ onda konfiguracija H može biti ignorisana i algoritam se vraća unazad. Tvrđenja 6 i 7 to opravdavaju, jer bolje rešenje od vrednosti $resenje$ ne postoji za takvo H. Takođe, ako je $jos > 0$, onda će još jos elemenata biti dodato u skup H, što znači da će se vrednost funkcije cilja povećati za bar jos najmanjih vrednosti F_h za $h \geq next$. Dodajući te vrednosti na

Algoritam 13: Rekurzivna rutina *Particija* za rešavanje jedne particije M_k

ulaz : H - tekući podskup $H \subseteq Z$
ulaz : jos - broj elemenata koje još treba dodati u H
ulaz : $sled$ - indeks prvog sledećeg elementa z_{sled} koji se dodaje u H
ulaz : $fragment$ - donja granica parcijalnog fragmenta nadproblema
if $Sito(H, jos, sled, fragment)$ **then** return ;
if $jos == 0$ **then**
    osveži brojače;
    $novo = Resenje(M(H), resenje)$;
    if *(novo* $< resenje$*)* **then** $resenje = novo$;
    return;
end
if *uslov za računanje fragmenta* **then**
    $Z_0 = \{z_i \in Z \setminus H | i < sled\}$;
    $fragment = Resenje(N'_k(resenje - ax, Z_0, H), \infty)$;
    if $Sito(H, jos, sled, fragment)$ **then** return ;
end
$posl = (|Z| - more + 1)$;
while $sled \leq posl$ **do**
    $Particija(H \cup z_{sled}, jos - 1, sled + 1, fragment)$;
    $sled++$;
end

prethodnu sumu, dobija se bolje sito.

Druga sita, sa boljom vrednošću donje granice, se definišu u zavisnosti od problema M i potproblema $M(H)$.

Ako H prođe kroz sito, algoritam proverava da li je već k elemenata dodato u skup H i ako jeste, rešava se $M(H)$. Vrednost $resenje$ se postavlja kao gornja granica pri rešavanju $M(H)$, dakle traži se samo rešenje koje je bolje od trenutno najboljeg. Ako postoji, vrednost $resenje$ se ažurira. U ovom koraku se menjaju vrednosti određenih brojača, što će biti objašnjeno u poglavlju 4.4.4.

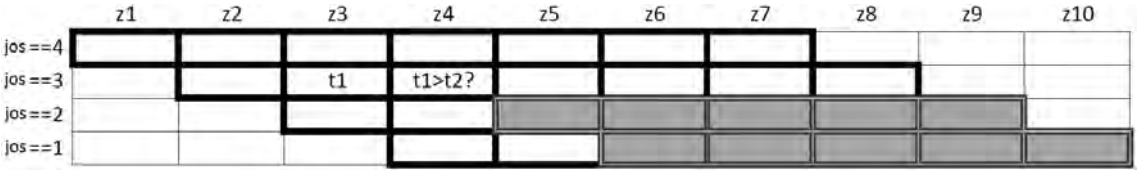
Računanje vrednosti $fragment = Resenje(N'_k(resenje - ax, Z_0, H), \infty)$ u svakom koraku rekurzije bi bilo previše vremenski zahtevno. Međutim, tvrđenje 7 nam omogućava da za sito $fragment + ax + \sum_{h \in H} F_h \geq resenje$ koristimo vrednost fragmenta koja je izračunata u nekom od koraka koji su direktno prethodili trenutnom koraku, kao i da vrednost izračunatu u trenutnom koraku, koristimo u svim koracima koji direktno slede. Zbog toga se vrednost fragmenta računa u zavisnosti od nekog kriterijuma.

Na kraju rutine $Particija$, ona se rekurzivno poziva sa dodavanjem jednog novog elementa u H, počev od elementa z_{sled} i završavajući sa z_{posl}.

Pri implementaciji rutine $Particija$ korišćen je sledeći nedeterministički kriterijum pri donošenju odluke da li računati fragment u nekom koraku ili ne. U svakom koraku rekurzije meri se vreme t_1 koje je potrebno da se završi obrada celog podstabla, počev od trenutnog koraka. Zatim, meri se prosečno vreme t_2 potrebno za rešavanje fragmenta. S obzirom da t_2 varira u zavisnosti od $|H|$, to prosečno vreme se računa odvojeno za svaku vrednost $|H|$. Konačno, ako je $t_1 > t_2$, gde je t_1 vreme izmereno za obradu prethodnog podstabla, onda se vrednost fragmenta računa u ovom koraku.

Korist od računanja fragmenta je dvostruka. Prvo, ako rešenje fragmenta zbog gornje granice ne postoji, celo podstablo od trenutnog koraka se ne mora pretraživati, već se pretraživanje vraća unazad. Drugo, ako rešenje fragmenta postoji, biće veće nego prethodno poznato, jer je u skupovima Z_0 i H fiksirana vrednost više promenljivih. Zbog toga će sito premašivati gornju granicu i vraćati se unazad ranije nego da je stara vrednost fragmenta korišćena. Iz tog razloga, koristi se $t_1 > t_2$ kao uslov pri odlučivanju da li računati vrednost fragmenta u nekom koraku ili ne.

Neka je $|Z| = 10$ i neka treba rešiti particiju M_4, slika 4.1. Neka je vreme



Slika 4.1: Primer pretraživanja particije M_4

potrebno da se obradi stablo od koraka $jos == 3$ i $sled == 3$ bilo t_1 i da se trenutno rutina *Particija* nalazi u koraku $jos == 3$ i $sled == 4$. Na slici je podstablo od tog koraka unapred osenčeno. Može se očekivati da će vreme potrebno za obradu tog stabla biti blisko vremenu t_1. Ako je t_1 veće od t_2 (prosečno vreme potrebno za dobijanje rešenja fragmenta), onda će nova vrednost fragmenta biti izračunata i na taj način će vreme potrebno za obradu podstabla biti smanjeno. Jasno je da vreme potrebno za obradu podstabla znatno opada sa porastom $|H|$, tako da je manje verovatno da će vrednost fragmenta biti izračunata na većim dubinama rekurzije. Kao posledica uslova $t_1 > t_2$, vrednost fragmenta se obično računa samo na prva 2 ili 3 nivoa rekurzije.

Način na koji sito funkcioniše ilustrovan je sledećim primerom: neka rutina upravo ulazi u osenčeni deo stabla ($jos == 2, sled == 5$). Neka je do tog koraka skup $H = \{z_1, z_4\}$ i neka je $fragment = Resenje(N_4'(resenje, Z_0, H), \infty)$, gde je $Z_0 = \{z_2, z_3\}$. Neka su vrednosti vektora F takve da počev od indeksa 5, F_7 i F_9 imaju najmanju vrednost. Ako je $F_1 + F_4 + F_7 + F_9 + fragment + ax \geq resenje$, onda osenčeno podstablo nema potrebe pretraživati, jer se u njemu ne može nalaziti rešenje bolje od trenutno najboljeg poznatog.

Ako se pozivi rutine *Sito* i rešavanje fragmenta ignorišu, ovaj algoritam bi testirao svih $\binom{n}{k}$ potproblema. Zbog toga, ukoliko je rutina *Sito* implementirana egzaktno, korišćenjem tačnih donjih granica, onda tvrđenje 8 pokazuje da rutina *Particija* pronalazi najbolje rešenje u particiji M_k, koje je bolje od najboljeg poznatog rešenja, ako ono postoji. Efikasnost ovog algoritma zavisi od rutine *Sito*, učestalosti rešavanja fragmenta i efikasnosti rutine za rešavanje fragmenta. Kao vrednost fragmenta mogu se koristiti i LP i MILP rešenja.

