Matematički fakultet
Univerzitet u Beogradu

Master rad

Rešavanje jednodimenzionog problema pakovanja kombinovanjem optimizacionih metoda

Rajačić Jasna

Beograd, septembar 2013

Tema rada:

Rešavanje jednodimenzionog problema pakovanja kombinovanjem optimizacionih metoda

Autor:

Jasna Rajačić

Članovi komisije:

Dr Dušan Tošić, *mentor*

Dr Zorica Stanimirović

Dr Miroslav Marić

Dr Vladimir Filipović

Sadržaj

Lista slika

Lista tabela

Lista algoritama

Predgovor

Koristim ovu priliku da se zahvalim svom mentoru, dr Dušanu Tošiću na savetima, sugestijama i podršci koju sam imala tokom izrade ovog rada.

Zahvaljujem se članovima komisije, profesorki Zorici Stanimirović i profesorima Miroslavu Mariću i Vladimiru Filipoviću koji su detaljno pročitali rad i svojim sugestijama doprineli njegovom finalnom uobličavanju.

I na kraju, zahvaljujem se svojoj porodici na strpljenju, razumevanju i podršci koju su mi pružali tokom svih ovih godina.

Uvod

Jednodimenzioni problem pakovanja u skladišta (Bin Packing Problem - BPP) se javlja u mnogim praktičnim situacijama, kao što su transport, problem pakovanja ranca, raspoređivanje posla na paralelnim mašinama, upravljanje kapitalom. Cilj je upakovati određen broj paketa različitih veličina u skladišta fiksnog kapaciteta, tako da se minimizuje broj upotrebljenih skladišta.

Garey i Johnson (1979) [12] su pokazali da ovaj problem pripada klasi NP-teških problema. Poznato je da egzaktne metode primenjene na BPP efikasno rešavaju problem malih do srednjih dimenzija, dok se problem velikih dimenzija uspešno rešava heuristikama.

Dalja istraživanja su pokazala da se problem efikasno rešava primenom metaheuristike zasnovane na evolutivnim algoritmima. Jedna od najefikasnijim metaheuristika je *Hibridni genetski algoritam grupisanja* (Hybrid Grouping Genetic Algorithm - HGGA) [8].

U ovom radu je predložena metaheuristika zasnovana na *optimizaciji pomoću mravlje kolonije* (Ant Colony Optimization - ACO) [16]. Ovo je nova metoda kombinatorne optimizacije. Prvi put je primenjena u aplikaciji koja služi za rešavanje *problema trgovačkog putnika* (Travelling Salesman Problem - TSP), Dorigo [19]. Zasnovana je na sposobnosti mrava da nađu pravi put do određene destinacije prateći trag koji ostavljaju za sobom – feromon. Pomoću aplikacije optimalno su rešavane male instance problema. Nakon publikacije rada, ideja je uspešno prilagođena i primenjena na razne probleme kombinatorne optimizacije. Ideja je korišćena i u ovom radu, uz neophodna prilagođavanja prirodi problema.

U prvom poglavlju rada predstavljen je problem pakovanja, njegova definicija i primene. Date su karakteristike do sada korišćenih metoda za rešavanje

problema i upoređeni dobijeni rezultati. Takođe, predstavljen je način rešavanja problema u ovom radu.

U drugom poglavlju su opisani algoritmi korišćeni u ovom radu. Pored opisa, razmatrane su dosadašnje primene ovih algoritama na BPP i druge probleme kombinatorne optimizacije. Posvećena je pažnja opštim parametrima algoritama, kao i njihovim optimalnim vrednostima.

U trećem poglavlju su algoritmi prilagođeni prirodi problema pakovanja. Objašnjena je informacija koju nosi heuristika, definisana fitnes funkcija, objašnjen način izgradnje rešenja. Metoda *lokalnog pretraživanja* (Local Search - LS) [25] je objašnjena i prilagođena BPP-u.

U četvrtom poglavlju su prikazani rezultati eksperimenata, kojim se želeo sagledati značaj predložene metode. Dato je objašnjenje o vrednostima mnogobrojnih parametara i pružen uvid u efikasnost izvršavanja algoritma u odnosu na razne vrednosti ovih parametara.

Na samom kraju rada nalaze se zaključci.

1

Jednodimenzioni problem pakovanja

1.1 Matematička definicija problema

Jednodimenzioni problem pakovanja se definiše na sledeći način (Garey i Johnson, 1979):

Neka je dat konačan skup L od n paketa koji su zadati svojim veličinama li, i=1,…,n, konstanta C (kapacitet skladišta) i broj skladišta N. Da li se paketi mogu spakovati u N ili manje skladišta bez prekoračenja kapaciteta?

Problem se formalno definiše na sledeći način:

Uvedimo binarne promenljive

$$x_{ij} = \begin{cases} 1, & ako\ je\ paket\ j\ u\ skladištu\ i \\ 0, & inače \end{cases}$$

$$y_i = \begin{cases} 1, & ako\ je\ skladište\ i\ upotrebljeno \\ 0, & inače \end{cases}$$

Cilj je minimizovati funkciju: $z = \sum_{i=1}^{N} y_i$

Uz uslove:

$$\sum_{j=1}^{n} l_j \, x_{ij} < C y_i \,, \qquad \forall \, i = 1,2,\dots,N \quad (2)$$

$$\sum_{i=1}^{N} x_{ij} = 1, \qquad \forall \; j = 1,2,\dots,n \quad (3)$$

$$y_i \in \{0,1\}, \; \forall i \quad (4)$$

$$x_{ij} \in \{0,1\}, \forall \, i,j \quad (5)$$

Uslovom (2) zahtevamo da nije prekoračen kapacitet skladišta. Uslov (3) ispunjava zahtev da se ma koji paket može naći u tačno jednom skladištu, dok uslovi (4) i (5) opisuju binarnu prirodu promenljivih y_i i x_{ij}.

1.2 Problemi grupisanja

BPP je deo porodice problema koji se temelji na podeli skupa paketa **U** na uzajamno disjunktne skupove **U***i*, *i =1,…,n,* tako da važi:

1. $\bigcup_{i=1}^{n} \boldsymbol{U}i = \boldsymbol{U}$
2. $\boldsymbol{U}i \cup \boldsymbol{U}j = \emptyset \,, \; i \neq j$

Problem se takođe može posmatrati i kao problem grupisanja. Tada bi cilj bio grupisati pakete skupa **U** u jednu ili više (a najviše **card(U)** – broj elemenata skupa U) grupa paketa, pri čemu će svaki paket pripadati najviše jednoj grupi.

Kod problema grupisanja, najčešće nisu dozvoljene sve mogućnosti grupisanja – rešenje problema mora zadovoljiti zadate uslove, inače se smatra nevažećim. Tako, često se dešava da neki paket nije moguće grupisati sa preostalim paketima. Cilj grupisanja je optimizacija funkcije cilja koja se definiše kao kolekcija svih dozvoljenih grupisanja. Slede dva primera iz porodice problema grupisanja, sa uslovima koje rešenje mora da ispuni (C je proizvoljna konstanta) i funkcijom cilja koja mora biti minimalna.

Iz priloženog se vidi da se problemi grupisanja karakterišu funkcijom cilja koja zavisi od kompozicije grupa. Dakle, akcenat je na grupi paketa, dok paket sam po sebi nema velikog značaja.

Problem	Uslovi	Funkcija cilja
BPP	Suma paketa u jednoj grupi < C	Broj grupa
Bojenje grafa	U ma kojoj grupi čvorovi nisu povezani	Broj grupa

Tabela 1.1: Problemi grupisanja.

1.3 Dosadašnje rešavanje BPP-a

Kako je poznato da BPP pripada klasi NP teških problema, to ne postoji algoritam koji daje optimalno rešenje u polinomskom vremenu [20]. Zato se pretraga za egzaktim algoritmom zamenjuje dizajnom aproksimativnih algoritama. Međutim, postoji pregršt egzaktnih algoritama koji se koriste za rešavanje ovog problema.

BPP može biti rešen korišćenjem algoritama celobrojnog programiranja, kao što je *metoda granjanja i ograničavanja* (Branch-and-Bound). Obzirom na veliki broj mogućih kombinacija smeštanja paketa u skladišta, vreme potrebno za izvršavanje programa je ogromno za veliko N.

Tradicionalne metode za rešavanje BPP podrazumevaju korišćenje brzih heuristika [12]. Postoji jednostavna heuristika koja brzo dolazi do skoro optimalnog rešenja, a bazirana je na metodi granjanja i ograničavanja (Martello i Toth). Ideja je sledeća: paketi seređaju u nerastućem redosledu i otvara se prvo skladište. Prolazeći kroz niz paketa, za svaki se proverava da li može stati u neko od praznih skladišta. Ako može, smešta se u to skladište; u suprotnom, otvara se novo. Ova heuristika je poznata pod nazivom *metoda prvog odgovarajućeg paketa* (First Fit Decreasing - FFD). Osim nje, heuristika koja pokazuje dobre rezultate je *metoda najboljeg odgovarajućeg paketa* (Best Fit Decreasing - BFD) [1]. Razlika između ova dva algoritma je u tome što se u metodi BFD paketi stavljaju u prvo najbolje popunjeno skladište u koje mogu stati. Ovo algoritam BFD čini jako komplikovanijim od FFD. *Metoda sledećeg odgovarajućeg paketa* (Next Fit Decreasing - NFD), kao i *metoda najgoreg odgovarajućeg paketa* (Worst Fit Decreasing - WFD) takođe pokazuju dobre rezultate. Rezultati ovih algoritama u najgorem slučaju su $\frac{11}{9}Opt + 4$, gde je *Opt* broj skladišta iskorišćenih u teorijskom optimumu. Kako ih je lako implementirati, oni se najčešće ugrađuju u evolutivne algoritme zarad poboljšanja njihove efikasnosti.

[1] Podrazumeva se da su paketi u metodama BFD, NFD, WFD poređani nerastuće.

Pored brzih heuristika, *metoda smanjivanja* koju su razvili Martello i Toth (Martello and Toth Reduction Procedure - MTP) [23] se takođe može koristiti za rešavanje BPP-a. Metoda je spora za instance velikih dimenzija, ali daje odlične rezultate. Ona se zasniva na dominaciji skladišta: ako su data dva skladišta $\mathbf{B}_1$ i $\mathbf{B}_2$, podskup paketa $\{i_1,...,i_l\}$ skladišta $\mathbf{B}_1$ i podela $\{P_1,...,P_l\}$ skladišta $\mathbf{B}_2$, i ako za svaki paket i_j postoji manja ili jednaka odgovarajuća particija P_j, onda kažemo da $\mathbf{B}_1$ dominira nad $\mathbf{B}_2$. Tada, rešenje koje sadrži skladište $\mathbf{B}_1$ neće imati veći broj skladišta od rešenja koje sadrži skladište $\mathbf{B}_2$. MTP metoda pokušava da pronađe skladišta koja dominiraju nad ostalim skladištima. Kada ih pronađe, problem se smanjuje tako što se iz inicijalnog skupa paketa sklanjaju paketi smešteni u dominantno skladište. Da bi se izbeglo izvršavanje algoritma u eksponencijalnom vremenu, u rešenje ulaze dominantna skladišta sa najviše tri paketa.