4.4.4 Struktura algoritma KADM

Kombinatorni algoritam dekompozicije modela objedinjuje izložene ideje, algoritme i tvrđenja i pronalazi optimalno rešenje problema M.

Algoritam 14: KADM

ulaz : Problem M
izlaz: Optimalno *resenje* problema M
$resenje = APDM(M)$;
foreach $k = 0..|Z|$ **do**
  $ax =$ izračunaj donju granicu $A * X$ (ili postavi $ax = 0$);
  if $Postoji(N_k, resenje - ax)$ **then**
    poređaj čvorove z_1 do z_n;
    $fragment = Resenje(N'_k(resenje - ax, \emptyset, \emptyset), \infty)$;
    $Particija(\emptyset, k, 1, fragment)$;
  end
  else if *not* $Postoji(N^+_{k+1}, resenje - ax)$ **then** break ;
end

Kao polazno rešenje uzima se rešenje dobijeno pomoću algoritma proste dekompozicije modela 12. U petlji se za svako k, počev od $k = 0$, proverava da li u particiji N_k postoji rešenje bolje od trenutno najboljeg. Rutina *Postoji* služi za to i ona koristi univerzalni rešavač. Ako postoji, računa se vrednost fragmenta nadproblema koja se prosleđuje rutini *Particija* opisanoj u prethodnom poglavlju i ona ažurira vrednost *resenje* ako nađe bolje rešenje. U algoritmu *Particija* redosled kojim se elementi skupa Z dodaju u skup H značajno utiče na efikasnost algoritma, zato što vrednost rešenja parcijalnog fragmenta zavisi od skupa H. Zbog toga je poželjno da elementi skupa Z budu poređani u nekom „dobrom“ poretku. U rutini *Particija*, svaki put kada skup $H, |H| = k$, prođe kroz sito i kada treba rešavati $M(H)$, poveća se vrednost pomoćnih brojača za svaki element $h \in H$. U narednoj iteraciji KADM, kada se rešava particija M_{k+1}, elementi skupa Z će biti poređani u opadajućem redosledu vrednosti tih brojača. Na taj način elementi koji su najčešće prošli kroz sito će biti prvi analizirani, s tim da elementi skupa Z, koji predstavljaju najbolje poznato rešenje, se uvek stavljaju na početak tog niza.

Konačno, ako u particiji N_k ne postoji bolje rešenje od trenutno najboljeg, proverava se particija N^+_{k+1} i ako ni u njoj ne postoji takvo rešenje, algoritam završava sa radom. Teorema 2 i tvrđenja 4 i 5 dokazuju da algoritam KADM nalazi optimalno rešenje problema M, pod uslovom da je sito u rutini *Particija* pravilno implementirano.

4.5 Rešavanje hab lokacijskih problema pomoću APDM i KADM

Algoritmi APDM i KADM su primenjeni na rešavanje problema USAHLP i CSAHLP. Promenljive iz skupa Z u definiciji problema M su zapravo promenljive $z_{ii}, i \in I$ kojima se definišu habovi, a vektor F predstavlja fiksne troškove. Promenljive iz skupa X iz definicije problema M su promenljive $Y_{kl}^{i}, i, k, l \in I$ kojima se računaju troškovi transfera. Promenljive iz skupa Y iz definicije problema M su promenljive $z_{ik}, i, k \in I, i \neq k$ kojima se određuje alokacija čvorova habovima.

MILP problem za USAHLP je predstavljen u poglavlju 2.1. MILP problem za CSAHLP je predstavljen u poglavlju 2.3. Oznaka SAHLP se koristi kada se izloženo odnosi na oba problema. Potproblem $SAHLP(H)$ je definisan u poglavlju 2.4. Oznake M i $M(H)$ će biti korišćene za MILP probleme za SAHLP i $SAHLP(H)$ da bi se slagale sa prethodnim izlaganjem.

4.5.1 Dekompozicija i particionisanje

Saglasno definicijama iz poglavlja 4.2 i 4.4.1, definišu se sledeći MILP problemi, uz napomenu da se skup uslova (2.16) koristi samo za CSAHLP.

1. Nadproblem N se sastoji od funkcije cilja (4.1) i uslova (2.4),(2.5),(2.16) i (2.8).

$$\min \sum_{i \in I} \sum_{k \in I} C_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{k \in I} f_k Z_{kk} \tag{4.1}$$

2. Za neko k, particije M_k i M_k^+ se definišu kao problem M sa uslovima (4.2) i (4.3), redom. Particije nadproblema N_k i N_k^+ jednake su prethodno definisanom nadproblemu N sa uslovima (4.2) odnosno (4.3), redom.

$$\sum_{i \in I} Z_{ii} = k \tag{4.2}$$

$$\sum_{i \in I} Z_{ii} \geq k \tag{4.3}$$

3. Fragment nadproblema $N'(granica)$ se definiše funkcijom cilja (4.4) i uslovima (4.5),(2.4),(2.5),(2.16) i (2.8).

$$\min \sum_{i \in I} \sum_{k \in I} C_{ik} Z_{ik} (\chi O_i + \delta D_i) \tag{4.4}$$

$$\sum_{i \in I} \sum_{k \in I} C_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{k \in I} f_k Z_{kk} \leq granica \tag{4.5}$$

4. $N'_k(granica)$ je $N'(granica)$ sa uslovom (4.2).

5. $N'_k(granica, Z_0, Z_1)$ je $N'_k(granica)$ sa (4.6) i (4.7).

$$Z_{ii} = 0, i \in Z_0 \tag{4.6}$$

$$Z_{ii} = 1, i \in Z_1 \tag{4.7}$$

6. Problem sa funkcijom cilja (4.8) i uslovima (2.18),(2.20) i (2.23) je nadproblem potproblema $M(H)$ i biće označen sa $N(H)$. Ovaj problem, koji sadrži $(n - h) * h$ binarnih promenljivih, koristi se samo za CSAHLP, kako bi se proverilo da li određena konfiguracija habova H ima dovoljan kapacitet da opsluži sve čvorove. Jasno je da je $N(H)$ mnogo lakše rešiti nego $M(H)$. $N(H)$ će biti korišćen kao jedno dodatno sito za CSAHLP.

$$\begin{aligned} \min \sum_{i \in I \setminus H} \sum_{k \in H} C_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{k \in H} C_{kk} (\chi O_k + \delta D_k) + \\ \sum_{i \in I \setminus H} \sum_{k \in H} \sum_{l \in H} \alpha C_{lk} W_{li} Z_{ik} + \sum_{k \in H} \sum_{l \in H} \alpha C_{kl} W_{kl} + \sum_{k \in H} f_k \end{aligned} \tag{4.8}$$

4.5.2 Implementacija KADM

Kombinatorni algoritam dekompozicije modela za SAHLP je implementiran kao što je opisano algoritmima 12, 13 i 14. Za donju granicu sabirka $A * X$, koji predstavlja sumu svih troškova transfera, korišćena je vrednost $ax = 0$.

Neka je $q = \sum_{k \in H} C_{kk} (\chi O_k + \delta D_k) + \sum_{k \in H} \sum_{l \in H} \alpha C_{kl} W_{kl} + \sum_{k \in H} f_k$, suma drugog, petog i šestog sabirka funkcije cilja (2.17) potproblema $M(H)$. Za dato H, q je konstanta, kao što je opisano u poglavlju 2.4. Neka je *granica* vrednost najboljeg poznatog rešenja.

Prilikom rešavanja particije M_h korišćenjem algoritma 13, pored sita $fragment+$

$\sum_{i \in H} F_i \leq granica$ koje je opisano u poglavlju 4.4.1, za dato $H, |H| = h$, određena su sledeća sita koja su specifična za SAHLP i implementirana su navedenim redosledom, počev od onih čija je složenost izračunavanja najmanja. Ukoliko H ne prođe kroz neko od navedenih sita, ono se odmah odbacuje, bez testiranja ostalih.

1. Pri rešavanju problema CSAHLP: ako je $\sum_{k \in H} G_k < \sum_{i \in I} O_i$, H se odbacuje. Ako je zbir kapaciteta svih habova manji od ukupnog toka saobraćaja, habovi neće moći da prime sav taj tok i bar jedan od uslova (2.20) neće biti ispunjen.

2. Kako je $\sum_{k \in H} Z_{ik} = 1$ i Z_{ik} su binarne promenljive, (uslov 2.18), za svako dopustivo rešenje $M(H)$ važi sledeća nejednakost:

$$\sum_{i \in I \setminus H} \sum_{k \in H} C_{ik} Z_{ik} (\chi O_i + \delta D_i) \geq \sum_{i \in I \setminus H} (\chi O_i + \delta D_i) \min_{k \in H} C_{ik} \tag{4.9}$$

   Ova donja granica za prikupljanje i distribuciju je određena tako što se svaki ne-hab čvor $i \in I \setminus H$ alocira habu $k \in H$ sa najmanjom vrednošću troškova transporta C_{ik}. Dakle, ako je $q + \sum_{i \in I \setminus H} (\chi O_i + \delta D_i) \min_{k \in H} C_{ik} > granica$, H se odbacuje, jer je vrednost funkcije cilja i bez troškova transfera veća od gornje granice.

   Vremenska složenost izračunavanja gornje sume je $O(n * h)$. Vrednosti $(\chi O_i + \delta D_i) C_{ik}$ za svako $i, k \in I$ se mogu izračunati na samom početku rada algoritma 14 sa prostornom i vremenskom složenošću n^2. Ako se to preprocesiranje uradi, ovo sito je moguće implementirati bez množenja, korišćenjem samo sabiranja i poređenja. Zapravo, u svakom koraku rutine 13, kada se novi hab dodaje u skup H, tekući minimum vrednosti $\min_{k \in H} C_{ik}$ se ažurira u konstantnom vremenu za svako i, tako da kada su svi habovi dodati u H, navedenu sumu je moguće izračunati u vremenu $O(n)$. Stoga, implementacija ovog sita zahteva $O(n)$ koraka u svakom rekurzivnom koraku rutine 13 i to bez množenja.

3. Na sličan način, kao prethodno sito, određuje se minimum vrednosti:

$$min[q + \sum_{i \in I \setminus H} \sum_{k \in H} C_{ik} Z_{ik} (\chi O_i + \delta D_i) + \sum_{i \in I \setminus H} \sum_{k \in H} \sum_{l \in H} \alpha C_{lk} (W_{li} + W_{il}) Z_{ik}]$$

   tako što se proveravaju sve moguće alokacije Z_{ik} čvora i habu k. Razlika je u tome što se ovom prilikom računa i transfer saobraćaja koji polazi od haba

ka čvorovima i od čvorova ka habu. Transfer saobraćaja, koji polazi od čvora i upućen je čvoru, nije uračunat. Ako je navedena suma veća od vrednosti *granica*, H se odbacuje. Prilikom računanja ove sume, alokacija jednog čvora je nezavisna od alokacije ostalih čvorova, te je računanje ove sume vremenske složenosti $O((n-h)*h^2)$, ali se množenje mora koristiti u svakom koraku za računanje vrednosti $C_{lk}(W_{li}+W_{il})$.

4. Kao što je obrazloženo u poglavlju 2.1, nakon obavljene alokacije, formule (2.2) i (2.3) daju istu vrednost. Za fiksiran skup habova H formula (2.2) postaje:

$$\min[\sum_{k\in H} F_k + \sum_{i\in I}\sum_{k\in H}\sum_{j\in I}\sum_{l\in H} W_{ij}(\chi C_{ik}+\alpha C_{kl}+\delta C_{lj})Z_{ik}Z_{jl}] \tag{4.10}$$

Sledeća nejednakost važi jer za svako i postoji tačno jedno k tako da $Z_{ik}=1$:

$$\begin{aligned}\min[\sum_{k\in H} F_k + \sum_{i\in I}\sum_{k\in H}\sum_{j\in I}\sum_{l\in H} W_{ij}(\chi C_{ik}+\alpha C_{kl}+\delta C_{lj})Z_{ik}Z_{jl}] \geq \\ \sum_{k\in H} F_k + \sum_{i\in I}\min_{k\in H}\{\sum_{j\in I}\min_{l\in H}[W_{ij}(\chi C_{ik}+\alpha C_{kl}+\delta C_{lj})]\}\end{aligned} \tag{4.11}$$

Ova formula daje bolju vrednost donje granice nego sita 2 i 3, ali je vremenske složenosti $O(n^2*h^2)$.

5. Samo za CSAHLP rešava se $N(H)$ pozivanjem $Resenje(N(H), granica)$, gde je $N(H)$ nadproblem potproblema definisan u poglavlju 4.5.1. Ako rešenje $N(H)$ ne postoji zbog ograničenja kapaciteta ili ako je lošije od vrednosti *granica*, H se odbacuje.

Ukoliko H prođe kroz sva ova sita, rešava se $M(H)$ pozivom $Resenje(M(H), granica)$, kao što je navedeno u rutini 13.

4.5.3 Rezultati

KADM i APDM su implementirani u programskom jeziku C. Kao univerzalni rešavač korišćen je CPLEX 12.6. Svi testovi su obavljeni na konfiguraciji sa procesorom Intel Core i5-3470 3.2 GHz sa 8GB RAM, pod Windows 7 Professional operativnim sistemom.

Testirane su sve instance prikazane u poglavlju 3.1.7, što znači ukupno 88 US-AHLP instanci i 30 CSAHLP instanci. Rezultati testova su prikazani u tabelama 4.1, 4.3, 4.4, 4.5 i 4.7. Kolone u tim tabelama označavaju sledeće:

- Naziv test instance;
- Rešenje dobijeno pomoću CPLEX 12.6 rešavača; oznaka "-" znači da CPLEX nije mogao da nađe rešenje zbog memorijskog ograničenja;
- Vreme potrebno da CPLEX pronađe i potvrdi optimalnost rešenja;
- Najbolje rešenje koje je pronašao APDM; *opt* znači isto kao CPLEX;
- Vreme potrebno da APDM dostigne najbolje rešenje;
- Broj iteracija potrebnih da APDM dostigne najbolje rešenje;
- Optimalno rešenje koje je pronašao KADM; *opt* znači isto kao CPLEX;
- Vreme potrebno da KADM dostigne najbolje rešenje;
- Vreme potrebno da KADM dokaže optimalnost rešenja;
- k_{max} - vrednost k u algoritmu 14 za koju je algoritam prekinuo sa radom jer ne postoji rešenje particija N_k i N_{k+1} bolje od najboljeg poznatog;
- Habovi - lista uspostavljenih habova u optimalnom rešenju (počev od 0);

APDM je testiran i zasebno, kao metaheuristička metoda, i u okviru KADM, kao deo egzaktne metode.

Prilikom zasebnog testiranja APDM, kriterijumi zaustavljanja bili su: maksimalno vreme izvršavanja jednako broju čvorova n i maksimum 200 iteracija bez popravljanja najboljeg rešenja. Iz rezultata dobijenih ovim testom vidi se da tako veliki broj iteracija u većini slučajeva nije potreban. APDM je pronašao optimalno rešenje u najviše 10 iteracija na 99 od ukupno 118 testiranih instanci. Zbog toga, prilikom testa egzaktnom metodom KADM, kriterijum zaustavljanja za APDM je bio samo jedan: maksimalni broj iteracija bez popravke najboljeg rešenja jednak $ln|Z|$. Kako su korišćeni različiti kriterijumi zaustavljanja, vreme izvršavanja APDM unutar KADM je manje nego prilikom zasebnog testiranja APDM.