Tražeći efikasnije metode za rešavanje BPP, istraživači pažnju poklanjaju evolutivnim pristupima. Najuspešnijim od ovih pristupa se pokazao *Hibridni genetski algoritam grupisanja* (Hybrid Grouping Genetic Algorithm - HGGA). HGGA koristi *grupisanje* – genetski algoritam radi sa celim skladištima umesto sa pojedinačnim paketima. Operator ukrštanja uzima izabrana skladišta prvog roditelja i forsira drugog roditelja da ta ista skladišta „usvoji“. Ukoliko je neko skladište drugog roditelja u konfliktu sa usvojenim skladištem, njegovi paketi se oslobađaju. Metoda *lokalnog pretraživanja* se koristi da vrati oslobođene pakete u rešenje; slobodni paketi se zamenjuju sa onima koji nisu slobodni u cilju boljeg popunjavanja skladišta koja imaju mali broj paketa velike veličine. Paketi koji su oslobođeni a metoda lokalnog pretraživanja ih nije smestila ni u jedno skladište, u rešenje se vraćaju pomoću FFD heuristike, koja ih smešta u nova skladišta. Operator mutacije funkcioniše na sličan način: nekoliko slučajno izabranih skladišta je otvoreno, njihovi paketi oslobođeni, i nakon toga vraćeni u rešenje pomoću procedure lokalnog pretraživanja. HGGA je do sad najefikasnija metoda kojom je rešavan jednodimenzioni problem pakovanja.

Još jedan pristup rešavanju BPP-a su predložili Levine i Ducatelle (2004). On je zasnovan na *metodi optimizacije pomoću mravlje kolonije* (Ant Colony Optimization - ACO). Ova metoda se naziva *Hibridni Max-Min mravlji sistem* (Hybrid Max-Min Ant System - HACO). Izgradnja rešenja u ovoj metodi se zasniva na FFD heuristici, a trag feromona se gradi pomoću verovatnoće da će se paketi određenih veličina naći zajedno u nekom skladištu. Na kraju se rešenje poboljšava pomoću lokalnog pretraživanja, pri čemu je broj paketa koji se zamenjuju u ovoj metodi ograničen na dva. Rezultati pokazuju da ova metoda daje dobre rezultate, koji su konkurentni rezultatima HGGA.

1.4 Primene

Problem pakovanja pripada velikoj porodici problema sečenja i pakovanja (Cutting and Packing Problems). Opštu strukturu ove grupe problema opisao je Dyckhoff [12]. Ona je takva da uvek raspolažemo kombinacijom grupe malih (paketa) i grupe velikih objekata (skladišta). Cilj je spakovati male objekte po obrascu i dodeliti ih velikim objektima. Primeri problema koji prate ovu strukturu su *problem ranca* (knapsack problem) [23], *problem utovara prtljaga* (vehicle loading problem) [7], *problem deljenja posla na procesore* (the multiprocessor sheduling problem) [7], pa čak i *problem raspoređivanja kapitala* (the multi-period capital budgeting problem) [7].

U jednodimenzionom problemu pakovanja cilj je pakete zadate svojim veličinama spakovati u skladišta unapred zadatog kapaciteta, tako da je broj skladišta minimalan. Ovaj problem ima puno praktičnih primena:

- *Problem utovara prtljaga.* Cilj je minimizovati broj vozila koja prevoze pakete određenih veličina do zadatih destinacija; pri tom, cilj je i minimizovati troškove transporta tih vozila. Ako je broj iskorišćenih vozila u različitim rešenjima isti, uzimamo rešenje sa kraćom rutom. Funkcija cilja uključuje, kako ukupnu dužinu rute, tako i broj vozila, pa je neophodno zadovoljiti brojne uslove, kao što je količina paketa koju stavljamo u jedno vozilo i sl. Dužinu rute vozila možemo dobiti nekim algoritmom za rešavanje *problema trgovačkog putnika* (Travelling salesman problem - TSP) [11].
- *Sečenje kablova na određene dužine.* Ovaj problem se često javlja u metalnoj industriji. Takođe, može se "preformulisati", u zahtev za sečenjem ploča na manje ploče određene dimenzije. Cilj je minimizovati materijal koji otpada.
- *Problem deljenja posla na procesore.* Ideja je dodeliti poslove različitih dužina trajanja procesorima, tako da se njihov broj minimizuje. Jedna od varijanti ovog poblema je i *deljenje vremena za reklamiranje na televiziji na reklame određenih (različitih) dužina*. Aplikacije vezane za ovaj problem se mogu naći na internet stranicama *Američkog udruženja matematičara* (American Mathematical Society - AMS) [29], kao i *Nacionalnog Instituta standarda i tehnologija* (National Institute of Standards and Technology - NIST) [30].

Postoje druge varijante problema pakovanja. Najčešće razmatrana je *dvodimenzioni problem pakovanja*, u kojoj su paketi zadati dvema veličinama –

širinom i dužinom. Matematička definicija dvodimenzionog problema pakovanja može se naći u [14], [6]. Takođe, problem ima primene u mnogim industrijama:

- *Problem pakovanja na traci* (Strip packing Problem, [6]). Ovaj problem podrazumeva pakovanje dvodimenzionih paketa (u daljem tekstu pravougaonika), na traci fikne širine u cilju minimizovanja visine iskorišćene trake. Pravougaonici se ne smeju preklapati. Moguće je dozvoliti rotaciju pravougaonika za 90º.
- *Problem nepravilnog sečenja materijala* (Irregular cutting stock Problem, [6]). Problem se javlja u tekstilnoj industriji, gde je cilj minimizovati utrošak materijala pri sečenju šnitova. Šabloni za sečenje se prave da bi što manje materijala otpadalo.

Trodimenzioni problem pakovanja je još jedna varijanta ovog problema. Njegova definicija i opširnije razmatranje mogu se naći u [8], [9], [21], [22].

Bischoff i Wäscher [7] u svom radu daju brojne razloge zašto su problemi pakovanja i sečenja zanimljiva tema za istraživanje. Prvo, ova grupa problema ima višestruke primene u mnogim industrijama, kao što je čelična, papirna ili industrija stakla. Kao što je rečeno u [8], postoje mnogi industrijski problemi koji izgledaju drugačije, ali se u suštini svode na ovu grupu problema. U ove probleme spadaju *VLSI dizajn, raspoređivanje kapitala, planiranje procesorskog vremena*. Drugo, ovi problemi su raznovrsni i u stvarnom životu – iako se čini da problemi pakovanja i sečenja imaju istu strukturu, može biti interesantnih razlika između njih. Poslednji je razlog – kompleksnost ove grupe problema. Većina problema ove grupe je NP kompletna. U ovu grupu spada jednodimenzioni problem pakovanja, koji je tema istraživanja ovog rada. Iz tog razloga, egzaktna rešenja su moguća samo za instance malih dimenzija; realni problemi se rešavaju korišćenjem heuristika, a potraga za što boljom heuristikom ostaje glavni problem istraživanja u ovom radu.

2

O algoritmima korišćenim u rešavanju problema pakovanja

2.1 Heuristike

Reč ***heuristika*** potiče od grčke reči ***heurisko*** i prevedena na srpski jezik ima značenje ***otkriti*** ili ***naći*** [3]. Heuristika predstavlja kompromis između potrebe da algoritam bude jednostavan i želje da isti razlikuje dobre i loše odluke.

Posmatrajmo listu objekata $\mathbb{L}$. Ukoliko govorimo o problemu pakovanja, heuristika će uzimati jedan po jedan paket i pokušavaće da ih spakuje u određeno skladište. Cilj joj je da dođe do optimalnog rešenja, bez garancije da će to uraditi.

Poznato je da je BPP NP težak problem. Veliki broj prethodnih istraživanja ovog problema podrazumevao je konstrukcije heuristika i njihova teorijska istraživanja. One koje se koriste za rešavanje problema pakovanja drže se ustaljenih pravila pakovanja – paketi se pakuju takvim redosledom da se dobije dopustivo (ne uvek i optimalno) rešenje u najkraćem mogućem vremenu. U ovom radu su razmatrane metode koje smeštaju pakete opadajuće – polazi se od paketa maksimalne veličine. One su poznate kao FFD, NFD, WFD, BFD.

2.1.1 Metoda prvog odgovarajućeg paketa - FFD

Ova metoda se najčešće koristi pri rešavanju jednodimenzionog problema pakovanja. Paketi seređaju u nerastućem redosledu po njihovoj veličini i otvara se prvo skladište. Svaki paket se smešta u prvo skladište u koje može stati, bez prekoračenja uslova kapaciteta. Metoda staje kada su svi paketi spakovani u skladišta.

Algoritam 2.1: Pseudo kod – FFD

```
Poređaj pakete nerastuće;
Ponavljaj:
        For svaki paket do
                For svako skladište do
                    If paket može stati u skladište
                       Spakuj paket;
                       Pređi na sledeći paket;
                    End if
                End for
            If paket ne može stati u skladište
               Otvori novo skladište;
               Spakuj paket u njega:
            End if
        End for
Sve dok: Nisu svi paketi spakovani;
Izlaz:   Paketi spakovani u skladišta.
```

Johnson je pokazao u [4] da algoritam u najgorem slučaju ima rezultat $\frac{9}{11}Opt + 4$, gde je *Opt* broj skladišta u teorijskom optimumu. Kasnija istraživanja pokazuju da je novi najgori slučaj $\frac{11}{9}Opt + 1$ [27].

Vreme izvršavanja FFD algoritma je $O(n\ logn)$.

2.1.2 Metoda najboljeg odgovarajućeg paketa – BFD

U metodi BFD se paketi takođeređaju u nerastućem poretku. Paketi se smeštaju u ono skladište u koje mogu stati, a koje ima najmanje preostalog slobodnog prostora. Ukoliko takvog nema, novo će biti otvoreno. Metoda staje kada su svi paketi raspoređeni.

Kao i prethodna, ova metoda u najgorem slučaju ima rezultat $\frac{11}{9}Opt + 4$. Njeno vreme izvršavanja je $O(n\ logn)$.

Algoritam 2.2: Pseudo kod – BFD

```
Poređaj pakete nerastuće;
Ponavljaj:
        For svaki paket do
                For svako skladište do
                    If paket može stati u otvoreno skladište
                            Spakuj paket;
                            Izračunaj preostali slobodan prostor u skladištu;
                    End if
                End for
            If paket ne može stati u skladište
                Otvori novo skladište;
                Stavi paket u njega:
            End if
        End for
Sve dok: Nisu svi paketi raspoređeni;
Izlaz:   Paketi raspoređeni u skladišta.
```

2.1.3 Metoda sledećeg odgovarajućeg paketa – NFD

Paketi se, kao u prethodnim metodama, ređaju u nerastućem poretku. Zatim se otvara prvo skladište. Paketi se prvo smeštaju u otvoreno skladište. Za svaki sledeći paket se proverava da li može stati u trenutno otvoreno skladište. Ukoliko paket ne može stati u to skladište, otvara se novo. Kada su svi paketi raspoređeni u skladišta, metoda staje.

Algoritam 2.3: Pseudo kod – NFD

```
Poređaj pakete nerastuće;
Ponavljaj:
        For svaki paket do
            If paket može stati u trenutno otvoreno skladište
                    Stavi paket;
            Else
                    Otvori novo skladište i smesti paket u njega;
                    Zatvori prethodno skladište;
        End for
Sve dok: Nisu svi paketi raspoređeni;
Izlaz:   Paketi raspoređeni u skladišta.
```

Metoda u najgorem slučaju ima rezultat $2Opt$. Vreme izvršavanja je $O(n)$.

2.1.4 Metoda najgoreg odgovarajućeg paketa – WFD

Paketi se ređaju u nerastućem poretku. Smeštaju se u ono skladište u koje mogu stati, a koje ima najviše preostalog slobodnog prostora. Ukoliko takvog nema, novo će biti otvoreno. Metoda staje kada su svi paketi raspoređeni.

Algoritam 2.4: Pseudo kod – WFD

```
Poređaj pakete nerastuće;
Ponavljaj:
        For svaki paket do
                For svako skladište do
                Izračunaj preostali prostor u skladištu;
                   If skladište ima max slobodnog prostora
                     i paket može stati u njega
                            Stavi paket;
                            Izračunaj preostali slobodan prostor u skladištu;
                   End if
                End for
             If paket ne može stati u skladište
                Otvori novo skladište;
                Stavi paket u njega:
             End if
        End for
Sve dok: Nisu svi paketi raspoređeni;
Izlaz:   Paketi raspoređeni u skladišta.
```

Najgori poznati rezultat metode je 2*Opt*. Vreme izvršavanja je $O(n \log n)$.

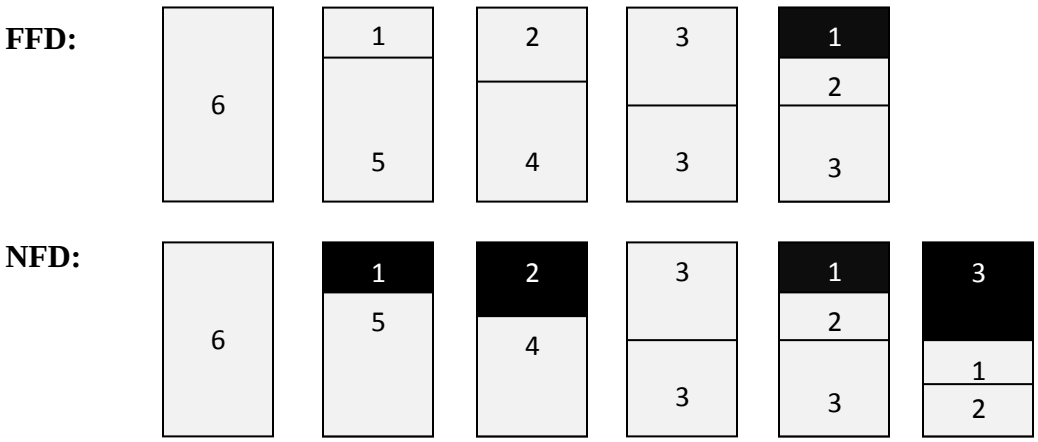
2.1.5 Primer upotrebe heuristika

Neka je dat niz paketa $\mathbb{L} = \{4,1,2,5,3,2,3,6,3\}$ i neka su skladišta kapaciteta 6. Paketi se ređaju u nerastućem poretku $\mathbb{L} = \{6,5,4,3,3,3,2,2,1\}$. Nakon toga, različite heuristike smeštaju pakete na različite načine. Od značaja je napomenuti da je teorijski optimum 5.

Primerom je pokazana razlika između gore objašnjenih heuristika. FFD uzima prvi paket iz liste $\mathbb{L}$ i smešta ga u prvo skladište u koje može stati. U prvoj iteraciji, otvoriće prvo skladište i smestiće ga u njega. Za svaki sledeći paket liste $\mathbb{L}$, FFD će proveriti da li može stati u neko od već otvorenih skladišta, a koje nije popunjeno. Tako će za pakete veličine 5, 4, 3 otvoriti nova skladišta. Kada dođe do drugog paketa veličine 3, smestiće ga u već otvoreno skladište koje ima dovoljno prostora da ga primi (skladište u koje je smešten prvi paket veličine 3). Slično,

nastaviće postupak za pakete ostalih veličina. Metoda NFD se ne vraća kroz niz skladišta tražeći ono u koje paket može stati, već ga pokušava smestiti ili u trenutno otvoreno, ili otvara sledeće i smešta paket u to skladište.

Metode BFD i WFD su za uzeti primer imale isti rezultat kao FFD, pa nisu razmatrane u primeru.



Slika 2.1: Primer raspoređivanja paketa.

2.2 Metaheuristike

Prefiks ***meta*** na grčkom jeziku znači ***iznad***, što ukazuje da metaheuristike kao metodi kombinatorne optimizacije rade na višem nivou od heuristika. Mogu biti predstavljene kao algoritmi koji su "unapređene heuristike u cilju što efikasnijeg i efektivnijeg pretraživanja prostora rešenja". Reč je prvi put upotrebio Glover [10], i od tad je prihvaćena kao naziv ove klase algoritama.

Pri rešavanju problema pakovanja, heuristike rade sa pojedinačnim paketom, dok metaheuristike najčešće rade sa kolekcijom paketa i njoj kao takvoj dodeljuju neku poziciju. Takođe, često koriste heuristike kao procedure za dekodiranje. Na primer, metaheuristika će naći zgodno uređenje paketa, a onda će neka heuristika ovako uređene pakete smeštati u skladišta.

Postoje mnogobrojne vrste metaheuristika, a neke od njih su inspirisane ponašanjem određenih jedinki u prirodi. U ovu grupu spadaju *metoda optimizacije pomoću mravlje kolonije*, *metoda optimizacije pomoću roja pčela* (Bee Colony Optimization) [5], *genetski algoritmi* (Genetic Algorithm - GA). Ove metaheuristike[1]

[1] Neke metaheurisike, kao što je ACO, polaze od parcijalnog ili praznog rešenja.

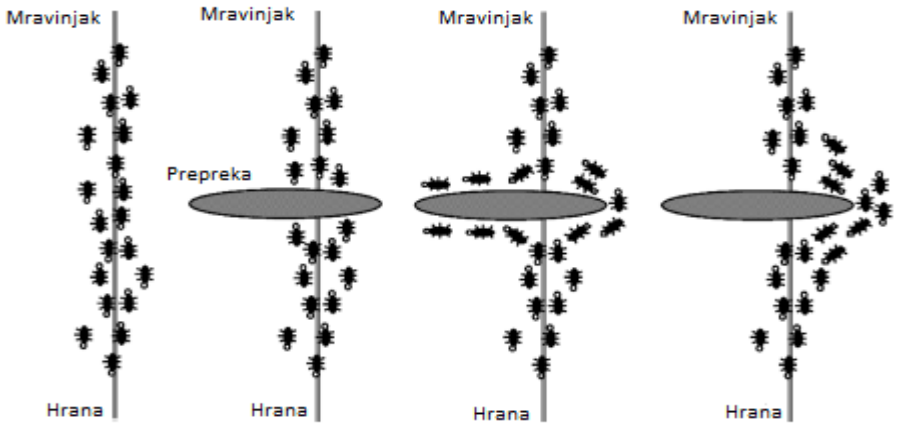
polaze od nekog inicijalnog rešenja. Iterativno zamenjuju novu generaciju rešenja prethodnom, sve dok kriterijum zaustavljanja nije zadovoljen. Razlikuju se dve podklase ove vrste algoritama – sa pamćenjem informacija o prethodnoj generaciji rešenja, ili bez pamćenja. U prvom slučaju, informacija o staroj populaciji se može iskoristiti da bi se generisala nova generacija.

2.2.1 Optimizacija pomoću mravlje kolonije – ACO

Osnovna ideja ovog algoritma je potekla iz društvenog ponašanja kolonije mrava u potrazi za hranom. Algoritam je predložio Dorigo [15]. ACO je prilagođavan različitim problemima kombinatorne optimizacije i pokazuje uspeh u rešavanju istih [18].

Interesovanje za primenom ovog algoritma se javilo jer kolonija mrava na jednostavan način sprovodi kompleksne zadatke, kao što je prenošenje hrane, ili pronalazenje najkraćeg puta do izvora hrane. Algoritam imitira kolektivno ponašanje mrava i njihov mehanizam komunikacije. Slika 2.1 ilustruje eksperiment rađen nad kolonijom mrava.

Kolonija ima pristup hrani preko dva puta koja vode do mravinjaka.Tokom puta koji prelaze, mravi za sobom ostavljaju trag – *feromon*, čija je uloga da vodi ostale mrave tako što će pratiti taj trag. Što je veća količina feromona na putu, veća je verovatnoća da će mravi izabrati taj put da se njime kreću do mravinjaka, odnosno izvora hrane. Određeni mrav bira put kojim će se kretati u zavisnosti od jačine feromona koji se nalazi na tom putu.



Slika 2.2: Kolonija mrava koja traži optimalan put kretajući se od izvora hrane do mravinjaka.

Proces isparavanja feromona naziva se proces evaporacije. Učešće mrava u procesu evaporacije zavisi od količine hrane koja je ostala na izvoru hrane, a koju mrav treba preneti. Sa slike 2.2 se vidi da, kada dođu do prepreke, jednaka je verovatnoća da će izabrati levi, odnosno desni put. Međutim, kako je levi put kraći, manje je vremena potrebno da se mrav vrati do izvora hrane, što dovodi do toga da će na levom putu ostati jači trag feromona. Što više mrava prođe tim putem, viši će biti nivo feromona, pa će se na kraju čitava kolonija kretati tim putem.

Donji algoritam predstavlja pseudo-kod ACO metode. Prvo se inicijalizuje informacija o feromonu. Dva osnovna koraka algoritma su: *konstrukcija rešenja* i *pojačavanje feromona*, koji se iterativno ponavljaju dok najbolje rešenje nije nađeno.

Algoritam 2.5: Pseudo kod – ACO

Inicijalizuj trag feromona;
Ponavljaj:
    For *svaki mrav* **do**
    -Konstruiši rešenja koristeći trag feromona;
    -Pojačaj trag feromona;
        Evaporacija;
        Pojačavanje;
Sve dok: *Kriterijum zaustavljanja zadovoljen;*
Izlaz: *Najbolje nađeno rešenje.*

Konstrukcija rešenja. Rešenje gradimo pomoću informacije iz heuristike i traga feromona. Mrave posmatramo kao proceduru koja gradi rešenje koristeći verovatnoću tako što dodaje komponente rešenja parcijalnim rešenjima dok ne napravi konačno. Ova informacija u sebi objedinjuje:

1. *Trag feromona*. Feromon pamti karakteristike dobro izgrađenog rešenja, a to će se koristiti pri izgrađivanju novih rešenja mrava. Feromon se menja dinamički tokom pretrage za rešenjem i na njemu se ogleda stečeno znanje. Predstavlja memoriju celokupnog procesa potrage svih mrava za rešenjem.
2. *Informacija iz heuristike.* Ova informacija pomaže mravima tako što im nagoveštava kako da odlučuju tokom izgradnje rešenja. Pitanje izbora heuristike je od velikog značaja za performance celokupnog algoritma.

Pojačavanje feromona. Feromon se pojačava korišćenjem već izgrađenih rešenja, kroz faze evaporacije i pojačavanja.