U tabelama 4.1 i 4.2 prikazani su rezultati za sve CSAHLP test primere. Pomoću univerzalnog rešavača CPLEX 12.6 dokazana je optimalnost rešenja na test primerima do 100 čvorova, ali ne i na test primerima sa 200 i 300 čvorova, zbog memorijskog ograničenja. KADM nije imao problem sa memorijom, jer KADM nikada ne rešava pun MILP problem sa $O(n^3)$ promenljivih. KADM je pronašao rešenje koje je bolje od prethodno najboljeg poznatog na test primerima $200LT$, $200TT$ i $300TT$ i dokazao optimalnost rešenja na svih 30 instanci. Vreme izvršavanja KADM na test primerima sa preko 25 čvorova je daleko bolje od vremena koje je bilo potrebno rešavaču CPLEX. KADM je pronašao optimalna rešenja za sve probleme sa najviše 100 čvorova za manje od jednog minuta, dok je CPLEX rešavaču bilo potrebno od 10 minuta do gotovo 2 sata. Na 3 lakša test primera sa više od 100 čvorova KADM se izvršavao manje od 5 minuta, dok je probleme na 3 teža primera rešavao od približno sat vremena do oko 4 sata. Te test primere paket CPLEX uopšte nije mogao da reši. U proseku, KADM je bio preko 75 puta brži od CPLEX rešavača pri rešavanju problema koje je CPLEX mogao da reši.

U radu [16] je objavljeno rešenje 231069.50 za test primer $200LL$. KADM je dokazao da je to rešenje pogrešno, te da je optimalno rešenje za tu instancu 241992.97.

Kao što je objašnjeno u poglavlju 3.1.7, uobičajeno je da se svaka USAHLP instanca testira 3 puta, sa različitim koeficijentom fiksnih troškova λ. Rezultati tih testova dati su u tabelama 4.3,4.4, 4.5 i 4.6.

Univerzalni rešavač CPLEX 12.6 je pronašao i potvrdio optimalnost rešenja na test primerima do dimenzije 130, ali test primere dimenzije 200 nije mogao da reši. KADM je uspešno i efikasno rešio sve probleme i pri tom pronašao dva rešenja koja su bolja od prethodno najboljih poznatih, oba za test primer 200TT: sa koeficijentom fiksnih troškova 0.9 najbolje poznato rešenje je bilo 266134.11, ali je KADM pronašao rešenje 266116.32 i potvrdio njegovu optimalnost; za koeficijent fiksnih troškova 0.8, najbolje poznato rešenje je bilo 258823.13, ali je KADM dokazao da je rešenje 258797.48 optimalno.

KADM je i do 100 puta brži od CPLEX rešavača pri rešavanju onih USAHLP problema koje je CPLEX mogao da reši. KADM je najduže rešavao test primer 130LL za $lambda = 0.8$ i to 86.19 sekundi, što je 27.7 puta brže od CPLEXa. U proseku, KADM je bio oko 50 puta brži od CPLEXa pri rešavanju ovih problema.

Tabela 4.1: Optimalna rešenja na test instancama za CSAHLP ($n \leq 100$)

Test	CPLEX 12.6		APDM			KADM				
inst.	Rešenje	t(s)	Reš.	t(s)	Iter.	Reš.	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
10LL	224250.05	0.05	opt	0.06	2	opt	0.06	0.31	6	6, 3, 2
10LT	250992.26	0.06	opt	0.01	1	opt	0.01	0.46	7	0, 3, 4, 9
10TL	263399.94	0.08	opt	0.13	2	opt	0.13	0.31	6	3, 4, 9
10TT	263399.94	0.05	opt	0.19	1	opt	0.19	0.45	6	3, 4, 9
20LL	234690.96	0.26	opt	0.10	2	opt	0.10	0.37	6	13, 6
20LT	253517.40	0.37	opt	0.74	1	opt	0.74	1.12	7	13, 9
20TL	271128.18	0.26	opt	0.02	1	opt	0.02	0.26	6	6, 18
20TT	296035.40	0.58	opt	0.11	1	opt	0.11	0.64	7	0, 9, 18
25LL	238977.95	0.87	opt	0.24	2	opt	0.24	0.70	7	7, 17
25LT	276372.50	2.00	opt	0.33	2	opt	0.33	4.43	8	8, 15, 24
25TL	310317.64	0.57	opt	0.02	1	opt	0.02	0.45	5	8, 22
25TT	348369.15	2.51	opt	0.31	2	opt	0.31	1.52	6	8, 15, 24
40LL	241955.71	4.62	opt	0.59	1	opt	0.59	0.76	6	28, 10
40LT	272218.32	4.31	opt	0.44	1	opt	0.44	3.69	7	13, 25, 29
40TL	298919.01	4.71	opt	0.20	3	opt	0.20	0.73	5	13, 18
40TT	354874.10	6.09	opt	1.67	8	opt	2.20	2.89	7	13, 39, 18
50LL	238520.59	13.46	opt	1.36	12	opt	0.60	0.97	7	34, 14
50LT	272897.49	36.68	opt	0.17	1	opt	0.17	7.65	8	25, 5, 31, 45
50TL	319015.77	17.21	opt	0.47	4	opt	0.47	1.15	5	2, 23
50TT	417440.99	3134.65	opt	12.35	24	opt	6.42	11.71	8	5, 25, 47, 27
100LL	246713.97	972.32	opt	2.14	3	opt	2.14	23.33	7	28, 63, 72
100LT	256155.33	909.04	opt	1.30	1	opt	1.30	32.20	8	75, 28, 67
100TL	362950.09	587.01	opt	1.62	2	opt	1.62	18.06	6	51, 43
100TT	474068.96	7022.95	opt	5.86	1	opt	5.86	52.65	8	4, 33, 85, 94
prosek		530.03	opt	1.27	3.29	opt	1.01	6.95		

Tabela 4.2: Optimalna rešenja na CSAHLP test instancama ($n = 200, 300$)

Test	APDM			KADM				
inst.	Rešenje	t(s)	Iter.	Rešenje	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
200LL	241992.97	6.41	1	241992.97	6.41	258.84	7	42, 158
200LT	**267218.35**	91.85	10	**267218.35**	91.23	4679.37	9	167, 123, 147, 40
200TL	273443.81	3.55	1	273443.81	3.55	147.38	8	53, 185, 94
200TT	**290582.04**	23.91	2	**290582.04**	23.91	262.97	9	167, 53, 185, 112
300LL	227667.00	42.74	2	227667.00	42.74	14562.28	9	226, 89, 189
300TT	**395968.65**	265.79	1	**395968.65**	265.79	3659.10	9	155, 12, 153, 130, 282
prosek		72.38	2.83		72.27	3928.32		

Tabela 4.3: Optimalna rešenja na USAHLP test instancama dimenzije $n \leq 130$ sa koeficijentom fiksnih troškova $\lambda = 1$