1. *Faza evaporacije.* U ovoj fazi feromon opada automatski. Svaka vrednost feromona τ_{ij} se smanjuje konstantom evaporacije $\boldsymbol{\rho} \in]0,1]$ koja se bira proizvoljno u ovom intervalu.

   $$\tau_{ij} = (1 - \rho) * \tau_{ij}\ ,\ \forall i,j$$

   Cilj evaporacije je da se izbegne prevremena konvergencija ka "dobrim" rešenjima i da se podstakne raznolikost u pretraživanju.
2. *Faza pojačavanja feromona*. U ovoj fazi se pojačava feromon pomoću pronađenog rešenja. Tri različite strategije mogu biti primenjene:
   - *Onlajn korak-po-korak pojačavanje*. Feromon pojačava mrav na svakom koraku izgradnje rešenja [17].
   - *Onlajn usporeno pojačavanje*. Feromon se pojačava kada jedan mrav izgradi svoje rešenje [26].
   - *Oflajn pojačavanje*. Feromon se pojačava jednom, kada svi mravi izgrade celokupno rešenje. Ovo je najpopularniji pristup, gde mogu biti primenjene različite strategije [1]:
     a) Pojačavanje prema kvalitetu rešenja. Feromon τ se pojačava pomoću najboljeg rešenja (ili *k* najboljih rešenja, gde je *k* manje od broja mrava). Vrednost koju dodajemo feromonu zavisi od kvaliteta najboljeg rešenja. Na primer, svakom paketu koji pripada najboljem rešenju $\boldsymbol{\pi^o}$, dodata je proizvoljno izabrana pozitivna konstanta $\boldsymbol{\Delta}$.

        $$\tau_{i\pi^o(i)} = \tau_{i\pi^o(i)} + \boldsymbol{\Delta}\ ,\ \forall i \in [1,n]$$

     b) Pojačavanje feromona po rangu. Rešenja najboljih *k* mrava će se pojačati vrednostima koje zavise od ranga njihovih rešenja [2].
     c) Pojačavanje feromona najgoreg rešenja. Pojačaće se feromon pomoću mrava koji ima najgore rešenje.
     d) Elitno pojačavanje feromona. Najbolje rešenje nađeno do tada će pojačati feromon da bi se podstakla što intenzivnija pretraga.

Centralna tema pri dizajniranju ACO algoritma je odluka o sledećem:

- **Informacija o feromonu**. Ovo je najbitnija komponenta algoritma. Potrebno je definisati parametre traga feromona; oni, pak, treba da oslikaju trenutno stanje rešenja za dati problem.
- **Izgradnja rešenja**. U ovom delu, bitno je voditi računa o odabiru heuristike. ACO je lako prilagoditi problemima za koje već postoje efikasni proždrljivi algoritmi (Greedy Algorithm).
- **Pojačavanje feromona**. Definišemo strategiju za pojačavanje feromona.

Teorijska analiza konvergencije ACO algoritma se može naći u [28].

2.2.2 Primena ACO algoritma: TSP i drugi problemi.

Prvi ACO algoritam, *Mravlji sistem* (Ant System - AS), razvijen je kao aplikacija za rešavanje TSP [19]. AS je postao veoma popularan nakon publikacije. Postoji dosta algoritama koji se zasnivaju na njemu, a koji rešavaju razne probleme kombinatorne optimizacije. Da bi se ACO algoritam prilagodio TSP-u, potrebno je predefinisati feromon i proceduru izgradnje rešenja.

Algoritam 2.6: Pseudo kod – ACO primenjen na TSP

Inicijalizuj trag feromona;
Ponavljaj:
For *svaki mrav* **do**
-Konstruiši rešenje koristeći trag feromona:
S = {1,2,...,n} */*gradovi koje mrav može izabrati*/*
Nasumično izaberi početni grad i;
Ponavljaj:
Izaberi novi grad j sa verovatnoćom $p_{ij} = \frac{\tau_{ij} \times \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik} \times \eta_{ik}^{\beta}}$
S = S – {j}; i = j;
Sve dok: *S = {};*
Kraj For *petlje*
-Pojačaj trag feromona:
For *i,j* ∈ *[1,n]* **Do**
Evaporacija;
For *i* ∈ *[1,n]* **Do**
Pojačaj;
Sve dok: *Kriterijum zaustavljanja zadovoljen;*
Izlaz: *Najbolje nađeno rešenje.*

Trag feromona. U ovom pristupu, trag feromona predstavlja vezu između gradova *i* i *j*. Ova informacija se predstavlja u obliku matrice dimenzije *n* x *n*, gde svaki element τ_{ij} predstavlja zahtev da destinacija (*i*, *j*) bude u turi. Matrica feromona se, u opštem slučaju, inicijalizuje istim vrednostima. Tokom pretrage, feromon se pojačava.

Konstrukcija rešenja. Svaki mrav za početnu destinaciju ima slučajno izabran grad, i rešenje gradi idući iz jednog u drugi, dok ih sve ne obiđe. Verovatnoća da će mrav, koji se nalazi u gradu *i*, izabrati da ide u grad *j* je data sa:

$$p_{ij} = \frac{\tau_{ij} \times \mu_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik} \times \mu_{ik}^{\beta}}, \forall j \in S$$

U ovoj jednakosti, τ_{ij} je feromon između grada *i* i *j*, a μ_{ij} je vrednost heuristike koja vodi mrava. Kako definicija heuristike zavisi od problema koji rešavamo, ovde je

$$\mu_{ij} = \frac{1}{d_{ij}}$$

gde d_{ij} predstavlja rastojanje između gradova *i* i *j*. Što je viša vredost parametra μ_{ij}, to je kraće rastojanje između izabranih gradova. Parametar β definiše važnost informacije koju nosi heuristika. Ako je β = 0, samo vrednost feromona utiče na pretragu.

Pojačavanje feromona. Kada svi mravi obiđu turu, feromon se pojačava. Pojačavanje je proporcionalno kvalitetu ture π*:*

$$\tau_{i\pi(i)} = \tau_{i\pi(i)} + \Delta, \forall i \in [1, n]$$

gde je $\Delta = {}^{1}/_{f(\pi)}$, a $f(\pi)$ dužina ture π. Tako će dobre ture "isploviti" jer mravi sarađuju na ostavljanju feromona. Neizostavan je i **proces evaporacije feromona**. Ova faza je ista kao u klasičnom ACO algoritmu.

$$\tau_{ij} = (1 - \rho) * \tau_{ij}, \forall i, j \in [1, n]$$

AS se pokazao kao dobar algoritam kod relativno malih instanci za TSP, ali nedovoljno dobar za velike instance. Kasnije, pojavljuju se poboljšanja ovog algoritma koja dobro rade sa instancama velikih dimenzija. Primeri ovih poboljšanja su Sistem mravlje kolonija (Ant Colony System) [17], i Min-Max Mravlji sistem (MIN-MAX Ant System) [24].

ACO je prilagođen mnogim problemima kombinatorne optimizacije, kao što su *Kvadratni problem dodeljivanja* (Quadratic Assignment Problem - QAP), *Problemi rasporeda* (Scheduling Problems), *Problem usmeravanja vozila* (Vehicle Routing Problem – VRP), *Problem bojenja grafa* (The Graph Coloring Problem), *Problem pakovanja ranca* (The Multiple Knapsack Problem).

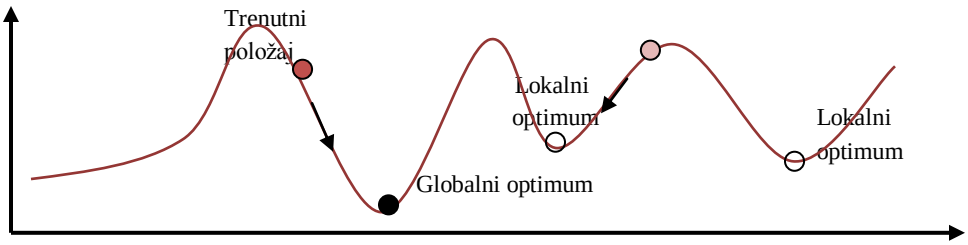
U radu sa ACO algoritmom, kao važan korak javlja se inicijalizacija mnogobrojnih parametara. Tabelom 2.1 su prikazani centralni parametri osnovne verzije ACO algoritma. Parametar β je vrlo osetljiv; njegova optimalna vrednost umnogome zavisi od problema koji rešavamo. Broj mrava nije kritičan parametar – njegova vrednost zavisi od zahteva problema za resursima.

Parametar	*Uloga*	*Praktična vrednost*
β	*Uticaj heuristike*	-
ρ	*Parametar evaporacije*	[0.01, 0.2]
κ	*Broj mrava*	[10, 50]

Tabela 2.1 Vrednosti centralnih parametara ACO metaheuristike

2.2.3 Lokalno pretraživanje

Lokalno pretraživanje je verovatno najstarija i najjednostavnija metaheuristika. Počinje od zadatog inicijalnog rešenja i u svakoj iteraciji menja trenutno rešenje susedstvom koje poboljšava funkciju cilja. Pretraga staje kada svi kandidati jednog susedstva nisu bolji od trenutnog rešenja – lokalni optimum je dostignut.



Slika 2.3 Metoda lokalnog pretraživanja

Za velike prostore pretraživanja, rešenja kandidati mogu biti podskupovi tog prostora. Uzimanje podskupova prostora pretraživanja ubrzava pretragu. Dopunske verzije lokalnog pretraživanja se karakterišu po načinu dobijanja rešenja (determinističke ili stohastičke) i načinu pretrage prostora rešenja.

Postoji više načina na koje se pretraga prostora rešenja može vršiti:

- **Najbolje poboljšanje.** Bira se susedstvo koje najbolje poboljšava funkciju cilja. Pretraga se vrši deterministički; pretražuje se celi prostor rešenja dok se ne dodje do traženog susedstva. Za veliki prostor pretrage, metoda je spora.
- **Prvo poboljšanje.** Uzima se prvo susedstvo koje je bolje od trenutnog rešenja i zamenjuje se starim. Podrazumeva parcijalnu evaluaciju prostora. U najgorem slučaju, pretražiće se celi prostor rešenja.
- **Nasumičan izbor.** Slučajno se bira rešenje među svim onim koja su bolja od trenutnog.

Kompromis između kvalitetnog rešenja i kratkog vremena pretraživanja se može postići upotrebom strategije prvog poboljšanja kada je početno rešenje slučajno izabrano, a strategije najboljeg poboljšanja, kada to nije slučaj. U mnogim

slučajevima u praksi pokazalo se da strategija pvog poboljšanja dovodi do istih rezultata kao i strategija najboljeg poboljšanja, ali za manje vremena.

Algoritam 2.7 predstavlja pseudo-kod lokalnog pretraživanja. Prvo se izabere početno rešenje. Nakon toga, iterativno se ispituje susedstvo trenutnog rešenja. Ukoliko neka tačka susedstva poboljšava trenutno rešenje, ono se tom tačkom zamenjuje.

Metoda lokalnog pretraživanja je veoma laka za implementaciju i daje sasvim zadovoljavajuće rezultate za kratko vreme. Zato se često primenjuje u praksi na različitim problemima optimizacije. Mana ove metode je konvergencija prema *lokalnom optimumu*. Detaljnije, algoritam je jako osetljiv prema početnom rešenju – velike razlike u kvalitetu rešenja mogu biti dobijene za neke probleme.