Test	CPLEX 12.6		APDM			KADM				
inst.	Rešenje	t(s)	Reš.	t(s)	Iter.	Reš.	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
10LL	224250.05	0.05	opt	0.08	2	opt	0.08	0.18	6	6, 3, 2
10TT	263399.94	0.07	opt	0.19	3	opt	0.19	0.28	6	9, 3, 4
20LL	234690.96	0.24	opt	0.11	3	opt	0.11	0.34	6	13, 6
20TT	271128.18	0.24	opt	0.03	2	opt	0.03	0.31	6	6, 18
25LL	236650.63	0.65	opt	0.16	3	opt	0.16	0.61	7	7, 17
25TT	295667.84	0.48	opt	0.45	8	opt	0.21	0.25	5	12
40LL	240986.23	6.77	opt	0.25	5	opt	0.25	0.59	7	13, 27
40TT	293164.84	3.24	opt	0.22	4	opt	0.22	0.49	4	18
50LL	237421.99	12.53	opt	1.53	15	opt	0.67	1.00	7	35, 14
50TT	300420.99	11.99	opt	1.45	16	opt	0.69	0.94	5	23
60LL	228007.90	26.73	opt	0.69	6	opt	0.73	1.97	7	40, 17
60TT	246285.03	23.97	opt	0.11	2	opt	0.11	0.50	4	18, 40
70LL	233154.29	85.04	opt	4.40	21	opt	2.40	3.96	8	18, 51
70TT	252882.63	50.31	opt	0.15	2	opt	0.15	1.20	5	18, 51
80LL	229240.37	191.76	opt	1.75	9	opt	2.11	3.88	8	54, 21
80TT	274921.57	111.42	opt	0.29	2	opt	0.29	2.85	6	4, 40, 51
90LL	231236.23	319.42	opt	13.99	40	opt	3.39	6.44	8	81, 25
90TT	280755.46	177.65	opt	0.47	3	opt	0.47	2.97	6	4, 40
100LL	238016.28	380.68	opt	0.34	2	opt	0.34	7.30	7	28, 72
100TT	305097.95	246.02	opt	0.37	2	opt	0.37	2.39	4	51
110LL	222704.77	494.03	opt	0.92	2	opt	0.92	5.17	7	31, 76
110TT	227934.63	475.99	opt	1.12	3	opt	1.12	5.08	6	76, 31
120LL	225801.36	820.07	opt	0.95	3	opt	0.95	9.24	7	31, 84
120TT	232460.82	628.31	opt	0.48	2	opt	0.48	7.03	6	31, 84
130LL	227884.63	1291.45	opt	9.91	14	opt	6.19	17.01	7	87, 35
130TT	234935.97	1175.34	opt	4.13	8	opt	4.07	13.64	7	35, 87
prosek		251.33	opt	1.71	7	opt	1.03	3.68		

Tabela 4.4: Optimalna rešenja na USAHLP test instancama dimenzije $n \leq 130$ sa koeficijentom fiksnih troškova $\lambda = 0.9$

Test	CPLEX 12.6		APDM			KADM				
inst.	Rešenje	t(s)	Reš.	t(s)	Iter.	Reš.	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
10LL	215425.86	0.06	opt	0.10	3	opt	0.10	0.32	7	6, 3, 2
10TT	253798.39	0.07	opt	0.04	2	opt	0.04	0.25	7	3, 2, 9
20LL	228785.69	0.33	opt	0.11	3	opt	0.11	0.46	7	13, 6
20TT	263350.01	0.24	opt	0.03	2	opt	0.03	0.25	6	6, 18
25LL	230539.76	0.67	opt	0.62	17	opt	0.39	0.47	7	17, 7
25TT	288066.67	0.67	opt	0.03	2	opt	0.03	0.29	6	8, 23
40LL	234013.74	5.19	opt	0.11	2	opt	0.11	0.71	7	21, 10, 27
40TT	289382.07	3.38	opt	0.20	4	opt	0.20	0.57	5	18
50LL	231658.98	13.88	opt	34.15	148	opt	0.61	1.01	7	14, 35
50TT	291435.49	11.76	opt	0.15	3	opt	0.15	0.66	5	16, 47
60LL	223905.16	34.59	opt	8.22	44	opt	1.48	2.93	8	17, 40
60TT	240748.21	24.18	opt	0.11	2	opt	0.11	0.56	5	18, 40
70LL	228337.78	117.86	opt	18.11	71	opt	2.35	5.18	8	51, 18, 45
70TT	246810.18	50.92	opt	0.14	2	opt	0.14	1.36	6	18, 51
80LL	224334.54	209.54	opt	1.14	6	opt	2.12	5.31	8	54, 21
80TT	266036.30	106.30	opt	0.28	2	opt	0.28	2.89	6	4, 40, 51
90LL	226769.05	463.64	opt	25.88	71	opt	4.15	11.28	9	25, 81, 57
90TT	275248.47	184.40	opt	0.47	3	opt	0.47	2.82	6	4, 40
100LL	232712.96	447.89	opt	16.85	32	opt	3.24	14.02	8	28, 72
100TT	301719.67	248.64	opt	0.36	2	opt	0.36	3.53	5	51
110LL	218620.41	562.00	opt	1.20	3	opt	1.20	9.66	8	31, 76
110TT	223327.29	492.44	opt	1.89	5	opt	1.89	6.02	7	76, 31
120LL	221693.72	1197.52	opt	3.90	8	opt	4.01	15.07	7	31, 84
120TT	227687.23	654.52	opt	1.00	3	opt	1.00	10.57	7	84, 31
130LL	223280.66	1566.42	opt	1.87	4	opt	1.87	30.27	8	87, 35
130TT	229626.87	1471.59	opt	2.09	5	opt	2.09	18.20	7	87, 35
prosek		302.64	opt	4.58	17.27	opt	1.10	5.56		

Tabela 4.5: Optimalna rešenja na USAHLP test instancama dimenzije $n \leq 130$ sa koeficijentom fiksnih troškova $\lambda = 0.8$

Test	CPLEX 12.6		APDM			KADM				
inst.	Rešenje	t(s)	Reš.	t(s)	Iter.	Reš.	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
10LL	206555.04	0.07	opt	0.01	1	opt	0.01	0.21	7	6, 0, 3, 4
10TT	242936.29	0.07	opt	0.03	1	opt	0.03	0.22	7	0, 3, 4, 9
20LL	222085.77	0.33	opt	0.20	8	opt	0.30	0.43	7	13, 5, 10
20TT	255571.84	0.25	opt	0.03	2	opt	0.03	0.24	7	6, 18
25LL	224428.90	0.98	opt	0.28	8	opt	0.82	0.95	8	7, 17
25TT	279548.43	0.67	opt	0.09	3	opt	0.09	0.51	6	8, 23
40LL	226032.09	5.24	opt	0.07	2	opt	0.07	0.61	8	21, 10, 27
40TT	285599.30	4.80	opt	0.39	7	opt	0.30	0.45	6	18
50LL	225895.98	21.83	opt	49.80	224	opt	0.66	1.30	8	14, 35
50TT	281803.92	11.81	opt	0.15	3	opt	0.15	0.75	6	16, 47
60LL	218573.39	35.35	opt	0.19	2	opt	0.19	4.68	9	38, 17, 54
60TT	235211.39	26.15	opt	0.10	2	opt	0.10	0.78	5	18, 40
70LL	221899.15	138.58	opt	5.33	26	opt	2.74	8.88	9	51, 18, 45
70TT	240737.73	50.88	opt	0.14	2	opt	0.14	1.33	6	18, 51
80LL	217929.65	264.26	opt	3.77	18	opt	3.29	10.40	9	54, 21, 43
80TT	257151.03	104.06	opt	0.28	2	opt	0.28	3.10	7	4, 40, 51
90LL	220106.77	578.91	opt	19.75	62	opt	2.96	17.97	9	25, 81, 57
90TT	268373.84	197.34	opt	0.26	2	opt	0.26	3.47	7	4, 81
100LL	227409.64	867.64	opt	1.31	5	opt	1.31	35.60	8	28, 72
100TT	297646.25	267.09	opt	0.31	2	opt	0.31	3.79	5	4, 51
110LL	214536.06	677.12	opt	8.25	18	opt	4.58	16.47	8	76, 31
110TT	218719.95	568.23	opt	2.92	7	opt	2.73	6.52	7	76, 31
120LL	215865.55	1359.88	opt	5.38	11	opt	4.92	24.52	8	31, 84, 76
120TT	222680.29	715.76	opt	1.10	3	opt	1.10	10.10	7	76, 31, 84
130LL	218676.70	2386.74	opt	3.17	6	opt	3.34	86.19	9	87, 35
130TT	224317.78	1537.16	opt	1.67	4	opt	1.67	25.79	8	87, 35
prosek		377.84	opt	4.04	16.58	opt	1.25	10.20		

Tabela 4.6: Optimalna rešenja na USAHLP test instancama dimenzije $n = 200$

Test		APDM			KADM				
inst.	λ	Rešenje	t(s)	Iter.	Rešenje	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
200LL	1	233802.98	2.74	2	233802.98	2.74	97.90	7	42, 147
200TT	1	*273435.15*	13.01	3	272188.11	37.37	108.17	8	53, 121
200LL	0.9	228753.70	12.93	6	228753.70	13.44	246.57	7	42, 147
200TT	0.9	**266116.32**	11.87	3	**266116.32**	11.87	113.09	8	53, 185, 94
200LL	0.8	223704.42	12.67	5	223704.42	12.67	783.83	8	42, 147
200TT	0.8	**258797.48**	7.39	2	**258797.48**	7.39	151.33	9	53, 185, 94
prosek			10.10	3.5		14.24	250.15		

Na ovim USAHLP i CSAHLP test primerima, APDM je pronašao pet rešenja koja su bolja od prethodno najboljih, a nije pronašao samo jedno 1 optimalno rešenje. To znači da je APDM uporediv sa najboljim poznatim metaheurističkim metodama.

Na kraju su testirana i 4 primera velikih dimenzija sa 300 i 400 čvorova, koje paket CPLEX ne može da reši. Rezultati su prikazani u tabeli 4.7. APDM je našao dobra rešenja ovih problema u samo 2, 3 i 4 iteracije, međutim, na 3 od 4 od njih ta rešenja nisu optimalna. Za test primere većih dimenzija potrebno je promeniti kriterijum zaustavljanja APDM i dozvoliti više vremena, ili koristiti neku od modifikacija predloženih u poglavlju 4.3. Sa druge strane, KADM je efikasno pronašao i dokazao optimalnost rešenja na 2 od 4 test primera. KADM je takođe dokazao optimalnost rešenja i na test primeru ap300L, ali posle nekoliko dana izračunavanja, dok je za samo pronalaženje tog optimalnog rešenja bilo potrebno manje od minut. Test primer ap400L ostaje jedini za koji nije dokazana optimalnost rešenja. Nakon rešavanja nekoliko particija, procenjeno je ukupno vreme izvršavanja i ono se meri mesecima. KADM je prekinut nakon što je završio rešavanje particije M_6. Nakon toga, rešeni su i nadproblemi N_k za $k > 6$, kako bi se pronašla vrednost $k_{max} = 11$. Dakle, teoretski je moguće da u particijama M_k za $7 \leq k \leq 10$ postoji bolje rešenje od 267873.65, iako se sa velikom sigurnošću može reći da je to rešenje optimalno. Iako KADM nije dokazao optimalnost, rešenje koje jeste pronašao je bolje od prethodno najboljeg poznatog rešenja 267921.71. $\binom{400}{4} \approx 10^9$, međutim, KADM je pronašao najbolje rešenje, u kome su uspostavljena 4 haba, za manje od 400 sekundi.

Tabela 4.7: Rešenja na USAHLP test instancama velikih dimenzija ($n = 300, 400$)

Test	APDM			KADM				
inst.	Rešenje	t(s)	Iter.	Rešenje	t(s)	$t_{kraj}(s)$	k_{max}	Habovi
ap300T	*283540.95*	70.21	3	276023.35	89.87	3579.49	11	189, 29, 153
ap400T	*284124.88*	172.45	4	284037.25	192.96	11102.97	11	178, 371, 100
ap300L	263913.15	46.29	2	263913.15	46.29	790423.44	11	169, 25, 286, 250, 78
ap400L	*268586.40*	96.35	2	**267873.65**	395.69	-	11	335, 179, 98, 302
prosek		96.33	2.75		181.20	-		

Poglavlje 5

Zaključak

U ovom radu su predstavljene egzaktne i metaheurističke metode za rešavanje tri lokacijska problema: USAHLP, CSAHLP i MLUFLP. Ti problemi, koji imaju značajnu primenu u oblasti operacionih istraživanja, pripadaju klasi $\mathbb{NP}$-teških problema i do sada su bile poznate različite metaheurističke metode za njihovo rešavanje.

Predstavljena metoda za rešavanje problema USAHLP sastoji se od genetskog algoritma sa dve lokalne pretrage. Korišćeno je mešovito binarno i celobrojno kodiranje. Cilj prve lokalne pretrage je pronalaženje bolje lokacije uspostavljenih habova, dok se druga lokalna pretraga bavi popravljanjem alokacije čvorova habovima.

Problem MLUFLP je rešavan memetskim algoritmom, zasnovanom na genetskom algoritmu sa lokalnom pretragom. U genetskom algoritmu korišćeno je binarno kodiranje, kojim se određuju lokacije habova. Lokalna pretraga služi da pronađe bolje lokacije habova. Za alokaciju klijenata habovima primenjena je metoda dinamičkog programiranja.

Hibridna metoda EA-BnB koja se sastoji od evolutivnog algoritma i metode grananja i ograničavanja primenjena je na rešavanje CSAHLP. Evolutivni algoritam pronalazi dobre konfiguracije habova i vrši alokaciju čvorova habovima u polinomskom vremenu izvršavanja. Primenjena metoda grananja i ograničavanja popravlja tu alokaciju za izabrane jedinke iz populacije; ona je paralelizovana i primenjena na specifičan način, kao heuristička metoda, sa određenim parametrima koji povećavaju efikasnost algoritma.

Konačno, definisana je metoda dekompozicije matematičkog modela na nadprobleme i potprobleme i predstavljena su dva algoritma: iterativni algoritam proste dekompozicije modela APDM i kombinatorni algoritam dekompozicije mo-

dela KADM. Metoda i algoritmi su izloženi na opšti način, te su kao takvi primenljivi na široku klasu lokacijskih problema. Oba algoritma se mogu primeniti i kao metaheuristička i kao egzaktna metoda; predviđeno je da APDM bude prekinut posle određenog broja iteracija, kada pronađe kvalitetno dopustivo rešenje, dok je KADM prvenstveno dizajniran kao egzaktna metoda koja polazi od rešenja koje je pronašao APDM a zatim ga popravlja i dokazuje optimalnost.

5.1 Naučni doprinos rada

Naučni doprinos ove disertacije predstavljaju sledeći najvažniji rezultati:

- Razvoj hibridne EA-BnB metode za rešavanje CSAHLP problema, sa naglaskom na BnB metodi koja je primenjena na originalan način i dala kvalitetna rešenja;
- Razvoj algoritama lokalne pretrage, čijom primenom se unapređuje efikasnost evolutivnih algoritama;
- Predlog metode dekompozicije modela jedne klase lokacijskih problema, uključujući formulacije i dokaze kojima se postavljaju osnove za razvoj algoritama APDM i KADM;
- Razvoj algoritma proste dekompozicije modela APDM, koji iterativno popravlja rešenja korišćenjem predložene dekompozicije, a čija je tačnost i efikasnost uporediva sa najboljim poznatim metaheurističkim metodama;
- Razvoj kombinatornog algoritma dekompozicije modela KADM koji pronalazi i dokazuje optimalnost rešenja;
- Primena predložene metode dekompozicije i algoritama KADM i APDM na dva poznata lokacijska problema;
- Rešavanje problema USAHLP i CSAHLP čije su instance velikih dimenzija i pronalaženje boljih rešenja od poznatih uz dokaz optimalnosti.