Algoritam 2.7: Pseudo kod – lokalno pretraživanje

Inicijalizuj početno rešenje:
- Izaberi neko rešenje iz prostora rešenja.
Ponavljaj:
Za *svaki korak* **uradi**
-Izaberi okolinu trenutnog rešenja;
-U njoj nađi sledeće rešenje prema nekom kriterijumu izbora;
-Za izabrano rešenje proveri da li je bolje od trenutnog;
-Ako jeste, zameni rešenje nađenim.
Sve dok: *Kriterijum zaustavljanja zadovoljen;*
Izlaz: *Najbolje nađeno rešenje.*

Iako je složenost ove metode u praksi prihvatljiva, ona u najgorem slučaju može biti eksponencijalna. Lokalno pretraživanje radi dobro ako u prostoru pretraživanja nema puno lokalnih optimuma ili ako je sličnost između raznih lokalnih optimuma velika.

Radi izbegavanja glavnog nedostatka ove metode, *konvergenciji ka lokalnom optimumu*, predloženo je korišćenje raznih alternativnih algoritama. Oni postaju popularni oko 1980. godine. Razlikuju se četiri pristupa:

1. Promena inicijalnog rešenja.
2. Prihvatanje susedstva koji ne poboljšavaju rešenje.
3. Menjanje susedstva.
4. Menjanje funkcije cilja ili ulaznih vrednosti problema.

3

Prilagođavanje algoritama jednodimenzionom problemu pakovanja

U ovom poglavlju je opisano kako se ACO metaheuristika prilagođava jednodimenzionom problemu pakovanja. Ideja je da mravi grade i dokazuju efikasnost svojih rešenja za zadatu instancu problema kombinatorne optimizacije. Iz generacije u generaciju, u memoriji se čuvaju informacije o uspešnim mravima. Tako se grade rešenja, sve dok se ne dođe do najboljeg.

3.1 Trag feromona

Kvalitet ACO aplikacije dosta zavisi od definiije traga feromona [18]. Od suštinskog značaja je izabrati definiciju koja odgovara prirodi problema. Kao što je rečeno, BPP je problem *grupisanja*. Ideja je da grupišemo pakete, što se razlikuje od mnogih problema kojima je ACO do sada bio prilagođavan. TSP je problem *poretka*: cilj je urediti gradove u određenom poretku. Trag feromona je za TSP problem imao sledeću definiciju: pristrasnost da grad *j* bude posećen nakon grada *i*.

Za BPP izabrana je sledeća definicija: $\tau(i,j)$ je pristrasnost da se paketi veličina i i j nađu u istom skladištu. Činjenica je da možemo imati više paketa veličine *i (j)*.

Ovaj pristup nam daje dve prednosti:

- Matrica feromona je kompaktna.
- Matrica čuva informacije o "dobrim" pakovanjima pojačavajući veze između veličina. Kada se pronađe dobar šablon, moći će da se koristi više puta tokom rešavanja problema.

3.2 Heuristika

Još jedna bitna stavka u prilagođavanju ACO algoritma je odabir heuristike koja će u kombinaciji sa tragom feromona graditi rešenja. U sekciji 2.1 su predstavljene četri heuristike koje su se dobro pokazale kod dosadašnjeg rešavanja BPP-a.

U ovom radu će biti povučena paralela između ACO aplikacije koja vrednost informacije iz heuristike dobija primenom FFD, NFD, BFD, WFD respektivno. Biće upoređene performanse aplikacije u odnosu na ove četiri metode.

Metode FFD i NFD rade na sledeći način: punimo skladište za skladištem tako što proveravamo da li sledeći paket iz niza paketa može stati u njega. Ukoliko može, smeštamo ga; ukoliko ne, otvaramo novo skladište (u slučaju NFD, zatvaramo staro). Metode BFD i WFD rade sledeće: punimo skladište za skladištem tako što proveravamo da li je preostali prostor trenutnog skladišta minimalan (odnosno maksimalan) u odnosu na ostala skladišta. Ako paket može stati u to skladište, smeštamo ga, a ako ne, otvaramo novo skladište.

U trenutku kada biramo paket koji ćemo smestiti u skladište, uvek ćemo izabrati paket maksimalne veličine iz niza preostalih paketa. Ovo nam pokazuje da je izbor paketa direktno povezan sa njegovom veličinom. Drugim rečima, uvek će pre biti izabran veliki paket sprem malog. Zati uzimamo da je vrednost heuristike jednaka veličini paketa, odnosno $\mu(j) = j$.

3.3 Izgradnja rešenja

Gore definisani trag feromona i heuristike sada se koriste od strane mrava koji grade rešenja. Svaki mrav počinje sa praznim skladištem i skupom paketa koji treba da smesti u skladišta. On konstruiše rešenje tako što puni skladišta dok svi paketi nisu spakovani. U slušaju da nijedan od preostalih paketa ne može stati u trenutno otvoreno skladište, ono se zatvara i otvara se novo.

Koji paket smestiti u otvoreno skladište? Odluka o ovome se zasniva na informaciji iz heuristike, ali i na informaciji iz feromona. Verovatnoća da će mrav k izabrati paket j da smesti u trenutno otvoreno skladište b data je sa (1) [18]:

$$p_k(s,b,j) = \begin{cases} \dfrac{[\tau_b(j)] \times [\mu(j)]^\beta}{\sum_{g \in J_k(s,b)} [\tau_b(g)] \times [\mu(g)]^\beta} & , ako\ j \in J_k(s,b) \\ 0 & , inače \end{cases} \qquad (1)$$

U ovoj jednakosti, $J_k(s,b)$ je skup paketa koji se mogu smestiti u trenutno otvoreno skladište. To su paketi koji su ostali nakon parcijalnog rešenja s, a koji mogu stati u skladište b. $\mu(j)$ je veličina, odnosno težina paketa j.

Vrednost feromona $\tau_b(j)$ paketa j u skladištu b je data sa (2). Predstavlja sumu svih vrednosti feromona između paketa i i j, a koji se već nalaze u skladištu b, podeljena sumom veličina svih paketa skladišta b. Ako je skladište prazno, vrednost feromona je predefinisana i iznosi 1.

$$\tau_b(j) = \begin{cases} \dfrac{\sum_{i \in b} \tau(i,j)}{|b|} & , ako\ je\ b \neq \emptyset \\ 1 & , \ inače \end{cases} \qquad (2)$$

3.4 Pojačavanje feromona

Pojačavanje feromona u ACO algoritmu se zasniva na pristupu Stützle i Hoos Max-Min AS (videti Sekciju 2.2.2) [24]. Ovaj pristup je lak za razumevanje i implementaciju i daje veoma dobre rezultate.

Ideja je da se samo najboljem mravu dozvoli ostavljanje feromona nakon svake iteracije. Postupak pojačavanja feromona je prilagođen BPP-u, jer u ovom problemu nemamo jedinstvene veličine paketa, već se paketi i i j mogu više puta naći zajedno u najboljem rešenju. Feromon se pojačava svaki put kada se paketi i i j nađu

zajedno. U (3), parametar m označava koliko su se puta paketi i i j našli zajedno u najboljem rešenju s_{best}.

$$\tau(i,j) = (1-\rho) \times \tau(i,j) + \ m \times f(s_{best}) \qquad (3)$$

Korišćenje samo najboljeg mrava za pojačavanje čini pretragu vrlo agresivnom. Biće podržane kombinacije skladišta koje se često javljaju u dobrim rešenjima.

Postoji nekoliko načina da se balansira između eksploatacije i pretraživanja. Jedan od načina je odabir između najboljeg mrava u iteraciji s_{Ib} i globalno najboljeg mrava s_{Gb}. Ako koristimo s_{Gb}, pojačavamo eksploataciju. Najpogodnije je uvesti parametar γ koji nagoveštava koliko čekamo u naizmeničnom smenjivanju mrava, dok ne upotrebimo s_{Gb} ponovo.

3.5 Funkcija prilagođavanja

Funkcija prilagođavanja je pokazatelj kvaliteta rešenja. Kod problema minimizacije, kakav je BPP, ona je najčešće jednaka inverzu vrednosti funkcije cilja datog rešenja. Trivijalan slučaj je kada je ona jednaka inverzu sume upotrebljenih skladišta. Međutim, ovo dovodi do neželjenih rezultata jer je često moguće napraviti mnogo kombinacija pakovanja sa samo jednim skladištem više nego u optimalnom rešenju. Ako svako takvo rešenje dobije istu vrednost funkcije prilagođavanja, ne postoji način da se algoritam vodi ka optimumu.

Falkenauer i Delchambre [9] predlažu funkciju prilagođavanja u kojoj je maksimizacija prosečne iskorišćenosti skladišta (što odgovara minimizaciji broja iskorišćenih skladišta) poboljšana tako što se uzimaju u obzir različito iskorišćena skladišta (manje odnosno više popunjena od proseka). Ovo je postignuto dodavanjem parametra z koji pomaže raznolikost između rešenja sa jednako popunjenim skladištima i rešenja kod kojih popunjenost skladišta varira – imamo skoro puna skladišta i ona sa niskim stepenom popunjenosti. Jednakost (4) je vrednost funkcije prilagođavanja rešenja s:

$$f(s) = \frac{\sum_{i=1}^{N}(F_i/C)^z}{N} \qquad (4)$$

U ovoj jednakosti N je broj skladišta, F_i ukupan sadržaj skladišta i, a C je kapacitet skladišta. Parametar z nam govori koliki akcenat stavljamo na popunjavanje skladišta. Ako je z = 1, dobijamo trivijalnu funkciju prilagođavanja koja je jednaka inverzu broja upotrebljenih skladišta. Kako se z povećava, tako rešenja sa većim

razlikama u popunjenosti skladišta postaju poželjnija u poređenju sa rešenjima koja sadrže jednako popunjena skladišta. Optimalna vrednost ovog parametra je 2 [9]. Vrednosti veće od dva dovode do prevremene konvergencije, jer se vrednosti funkcije prilagođavanja u parcijalnim rešenjima mogu previše približiti vrednosti funkcije prilagođavanja u optimalnom rešenju.

U kontekstu ACO algoritma, funkcija prilagođavanja se koristi u dve svrhe:

- *Upoređuje rešenja mrava međusobno*. Ovo je moguće jer smo već uspostavili razlike između mrava. Naime, ne može svaki mrav da pojačava feromon.
- Koristi se za *pojačavanje efekta* koji imaju najbolji elementi rešenja nađenih u prvom koraku.

3.6 Lokalno pretraživanje

Algoritam lokalnog pretraživanja počinje od nekog inicijalnog rešenja i vrši pretragu u određenom susedstvu tog rešenja sa ciljem da nađe bolje rešenje. Definisanje susedstva (okoline) je kritična tačka algoritma i mora se prilagoditi prirodi problema.

U mnogim aplikacijama razvijenim za rešavanje NP teških problema, ACO algoritam se najbolje ponaša kada je u kombinaciji sa lokalnim pretraživanjem. Ovaj algoritam lokalno optimizuje rešenja mrava, pa se ovakva rešenja koriste za pojačavanje feromona. Razlog za ovakvo ponašanje je što ACO tačnije sprovodi grubu pretragu, pa se rešenja kojima rezultuje ovakva pretraga kasnije lokalno optimizuju adekvatnim algoritmom.