Iz svega prethodno navedenog, može se zaključiti da su razvijene metode od izuzetnog značaja za bolje i efikasnije rešavanje lokacijskih problema. Istraživanje prikazano u ovom radu predstavlja doprinos u oblasti optimizacije lokacijskih problema, kombinatorne optimizacije i u razvoju egzaktnih i metaheurističkih metoda.

Literatura

[1] K. Aardal, F. A. Chudak, and D. B. Shmoys. "A 3-approximation algorithm for the k-level uncapacitated facility location problem". *Information Processing Letters* 72(5) (1999), pp. 161–167.

[2] S. Abdinnour-Helm. "A hybrid heuristic for the uncapacitated hub location problem". *European Journal of Operational Research* 106(2–3) (1998), pp. 489 –499.

[3] S. Abdinnour-Helm and M. Venkataramanan. "Solution approaches to hub location problems". *Annals of Operations Research* 78(0) (), pp. 31–50.

[4] R. Abyazi-Sani and R. Ghanbari. "An efficient tabu search for solving the uncapacitated single allocation hub location problem". *Computers & Industrial Engineering* 93 (2016), pp. 99–109.

[5] S. Alumur and B. Y. Kara. "Network hub location problems: The state of the art". *European Journal of Operational Research* 190(1) (2008), pp. 1 –21.

[6] A. Bailey, B. Ornbuki-Berrnan, and S. Asobiela. "Discrete pso for the uncapacitated single allocation hub location problem" in *Computational intelligence in production and logistics systems (CIPLS), 2013 IEEE workshop on*. IEEE. 2013, pp. 92–98.

[7] J. E. Beasley. "OR-Library: distributing test problems by electronic mail". *Journal of the operational research society* 41(11) (1990), pp. 1069–1072.

[8] J. E. Beasley. "Obtaining test problems via internet". *Journal of Global Optimization* 8(4) (1996), pp. 429–433.

[9] G. Bruno, A. Genovese, and G. Improta. "A historical perspective on location problems". *BSHM Bulletin: Journal of the British Society for the History of Mathematics* 29(2) (2014), pp. 83–97.

[10] J. F. Campbell. "Integer programming formulations of discrete hub location problems". *European Journal of Operational Research* 72(2) (1994), pp. 387 –405.

[11] J. F. Campbell and M. E. O'Kelly. "Twenty-Five Years of Hub Location Research". *Transportation Science* 46(2) (2012), pp. 153–169. eprint: `http://dx.doi.org/10.1287/trsc.1120.0410`.

[12] M. Charikar and S. Guha. "Improved combinatorial algorithms for the facility location and k-median problems" in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 378–388.

[13] J.-F. Chen. "A hybrid heuristic for the uncapacitated single allocation hub location problem". *Omega* 35(2) (2007), pp. 211 –220.

[14] F. A. Chudak and D. B. Shmoys. "Improved approximation algorithms for the uncapacitated facility location problem". *SIAM Journal on Computing* 33(1) (2003), pp. 1–25.

[15] F. A. Chudak and D. P. Williamson. "Improved approximation algorithms for capacitated facility location problems" in *Integer programming and combinatorial optimization*. Springer, 1999, pp. 99–113.

[16] I. Contreras, J. Díaz, and E. Fernández. "Lagrangean relaxation for the capacitated hub location problem with single assignment". *OR spectrum* 31(3) (2009), pp. 483–505.

[17] I. Contreras and E. Fernández. "General network design: A unified view of combined location and network design problems". *European Journal of Operational Research* 219(3) (2012), pp. 680–697.

[18] I. Correia, S. Nickel, and F. Saldanha Da Gama. "The capacitated single-allocation hub location problem revisited: A note on a classical formulation". *European Journal of Operational Research* 207(1) (2010), pp. 92 –96.

[19] P. Crescenzi and V. Kann. *A compendium of NP optimization problems*. ftp://ftp.nada.kth.se/Theory/Viggo-Kann/compendium.pdf, 1998.

[20] T. D., M. N., K. J., and F. V. *Genetski algoritmi*. Matematički Institut SANU, 2004.

[21] M. da Graca Costa, M. E. Captivo, and J. Clímaco. "Capacitated single allocation hub location problem—A bi-criteria approach". *Computers & Operations Research* 35(11) (2008), pp. 3671–3695.

[22] R. Dawkins. *The selfish gene*. 199. Oxford university press, 2006.

[23] P. Dearing. "Location problems". *Operations Research Letters* 4(3) (1985), pp. 95 –98.

[24] Z. Drezner and H. W. Hamacher. *Facility Location: Applications and Theory*. Berlin: Springer-Verlag, 2002.

[25] J. Ebery, M. Krishnamoorthy, A. Ernst, and N. Boland. "The capacitated multiple allocation hub location problem: Formulations and algorithms". *European Journal of Operational Research* 120(3) (2000), pp. 614–631.

[26] U. Eckhardt. "Weber's problem and Weiszfeld's algorithm in general spaces". *Mathematical Programming* 18(1) (1980), pp. 186–196.

[27] N. J. Edwards. "Approximation algorithms for the multi-level facility location problem". PhD thesis. Cornell University, 2001.

[28] A. T. Ernst and M. Krishnamoorthy. "Efficient algorithms for the uncapacitated single allocation p-hub median problem". *Location Science* 4(3) (1996). Hub Location, pp. 139 –154.

[29] A. T. Ernst and M. Krishnamoorthy. "Exact and heuristic algorithms for the uncapacitated multiple allocation p-hub median problem". *European Journal of Operational Research* 104(1) (1998), pp. 100 –112.

[30] A. Ernst and M. Krishnamoorthy. "Solution algorithms for the capacitated single allocation hub location problem". *Annals of Operations Research* 86(0) (1999), pp. 141–159.

[31] R. Z. Farahani, M. Hekmatfar, A. B. Arabani, and E. Nikbakhsh. "Hub location problems: A review of models, classification, solution techniques, and applications". *Computers & Industrial Engineering* 64(4) (2013), pp. 1096–1109.

[32] V. Filipović. "Fine-grained tournament selection operator in genetic algorithms". *Computing and Informatics* 22(2) (2012), pp. 143–161.

[33] V. Filipović, J. Kratica, D. Tošić, and D. Dugošija. "Applications of Soft Computing: From Theory to Praxis" in ed. by J. Mehnen, M. Köppen, A. Saad, and A. Tiwari. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. GA Inspired Heuristic for Uncapacitated Single Allocation Hub Location Problem, pp. 149–158.

[34] R. Galvao, L. Espejo, and B. Boffey. "A hierarchical model for the location of perinatal facilities in the municipality of Rio de Janeiro". *European Journal of Operational Research* 138(3) (2002), pp. 495–517.

[35] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[36] A. Goldman. "Optimal location for centers in a network". *Transportation Science* 3 (1969), pp. 352–360.

[37] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[38] C. J.F., E. A., and K. M. *Hub Location Problems in Hamacher H, Drezner Z, Facility Location: Applications and Theory.* Springer-Verlag, Berlin-Heidelberg, 2002, pp. 373–407.

[39] K. J.G. "Heuristics for the p-hub location problem". *European Journal of Operational Research* 53(1) (1991), pp. 25 –37.

[40] J. G. Klincewicz. "Avoiding local optima in thep-hub location problem using tabu search and GRASP". *Annals of Operations Research* 40(1) (), pp. 283–302.

[41] J. Krarup and P. M. Pruzan. "The simple plant location problem: Survey and synthesis". *European Journal of Operational Research* 12(1) (1983), pp. 36 –81.