Kako naći početno rešenje? Algoritmi za lokalnu pretragu pretražuju okolinu nekog inicijalnog rešenja, a naći dobro inicijalno rešenje nije lako. Taj posao obavlja ACO – mravi sa određenom verovatnoćom kombinuju pakete i tako grade dobra rešenja, koja su bazirana na prethodno nađenom optimumu, što je odlična početna tačka za lokalno pretraživanje.

Algoritam lokalnog pretraživanja, koji je već primenjivan na BPP i daje dobre rezultate, je korišćen u HGGA, tako da je i ovde primenjen. Ideja je sledeća: određen broj paketa u inicijalnom rešenju je *oslobođen*. Pokušava se *zamenom* jednog ili dva paketa iz svakog skladišta inicijalnog rešenja sa jednim ili dva oslobođena paketa. Pri tom se pazi da kapacitet skladišta nije prekoračen, ali da je skladište više popunjeno nego pre zamene. Nakon prolaska kroz sva skladišta, preostali slobodni paketi se dodaju rešenju pomoću izabrane heuristike. Ovaj pristup je inspirisan *kriterijumom dominacije* (Martello i Toth), gde se prednost daje skoro punim skladištima koja sadrže

velike pakete u odnosu na manje puna skladišta u kojima se nalaze paketi malih veličina.

U ACO algoritmu, svako rešenje kreirano od strane jednog mrava prolazi kroz fazu lokalne optimizacije. Tada se prazne *loc* najmanje punih skladišta, a paketi iz tih skladišta postaju *slobodni*. Parametar *loc* je definisan empirijski i o njemu ćemo govoriti u Sekciji 4. Dalje, za svako skladište koje nije ispražnjeno, ispituje se da li se neki paket iz njega može zameniti nekim od slobodnih paketa (ili više njih), ali tako da skladište bude više popunjeno nego pre zamene. Algoritam pokušava da zameni dva paketa sa dva slobodna, dva paketa jednim slobodnim ili jedan paket jednim slobodnim paketom. Paketi koji ostaju neraspoređeni se ubacuju u rešenje pomoću neke od četiri prethodno objašnjenih heuristika. Time se kreira *kompletno rešenje*. Ova procedura se ponavlja sa kreiranim rešenjem – prazne se njegovih poslednjih *loc* punih skladišta.

Ponavljanje procedure se odvija sve dok više nema pomaka u funkcijama prilagođavanja između rešenja pre i posle primene algoritma lokalnog pretraživanja. Ovaj algoritam koji uzima rešenje nastalo ACO algoritmom i brzo ga dovodi do najbližeg lokalnog optimuma.

Nakon toga, feromon se pojačava pomoću lokalno "dokazanih" rešenja.

3.6.1 Primer lokalnog pretraživanja

Neka je kapacitet skladišta jednak 10 i neka je izabrana heuristika kojom će preostali paketi da se smeštaju FFD. Parametar *loc* u primeru ima vrednost 2.

- *Rešenje pre primene lokalnog pretraživanja*
  | 6 3 | 7 2 | 4 3 | 5 2 | 3 5 | 4 4 |
  U svakom skladištu (između svaka dva znaka |), smešteni su paketi određenih veličina. U prvom skladištu se nalaze paketi veličina 6 i 3. Kapacitet skladišta je 10, broj iskorišćenih skladišta je 6.
  Gubitak u skladištu definišemo kao preostali prostor nakon smeštanja paketa u njega do popunjavanja kapaciteta skladišta. Tako u prvom skladištu imamo gubitak 1, a ukupan gubitak je jednak 12.

- *Otvaramo dva najmanje puna skladišta (to su skladišta 3 i 4) i oslobađamo pakete:*
  Preostala skladišta: **| 6 3 | 7 2 | 3 5 | 4 4 |**
  Slobodni paketi: **5, 4, 3, 2**

- *Vršimo zamenu:*

  Prvo skladište: | **6 3** | ⟶ | **6 4** | Ostalo {5, 3, 3, 2}.
  Drugo skladište: | **7 2** | ⟶ | **7 3** | Ostalo {5, 3, 2, 2}.

  Treće skladište: | **3 5** | ⟶ | **5 5** | Ostalo {3, 3, 2, 2}.
  Četvrto skladište: | **4 4** | ⟶ | **4 4** | Ostalo {3, 3, 2, 2}.

- *Smeštamo ostale pakete pomoću FFD algoritma:*

  Četvrto skladište: | **4 4** | ⟶ | **4 4 2** | Ostalo {3, 3, 2}.
  Novo skladište: {3, 3, 2} ⟶ | **3 3 2** | Ostalo ∅.

- *Konačno rešenje:*

  | 6 4 | 7 3 | 5 5 | 4 4 2 | 3 3 2 |
  Broj iskorišćenih skladišta je 5, a gubitak 2.

- *Ponavljanje procedure:*

  Nema mogućeg napretka; stajemo.

4

Eksperimentalni rezultati

U ovom poglavlju biće reči o rezultatima dobijenim računarskom analizom performansi opisanog algoritma. Prvo su razmatrane vrednosti parametara ove metode i predložene su optimalne. Nakon toga, upoređivani su rezultati heuristika FFD, NFD, BFD, WFD, kao i rezultati predloženog algoritma u zavisnosti od toga koja se brza heuristika koristi. Na kraju se rezultati algoritma porede sa rezultatima drugih metoda koje su dale odlične rezultate pri rešavanju ovog problema. Instance za testiranje su uzete iz [13].

4.1 Vrednosti parametara

Kada se ACO algoritam kombinuje sa metodom lokalnog pretraživanja, potrebno je predefinisati vrednosti parametara u odnosu na one date u sekciji 2.2.2. Do optimalnih vrednosti parametara se u većini slučajeva dolazi empirijski. Razmatrani parametri su centralni deo ACO algoritma – *br_mrava, β, γ, ρ, loc.* O svakom od njih biće dato detaljno objašnjenje.

4.1.1 Parametar *br_mrava*

Prvi parametar koji je neophodno definisati je *br_mrava*. U opštem slučaju, ovaj parametar zavisi od problema koji se rešava, pa mora biti dobijen eksperimentalno. Vrednosti ACO algoritma su stabilne kada govorimo o promeni vrednosti ovog parametra. Da bismo došli do optimalnog broja mrava, napravljeni su test primeri sa instacama raznih dimenzija. Test primeri *u120*, *u250*, *u500*, *u1000* govore koja je dimenzija instance u pitanju. Za testiranja korišćene su dimenzije 120, 250, 500 i 1000 retrospektivno.

Ovaj parametar ne zavisi u velikoj meri od veličine instance problema koji rešavamo. Najbolji rezultati su dobijeni kada je imao vrednost 10 i 15. Razlog činjenice da je manji broj mrava dovoljan za izgradnju rešenja može biti to što se svako rešenje dovodi do skoro optimalnog, nakon čega se primenjuje lokalno pretraživanje, pa manje mrava dovodi do rešenja koja mogu biti interesantna.

br_mrava	*u120*	*u250*	*u500*	*u1000*
10	0.057	0.083	0.458	1.285
15	0.068	0.099	0.567	1.659
20	0.078	0.161	1.103	1.457
25	0.083	0.140	0.733	1.862

Tabela 4.1: Vreme izvršavanja programa u zavisnosti od parametra *br_mrava* dato u sekundama.

4.1.2 Parametar β

Ovaj parameter definiše važnost informacije koju nosi heuristika. U ovom radu, ACO je dobijao ovu informaciju iz četiri različite heuristike – FFD, NFD, BFD i WFD redom. Ponašanje parametra je uzeto iz aritmetičke sredine ovih algoritama – za svaku instancu i vrednost parametra, parametar je ispitan za sve četiri heuristike i uzeta je aritmetička sredina dobijenih vrednosti.

Mnogi problemi kombinatorne optimizacije postaju pogodni za rešavanje kada je vrednost parametra β=2. U ACO algoritmu sa lokalnim pretraživanjem ovaj parametar nije suštinski, jer metoda lokalnog pretraživanja koristi informaciju iz heuristike da dokaže korektnost rešenja i da ga poboljša, pa je važnost ove informacije umanjena. Kod velikih instanci problema, β=1 daje najbolje rezultate (Tabela 4.2).

β	*u120*	*u250*	*u500*	*u1000*
1	0.077	0.092	0.606	1.516
2	0.065	0.088	1.351	4.329
5	0.083	0.158	12.749	25.574
10	0.104	0.234	38.013	473.703

Tabela 4.2 : Vreme izvršavanja programa u zavisnosti od parametra *β* dato u sekundama.

4.1.3 Parametar *z*

Parametar *z*, koji definiše funkciju prilagođavanja, ima optimalnu vredost jednaku 2. Ovo je dokazao Falkenauer [8]. Za $z = 1$ se dobijaju znatno lošiji rezultati, dok za $z > 2$ rezultati nisu bolji od rezultata dobijenih za optimalnu vrednost parametra.

4.1.4 Parametar *γ*

Iz činjenice da se lokalno pretraživanje fokusira na interesantna mesta u prostoru rešenja sledi da je za pretragu potrebna manja eksploatacija mrava. Zato parametar γ uzima male vrednosti. Testiranjem je pokazano da je optimalna vrednost ovog parametra 1. Ovo znači da će se svako pojačavanje feromona biti izvršeno pomoću globalno najboljeg mrava.

γ	*u120*	*u250*	*u500*	*u1000*
1	0.057	0.083	0.442	1.171
2	0.062	0.094	0.687	1.784
3	0.063	0.099	0.504	1.301
5	0.058	0.099	0.614	1.175

Tabela 4.3 : Vreme izvršavanja programa u zavisnosti od parametra *γ* dato u sekundama.

4.1.5 Parametar *ρ*

Parametar evaporacije feromona ρ je davao vrlo nejasne rezultate, jer za različite dimenzije instance su se različite vrednosti parametra pokazale najboljim. Pri tom se ovde, kao parametar ρ podrazumeva 1- ρ iz Sekcije 2.2.1 (uzeto radi lakšeg zapisa).

Uglavnom sve vrednosti između 0.1 i 0.9 daju dobre rezultate. Međutim, niske vrednosti znače da feromon traje samo jednu generaciju, pa su nova rešenja

vođena isključivo prethodnim. Ovo dovodi do agresivne pretrage prostora, pa se za neke teške probleme optimtimum nađe veoma brzo, što može izazvati zaglavljivanje algoritma blizu lokalnog optimuma.

Razne vrednosti su isprobane, ali su uzete veće jer su manje nesigurne kada je reč o konvergenciji ka lokalnom optimumu.

ρ	*u120*	*u250*	*u500*	*u1000*
0.10	0.068	0.109	0.474	1.275
0.25	0.063	0.104	0.613	1.124
0.50	0.062	0.110	0.401	1.177
0.75	0.063	0.112	0.453	1.197
0.95	0.062	0.097	0.518	0.986

Tabela 4.4 : Vreme izvršavanja programa u zavisnosti od parametra *ρ* dato u sekundama.

4.1.6 Parametar *loc*

Poslednji parametar koji treba ispitati je broj skladišta *loc* iz kojih oslobađamo pakete u fazi lokalnog pretraživanja. Ovaj parametar zavisi od instance koju testiramo. Algoritam daje optimalno rešenje za različite vrednosti ovog parametara, pa je uzeta vrednost 4, uz prethodnu eksperimentalnu proveru.

loc	*u120*	*u250*	*u500*	*u1000*
2	0.066	0.103	1.050	4.076
3	0.063	0.110	0.586	1.873
4	0.068	0.100	0.482	1.142
5	0.071	0.172	0.626	1.294
10	0.083*	0.174	0.896	1.170

Tabela 4.5 : Vreme izvršavanja programa u zavisnosti o parametra *loc* dato u sekundama.
Vrednosti ostalih parametara su optimalne.
* označava da nije dobijen teorijski optimum

4.2 Poređenje brzih heuristika

U Sekciji 2.1 su predstavljene heuristike koje se najčešće koriste kod rešavanja problema pakovanja i daju najbolje rezultate. Objašnjena je važnost heuristike i informacije koju ona nosi, a od koje zavisi brzina celokupnog algoritma.

U ovom poglavlju biće upoređene brzine izvršavanja pomenutih heuristika. One su pokretane za različite dimenzije instanci.

Heuristike su implementirane u Javi a testiranje je izvšeno na mašini koja ima procesor Intel Celoron, 1.73GHz.

	u120	*u250*	*u500*	*u1000*
FFD	0.531033	0.597954	0.786479	1.098585
NFD	0.608417	0.674830	0.798720	1.142213
BFD	5.284068	5.578800	7.497821	14.670705
WFD	5.163053	5.368559	6.881902	12.545984

Tabela 4.6 : Rezultati upoređivanja vremena izvršavanja heuristika.
Vreme dato u milisekundama.

Iz tabele se vidi da heuristika FFD daje najbolje rezultate pri rešavanju jednodimenzionog problema pakovanja. NFD je zanemarljivo sporija heuristika, dok BFD i WFD zahtevaju mnogo više vremena za izvršavanje. Razlog ovome je prolazak kroz niz skladišta koji se obavlja pomoću pomenutih heuristika. Pri svakom pokušaju smeštanja paketa, prolazi se kroz niz svih skladišta i računa koliko slobodnog mesta je u svakom od njih ostalo. Nakon toga, računa se koje skladište ima najmanje, odnosno najviše slobodnog mesta takvog da se izabrani paket može u njega smestiti.

Ova procedura čini algoritme BFD i WFD znatno komplikovanijim, ali ne i efikasnijim. Naime, ni jedna od heuristika nije dostigla teorijski optimum. Sada dolazi do izražaja značaj koji ACO algoritam ima – on inicijalno spakovana rešenja probanim heuristikama „preuređuje“ i dolazi do skoro optimalnog rešenja, a u većini slučajeva optimalnog rešenja.

U tabeli 4.7 dat je broj skladišta koje su heuristike iskoristile. U koloni *Opt* dati su teorijski optimumi koje su dostigle heuristike FFD, NFD, BFD i WFD za dimenzije instanci 120, 250, 500 i 1000 redom. *Opt+x* znači da je određena heuristika pakete rasporedila u *x* skladišta više nego optimalno rešenje.

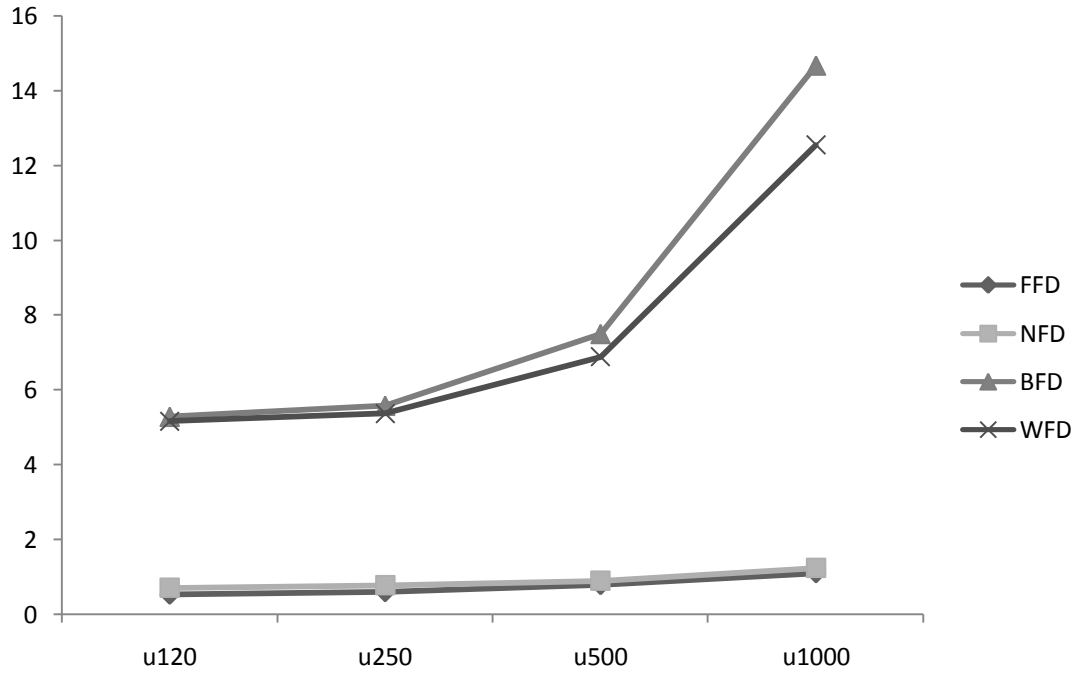
	Opt	FFD	NFD	BFD	WFD
u120	***48***	*Opt* + 1	*Opt* + 19	*Opt* + 1	*Opt* + 2
u250	***99***	*Opt* + 1	*Opt* + 39	*Opt* + 1	*Opt* + 2
u500	***198***	*Opt* + 2	*Opt* + 82	*Opt* + 2	*Opt* + 2
u1000	***399***	*Opt* + 4	*Opt* + 164	*Opt* + 4	*Opt* + 4

Tabela 4.7 : Broj iskorišćenih skladišta u svakoj heuristici.

Razlika u broju skladišta NFD heuristike u odnosu na preostale tri proizilazi iz toga što se preko NFD posmatra *samo sledeći paket* u odnosu na trenutni koji je raspoređen. Sledeći paket se pokušava smestiti u trenutno otvoreno skladište, a ako se u tome ne uspe, otvara se novo. Dakle, ne prolazi se kroz celokupni niz paketa, niti kroz niz već otvorenih skladišta. Heuristika BFD testirana na datim instancama daje iste rezultate kao FFD, ali je vreme izvršavanja znatno veće.

Iz pokazanog proizilazi da je za rešavanje jednodimezionog problema pakovanja najbolje koristiti heuristiku FFD. U sledećem poglavlju videćemo koja se heuristika najbolje pokazala pri kombinovanju sa ACO metaheuristikom i lokalnim pretraživanjem.

Na Slici 4.1 je dato poređenje vremena izvršavanja pomenutih heuristika.



Slika 4.1 : Poređenje vremena izvršavanja heuristika.

4.3 Efikasnost ACO algoritma kombinovanog sa različitim heuristikama

Kvalitet ACO algoritma dosta zavisi od informacije koju dobija iz heuristike. U prethodnom poglavlju su upoređena vremena izvršavanja četiri već predstavljene metode.

U ovom poglavlju biće upoređeni rezultati ACO algoritma u zavisnosti od toga od koje heuristike algoritam dobija informaciju. Sve vrednosti parametara su optimalne (videti Sekciju 4.1).

	u120	*u250*	*u500*	*u1000*
ACO (FFD)	49.949217	81.303023	648.139368	2843.421072
ACO (NFD)	52.020834	79.765569	491.532009	1089.491660
ACO (BFD)	51.352208	104.329493	814.436775	1201.873814
ACO (WFD)	51.321411	98.105162	855.232495	1147.719506

Tabela 4.8 : Rezultati upoređivanja vremena izvršavanja ACO algoritma. Vreme dato u milisekundama.

Iz tabele se vidi da se ACO algoritam, koji informaciju dobija iz FFD heuristike, dobro ponaša kada su u pitanju instance manjih dimenzija. Za rešavanje problema velikih dimenzija, kvalitetnijom se pokazala heuristika NFD.

U praksi se najboljom heuristikom za kombinovanje sa ACO algoritmom pokazala FFD. Generalno, ona je i najbrža jer se završava čim dođe do kraja niza paketa kroz koji prolazi samo jednom. Sa druge strane, BFD (odnosno WFD) se ne zadovoljava prvim dovoljno velikim skladištem u koje može stati paket. Ovim dolazimo do uštede, jer ćemo sačuvati skladišta koja imaju više slobodnog prostora, a paket ćemo smestiti u skladište koje ima minimum slobodnog prostora dovoljnog da primi paket date veličine. Dakle, FFD se pokazala kao najbrža, ali ona nema tendenciju da minimizuje gubitak skladišta u onoj meri u kojoj to radi BFD. NFD je modifikacija FFD heuristike.

Rezultati poređenja[1] su sledeći:

$$ACO\ (NFD) < ACO\ (FFD) < ACO\ (WFD) < ACO\ (BFD)$$

Od značaja je napomenuti kada je aplikacija dostigla teorijski optimum.Ovde dolazi do izražaja efikasnost ACO algoritma hibridizovanog lokalnim

[1] Poređenje je vršeno na osnovu vremena izvršavanja algoritma. Kako se NFD pokazao bržom za instance velikih dimenzija, ova heuristika je uzeta za najbržu.

pretraživanjem. U prethodnom poglavlju je pokazano da nijedna od četiri heuristike ne dostiže teorijski optimum – međutim, po performansama su FFD i BFD bile bolje od ostale dve heuristike. Zato je jasna dobra vrednost inicijalnog rešenja. Tabelom 4.9 dato je poređenje broja skladišta koje je metoda iskoristila u zavisnosti od inicijalnog rešenja (izabrane heuristike).

	Opt	*ACO (FFD)*	*ACO (NFD)*	*ACO (BFD)*	*ACO (WFD)*
u120	***48***	*Opt*	*Opt*	*Opt*	*Opt*
u250	***99***	*Opt + 2*	*Opt + 2*	*Opt*	*Opt*
u500	***198***	*Opt*	*Opt + 3*	*Opt + 1*	*Opt + 1*
u1000	***399***	*Opt*	*Opt*	*Opt* + 3	*Opt + 2*

Tabela 4.9 : Broj skladišta ACO algoritma u zavisnosti od početne informacije.

4.4 Poređenje ACO algoritma sa drugim pristupima BPP-u

ACO algoritam je upoređen sa MTP, HGGA i AP algoritmom. Test primeri su uzeti iz [13]. Ovi primeri imaju kapacitet skladišta 150, dok je broj paketa 120, 250, 500 i 1000 redom. Vrednosti parametara koje su korišćene su one koje su u Sekciji 4.1 dale najbolje rezultate za instance određenih dimenzija.

Upoređeni su ACO algoritmi sa vrednostima iz heuristika FFD i NFD, jer su ove dve heuristike dokazano brze od BFD i WFD. Razlog ovome je što se FFD bolje ponaša kada su instance manjih dimenzija u pitanju, dok se heuristika NFD pokazala bolje za instance većih dimenzija (videti Sekciju 4.3).

Kako su algoritmi pokretani na raznim mašinama, skalirano je CPU vreme da bi rezultati bili uporedivi. Istraživanjem je utvrđeno da je vreme CPU vremena dobijenog na ovoj mašini neophodno pomnožiti konstantom 4,6 da bi dobijeni rezultati bili uporedivi.