[42] E. Lawler and D. Wood. "Branch-and-Bound Methods: A Survey". *Operations Research* 14(4) (1966), pp. 699–719. eprint: `http://dx.doi.org/10.1287/opre.14.4.699`.

[43] M. Marić. "Rešavanje nekih NP–teških hijerarhijsko-lokacijskih problema primenom genetskih algoritama". *Doktorska disertacija, Matematički fakultet, Beograd* (2008).

[44] M. Marić. "An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem". *Computing and Informatics* 29(2) (2010), pp. 183–201.

[45] M. Marić, Z. Stanimirović, and S. Božović. "Hybrid metaheuristic method for determining locations for long-term health care facilities". *Annals of Operations Research* 227(1) (2015), pp. 3–23.

[46] M. Marić, Z. Stanimirović, and P. Stanojević. "An efficient memetic algorithm for the uncapacitated single allocation hub location problem". *Soft Computing* 17(3) (2013), pp. 445–466.

[47] M. Marić, Z. Stanimirović, A. Djenić, and P. Stanojević. "Memetic Algorithm for Solving the Multilevel Uncapacitated Facility Location Problem". *Informatica* 25(3) (2014), pp. 439–466.

[48] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., 1990.

[49] J. Meier and U Clausen. *Solving classical and new single allocation hub location problems on euclidean data*. Tech. rep. Tech. rep., Optimization Online, http://www. optimization-online. org/DB_HTML/2015/03/4816. html, 2015.

[50] Z. Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 2013.

[51] P. Moscato et al. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[52] M. E. O'Kelly. "The Location of Interacting Hub Facilities". *Transportation Science* 20(2) (1986), pp. 92–106. eprint: `http://dx.doi.org/10.1287/trsc.20.2.92`.

[53] M. E. O'Kelly. "A quadratic integer program for the location of interacting hub facilities". *European Journal of Operational Research* 32(3) (1987), pp. 393 –404.

[54] M. E. O'Kelly. "HUB FACILITY LOCATION WITH FIXED COSTS". *Papers in Regional Science* 71(3) (1992), pp. 293–306.

[55] M. Peker, B. Y. Kara, J. F. Campbell, and S. A. Alumur. "Spatial Analysis of Single Allocation Hub Location Problems". *Networks and Spatial Economics* (2015), pp. 1–27.

[56] G. R. Raidl and J. Gottlieb. "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem". *Evolutionary Computation* 13(4) (2005), pp. 441–475.

[57] M. Randall. "Solution approaches for the capacitated single allocation hub location problem using ant colony optimisation". *Computational Optimization and Applications* 39(2) (2008), pp. 239–261.

[58] B. Rostami, C. Buchheim, J. F. Meier, and U. Clausen. "Lower bounding procedures for the single allocation hub location problem". *Electronic Notes in Discrete Mathematics* 52 (2016), pp. 69–76.

[59] M. R. Silva and C. B. Cunha. "New simple and efficient heuristics for the uncapacitated single allocation hub location problem". *Computers & Operations Research* 36(12) (2009). New developments on hub location, pp. 3152 –3165.

[60] D. Skorin-Kapov and J. Skorin-Kapov. "On tabu search for the location of interacting hub facilities". *European Journal of Operational Research* 73(3) (1994), pp. 502 –509.

[61] Z. Stanimirović. "Solving the capacitated single allocation hub location problem using genetic algorithm". *Recent advances in stochastic modelling and data analysis. World Scientific Publishing Co Pte Ltd* (2007), pp. 464–471.

[62] Z. Stanimirović. "An efficient genetic algorithm for the uncapacitated multiple allocation p-hub median problem." *Control & Cybernetics* 37(3) (2008).

[63] Z. Stanimirović, M. Marić, N. Radojičić, and S. Božović. "Two efficient hybrid metaheuristic methods for solving the load balance problem". *Applied and Computational Mathematics* 13(3) (2014), pp. 332–349.

[64] P. Stanojević. "A Parallel Branch and Bound Algorithm with Restarts for Solving the Hierarchical Covering Location Problem". *Mathematica Balkanica* 25 (2011), pp. 555–567.

[65] P. Stanojević, M. Marić, and Z. Stanimirović. "A hybridization of an evolutionary algorithm and a parallel branch and bound for solving the capacitated single allocation hub location problem". *Applied Soft Computing* 33 (2015), pp. 24–36.

[66] B. C. Tansel, R. L. Francis, and T. J. Lowe. "State of the art—location on networks: a survey. Part I: the p-center and p-median problems". *Management Science* 29(4) (1983), pp. 482–497.

[67] H. Topcuoglu, F. Corut, M. Ermis, and G. Yilmaz. "Solving the uncapacitated hub location problem using genetic algorithms". *Computers & Operations Research* 32(4) (2005), pp. 967 –984.

[68] M. Živković. *Algoritmi*. Matematički Fakultet.

Biografija

Osnovni podaci Predrag Stanojević je rođen 24.7.1976. godine u Beogradu. Osnovnu školu „Veselin Masleša“ završio je kao nosilac Vukove diplome i đak generacije. Završio je tri godine Matematičke gimnazije, dok je četvrtu godinu završio u Sjedinjenim Američkim Državama u školi „Ridley Senior High School“ u gradu Folsom u Pensilvaniji. Diplomirao je na Matematičkom fakultetu, smer Računarstvo i informatika. Godine 2008. upisao je doktorske studije na istom smeru. Položio je sve ispite predviđene planom i programom doktorskih studija sa prosečnom ocenom 10,00. Pored kurseva predviđenih programom, pohađao je i položio dva kursa u okviru projekta „SEE Doctoral Studies in Mathematical Sciences“ (144703-TEMPUS-2008-BA-JCPR).

Učešće na projektima i konferencijama

- „SEE Doctoral Studies in Mathematical Sciences“, 144703-TEMPUS-2008-BA-JCPR, kursevi: „PhD Course Data Structures and High Performance Computing“, Activity II.2.3, Skopje, Makedonija, Novembar 2011, Complete Course Grade A i „PhD Course Algebraic Combinatorics, Computability and Complexity“, Activity II.2.6, Sofia, Bulgaria, Oktobar 2011, Final Grade A
- MASSEE, International Congress on Mathematics, MICOM 2009, „SEE Doctoral Studies in Mathematics“, SEE Young Researchers Workshop, Septembar 2009, Ohrid, Makedonija
- ETRAN 53, Jun 2009, Vrnjačka Banja

Ostali podaci Živeo je u Australiji, gde je radio kao programer 2 godine. Profesionalno se bavi enigmatikom od 2005. godine. Autor je više od 140 zbirki enigmatskih problema.

Прилог 1.

Изјава о ауторству

Потписани-а Предраг Станојевић

број уписа 2018/2009

Изјављујем

да је докторска дисертација под насловом

Егзактне и метахеуристичке методе за решавање НП-тешких локацијских проблема

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 12.9.2016. године

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: Предраг Станојевић

Број уписа: 2018/2009

Студијски програм: Информатика

Наслов рада: Егзактне и метахеуристичке методе за решавање НП-тешких локацијских проблема

Ментор: др Мирослав Марић, ванредни професор

Потписани: Предраг Станојевић

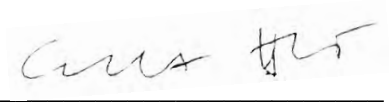
изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 12.9.2016. године



Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

“Егзактне и метахеуристичке методе за решавање НП-тешких локацијских проблема”

која је моје ауторско дело.

Дисертацију са свим прилозима предао сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

④. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 12.9.2016. године

_____

1. Ауторство - Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. Ауторство – некомерцијално. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. Ауторство - некомерцијално – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. Ауторство - некомерцијално – делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. Ауторство – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. Ауторство - делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.