Na donjim tabelama dati su rezultati upoređivanja vremena izvršavanja algoritama i broj skladišta koje su algoritmi iskoristili u rešenjima.

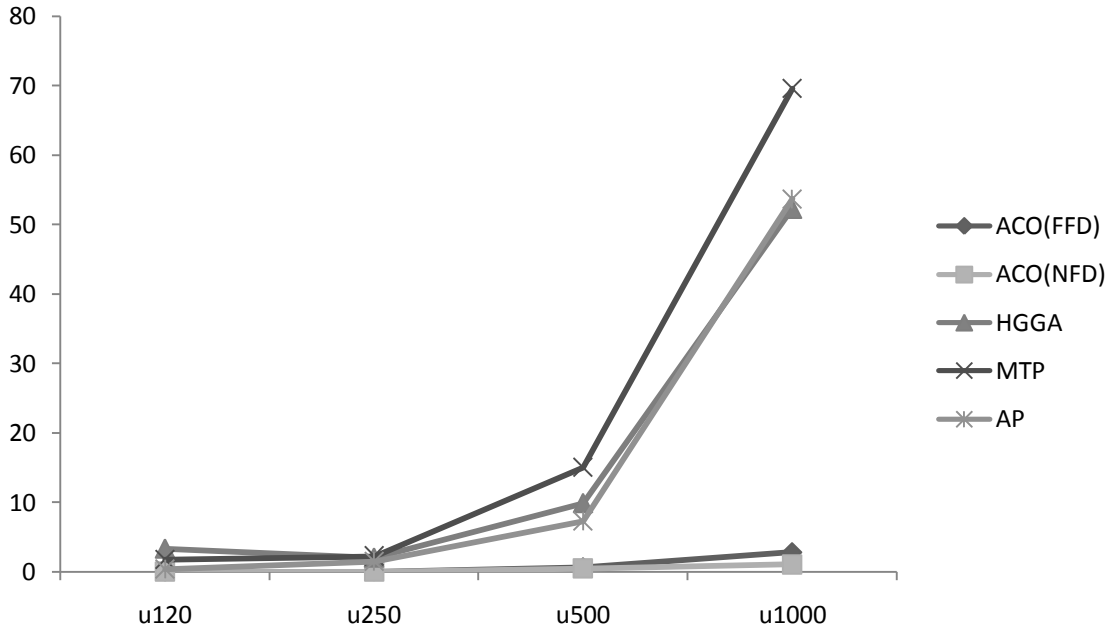
	u120	*u250*	*u500*	*u1000*
ACO (FFD)	0.049949	0.081303	0.648139	2.843421
ACO (NFD)	0.052021	0.079765	0.491532	1.089492
HGGA	3.339587	2.050825	9.977732	52.317715
MTP	1.848113	2.325678	15.090400	69.615887
AP	0.430262	1.585570	7.384401	53.702232

Tabela 4.10 : Rezultati upoređivanja ACO algoritma sa drugim algoritmima. Vreme dato u sekundama

	u120	*u250*	*u500*	*u1000*
ACO (FFD)	*Opt*	*Opt* + 2	*Opt*	*Opt*
ACO (NFD)	*Opt*	*Opt* + 2	*Opt* + 3	*Opt*
HGGA	*Opt* + 2	*Opt* + 3	*Opt*	*Opt*
MTP	*Opt* + 2	*Opt* + 12	*Opt* + 44	*Opt* + 78
AP	*Opt*	*Opt*	*Opt*	*Opt* + 4

Tabela 4.11 : Broj iskorišćenih skladišta.

Važno je napomenuti da su se situacije u kojima nije dostignut teorijski optimum dešavale usled variranja parametra *loc*. Iz priloženog se vidi da su rezultati ACO algoritma komparativni sa rezultatima do sad najboljih metoda za rešavanje jednodimenzionog problema pakovanja.



Slika 4.2 : Poređenje vremena izvršavanja algoritama

Dalje, izvršavanje je upoređeno sa ILOG AMPL-om, ali treba napomenuti da je ova verzija AMPL-a studentska i da se instance do određenih dimenzija mogu pokretati, inače se prekoračava broj dozvoljenih uslova i promenljivih studentske verzije AMPL-a. AMPL je jezik koji se koristi za probleme linearne i nelinearne optimizacije. Pored *Cplex* solvera, probleme u AMPL-u rešavaju solveri *Minos* i *Gurobi*. Minos ne rešava probleme celobrojnog programiranja (Integer Programming - IP), već ih rešava ignorišući uslov celobrojnosti, te njime BPP nije rešavan. Gurobi je

solver koji rešava probleme linearnog programiranja (Linear Programming - LP) i celobrojnog programiranja pa su rezultati koje je dao uporedivi sa rezultatima Cplex-a.

Dimenzija instance	***ACO***	***ILOG AMPL – CPLEX***	***ILOG AMPL - GUROBI***
15	0.011	0.027	0.016
20	0.021	0.073	0.016
25	0.011	0.049	0.016
30	0.022	/	0.047
35	0.020	/	0.468
40	0.026	/	0.031
45	0.024	/	/

Tabela 4.12 : Rezultati upoređivanja ACO-a sa AMPL-om. Vreme dato u sekundama.
/ označava da je prekoračen broj dozvoljenih promenlljivih
i uslova stud.verzije AMPL-a

.

5

Zaključak

Kao jedan od glavnih faktora stresa u modernom životu javlja se nedostatak vremena a samim tim i čekanje. Međutim, kada je raspored vremena dobro dizajniran, čekanje može biti minimizovano. Problemi pakovanja su sveprisutni u svakodnevnici – organizacija časova u školama, raspored polaska vozova, aviona i autobusa, raspoređivanje poslova na mašine koje se koriste u svakodnevnoj proizvodnji. Matematika nudi dubok uvid u probleme pakovanja, a samim tim i u organizaciju svakodnevnog života.

Postoji puno primena problema pakovanja koje nam mogu pasti na um. Smeštanje fajlova u računaru u memorijske blokove određenih veličina, ili snimanje muzike na disk, samo su neke od njih.

Moć matematike leži u tome što za razne probleme postoje matematičke tehnike koje se primenjuju u mnogobrojnim situacijama. Kada se neki problem prilagođava određenoj situaciji, dolazi do napretka u njegovoj formulaciji. Čini se kao da je nemoguće sve probleme raspoređivanja i organizacije zadataka utopiti u ideju o problemu pakovanja. Međutim, termini *procesor*, *mašina*, *zadatak* zamenjuju ono što je u originalnoj verziji problema termin *skladište*.

Ovim se dobijaju široke primene problema čije je rešavanje u radu razmatrano. Pokušano je da se raznim metodama kombinatorne optimizacije i njihovim kobminovanjem dobije efikasan rezutat. Nakon toga, rezultati su upoređeni sa metodama koje su do sada davale dobre rezultate i potvrđeno je da je hibridna

metoda dobijena kombinovanjem heuristike FFD ili NFD, ACO metaheuristike i lokalnog pretraživanja metoda koja daje odlične rezultate kada je u pitanju rešavanje jenodimenzionog problema pakovanja.

Cilj je nastaviti rad i unaprediti metodu. Ideja je da se kombinovanjem algoritama koji su brzi i efikasni dobije novi algoritam koji će biti efikasan u rešavanju ovog široko primenljivog problema.

Bibliografija

[1] **A. Merkle, M. Middendorf.** Swarm intelligence. *In: Search Methodologies. Springer, 2005, p. 401-435.*

[2] **B. Bullnheimer, R. F. Hartl, C.A. Strauss.** New rank based version of the ant system: A computational study. *Central European Journal for Operations and Economics,1999, p.25-38.*

[3] **C. Reeves.** Modern heuristic techniques for combinatorial problems. *Blackwell Scientific Publications, Oxford, 1993.*

[4] **D. Johnson, A. Demers, J. Ulman, M. Garey, R. Graham.** Worst-case performance bounds for simple one dimensional packing algorithms. *SIAM Journal on computing, 1974, p. 229-325.*

[5] **D. Karaboga, B. Basturk.** A powerfull and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global optimization, 2007, p. 459-471.*

[6] **D. Prisinger, M. Sigurd.** The two-dimensional bin-packing problem with variable bin sizes and costs. *Discrete Optimization, 2005, p. 154-167.*

[7] **E. Bischoff, G.Wascher.** Cuting and Packing. *European Journal of Operational Research, 1995, p. 503–505.*

[8] **E. Falkenauer.** A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics 2,1996, p. 5-30.*

[9] **E.Falkenauer, A.DelChambre.** A genetic Algorithm for Bin packing and Line Balancing. *In: Proc. Of the IEEE 1993 International Conference on Robotics and Automation, V.2, Piscataway, 1992, p.1186-1192.*

[10] **F. Glover.** Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 1986, p. 533-549.*

[11] **G. Gutin, A. Punnen.** The travelling salesman problem and Its Variations. *Kluwer Academic Publishers, Norwell, USA, 2002.*

[12] **H. Dyckhoff.** A typology of cutting and packing problems. *European Journal of Operational Research, 1990, p. 145-159.*

[13] http://people.brunel.ac.uk/~mastjjb/jeb/info.html

[14] **J. Hayek, A. Moukrim, S. Negre.** New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research, 2008, p.3184-3201.*

[15] **M. Dorigo.** Optimization, learning and natural algorithms. *PhD thesis, Politecnico di Milano, Italy, 1992.*

[16] **M. Dorigo, G. Di Caro**. The Ant Colony Optimization metaheuristics. *In:Corne, D. et al.(eds.): New Ideas in Optimization, McGraw Hill, UK, 1999, p. 11- 32.*

[17] **M. Dorigo, L. Gambradella.** Ant Colony System: A cooperative approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation,1997, p.53-66.*

[18] **M. Dorigo, T. Stutzle.** The ant colony optimization metaheuristics: Algorithms, aplications and advances. *In: Handbook of Metaheuristics. Kluwer Academic Publishers, 2002, p.251-285.*

[19] **M. Dorigo, V. Maniezzo, A. Colorni.** The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man and Cybernetics B, 1996, p. 29-41.*

[20] **M. Gary, D. Johnson.** Computers and Intractability: A Guide to Theory of Np-completeness. *W.H.Freman, New York, 1979.*

[21] **M. Gen, R. Cheng.** Genetic algorithms and engineering optimization. *Wiley & Sons, 2000, p.61-69.*

[22] **M. Labb, G. Laporte, S. Martello**. Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research, 2003, p. 490-498.*

[23] **S. Martello**, **P. Toth**. Knapsack Problems, Algorithms and Computer Implementations. *Wiley & Sons, UK, 1999.*

[24] **T. Stuztzle, H. Hoos.** Max-Min ant system. *Future generation Computer Systems, 2000, p. 889-914.*

[25] **T. Stuztzle, H. Hoos.** Stochastic local search: Foundations and applications. *Elsevier Inc., San Francisko, CA, 2005.*

[26] **V. Maniezzo.** Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing, 1999, p.358-369.*

[27] **V. Vazirani.** Aproximation algorithms. *Springer-Verlag New York Inc., New York, NY, USA, 2001.*

[28] **W. J. Gutjahr.** ACO algorithm with guaranteed convergence to the optimal solution. *Information Processing Letters, 1992, p. 172-184.*
[29] www.ams.org
[30] www.nist.gov