SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1973

Metode za rješavanje statičkog problema raspoređivanja u okruženju nesrodnih strojeva

Lucija Ulaga

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2019.

DIPLOMSKI ZADATAK br. 1973

Pristupnik: **Lucija Ulaga (0036483630)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Metode za rješavanje statičkog problema raspoređivanja u okruženju nesrodnih strojeva**

Opis zadatka:

Proučiti problem raspoređivanja u okruženju nesrodnih strojeva i metode koje se mogu iskoristiti za rješavanje statičke inačice problema. Prilagoditi problemu jedan postupak iscrpnog pretraživanja za dobivanje optimalnih rješenja. Ostvariti i isprobati različite inačice problemski specifičnih heurističkih postupaka, kao i nekoliko metaheurističkih postupaka za rješavanje statičkog problema raspoređivanja. Usporediti učinkovitost ostvarenih postupaka s postojećim rješenjima iz literature. Analizirati utjecaj veličine primjerka problema na učinkovitost ostvarenih postupaka. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Prof. dr. sc. Domagoj Jakobović

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Doc. dr. sc. Marko Čupić

Zahvaljujem se obitelji koja mi je svojom podrškom kroz sve godine studiranja pomogla ovo razdoblje života učiniti sretnim i ispunjenim, a posebno hvala roditeljima što su moj studij stavljali na prvo mjesto i time mi omogućili bezbrižan put do diplome.

Hvala mentoru prof. dr. sc. Domagoju Jakobovića na pomoći tijekom godina studiranja i također veliko hvala dr. sc. Marku Đurasevića na uloženom trudu, vremenu i savjetima u izradi ovog rada.

SADRŽAJ

1. Uvod

Optimizacija ili pronalazak najboljeg među svim mogućim rješenjima je vrlo čest i kompleksan problem koji se javlja u raznim područjima. Već dugi niz godina razvijaju se mnogobrojne metode i algoritmi za rješavanje raznovrsnih problema optimizacije. Bez obzira o kojem problemu se radi, prvi i vrlo bitan korak u rješavanju problema je odabir metode. To može biti dugotrajan proces, pogotovo ako u literaturi ne postoji međusobna usporedba različitih algoritama na sličnom ili istom problemu. Upravo to je motivacija ovog rada, kojem je cilj dati pregled i usporedbu raznih pristupa u rješavanju problema statičkog raspoređivanja u okruženju nesrodnih strojeva. Implementirane su razne metode, od iscrpnog pretraživanja i pravila raspoređivanja do heuristika i metaheuristika. Rezultati su dobiveni na problemima različitih dimenzionalnosti i uspoređeni s postojećim metodama iz literature.

Tekst je podijeljen u tri glavne cjeline. Prvo je detaljno opisan problem, zatim sve korištene metode i konačno su izneseni rezultati, usporedbe i zaključci o pojedinim metodama i pristupima.

2. Problem

Raspoređivanje je problem raspodjele aktivnosti na resurse pri čemu se teži izradi optimalnog rasporeda po jednom ili više zadanih kriterija.

Raspoređivanje na više paralelnih strojeva predstavlja problem koji nije proučavan samo u teoriji, već se javlja i u praksi. U industriji, raspoređivanje je bitan dio cjelokupnog procesa proizvodnje na produkcijskim linijama koje uglavnom imaju više strojeva koji rade paralelno. Drugi primjeri su raspoređivanje procesa u računalima u oblaku, višeprocesorskim računalima ili spletovima računala, zatim raspoređivanje pacijenata po operacijskim salama i slično. Dodatni primjeri opisani su u Pinedo (2005), Pinedo (2008) i Sule (2008).

Zbog česte primjene u praksi, kroz godine su razvijane mnogobrojne metode i algoritmi za rješavanje različitih varijanti problema raspoređivanja. Jedan općeniti algoritam na ulaz prima n poslova koje je potrebno distribuirati na m strojeva te razna svojstva poput trajanja poslova, vremena dolaska poslova u sustav, roka izvršavanja i težina poslova. Izlazna vrijednost algoritma, odnosno rješenje, je raspored dobiven minimizacijom vremena izvođenja ili drugih troškova.

Pojam raspoređivanja obuhvaća mnogo različitih varijanti problema kojima je cilj distribuirati poslove po strojevima. Stoga, postoji nekoliko klasifikacija - ovisno o okruženju strojeva, kriteriju optimizacije, dodatnim ograničenjima i uvjetima raspoređivanja.

Okruženje mogu činiti identični, uniformni ili nesrodni strojevi. Vremena izvršavanja pojedinog posla na identičnim strojevima ne ovise o stroju, odnosno trajanje posla je isto na svim strojevima. Trajanja pojedinog posla na uniformnim strojevima su proporcionalna brzinama pojedinog stroja, dok su na nesrodnim strojevima trajanja različita i nasumična. Također, pojedini posao može biti izvršen na jednom stroju ili na seriji strojeva.

Neki od najčešćih kriterija optimizacije su: ukupno trajanje rasporeda, kašnjenje svih poslova, vrijeme zadržavanja poslova u sustavu, maksimalno kašnjenje, broj zakašnjelih poslova, ukupno težinsko kašnjenje i drugi.

Primjeri dodatnih ograničenja koja se mogu uzeti u obzir pri izgradnji rasporeda su: kvarovi strojeva, izvođenje s prekidima poslova, ograničenje u izvođenju (nezavisni ili zavisni poslovi), grupna obrada poslova, vremena dolaska poslova u sustav, vremena postavljanja i mnogi drugi.

Zadnja podjela odnosi se na uvjete pri kojima se izvršava raspoređivanje, a to su: pouzdanost parametara (deterministički ili stohastički), dostupnost parametara (predodređeno raspoređivanje ili raspoređivanje na zahtjev) i konstrukcija rasporeda (statička ili dinamička). Statičko raspoređivanje je problem u kojem su sve informacije o problemu dostupne unaprijed te je raspored (rješenje) potrebno izraditi prije samog početka rada sustava. Kod dinamičkog raspoređivanja, informacije o poslovima nisu dostupne prije početka rada sustava, već postaju dostupne kako poslovi dolaze u sustav. Raspored je u tom slučaju potrebno konstruirati paralelno s radom sustava, pri čemu je brzina donošenja odluka najbitniji faktor.

Minimizacija ukupnog trajanja izvođenja na dva identična, paralelna stroja dokazano je NP-težak problem (Garey, 1997). Minimizacija ukupnog kašnjenja na jednom stroju je također NP-težak problem (Du i Leung, 1990). Stoga, kompleksnije instance problema izvedene iz ovih osnovnih također spadaju u klasu NP-teških problema.

Ovaj rad bavi se NP-teškim problemom statičkog raspoređivanja nezavisnih poslova na nesrodne strojeve, uz minimizaciju ukupnog težinskog kašnjenja. Razmatra se ograničenje vremena dolaska u sustav. Posao se mora izvršiti samo jednom na nekom od dostupnih strojeva. Stroj u nekom trenutku može izvršavati samo jedan posao. Kada posao započne na stroju, mora se na istom do kraja izvršiti. Svi parametri poslova su deterministički i dostupni prije početka rada sustava.

Instanca problema određena je brojem poslova n, brojem strojeva m te sljedećim parametrima poslova:

- težina (w_j) - pojedini posao ima definiranu težinu, neovisno o stroju
- vrijeme izvođenja ($p_{i,j}$) - svaki posao ima definiranih *m* vremena izvođenja, gdje je *m* broj strojeva u sustavu
- vrijeme dolaska u sustav (r_j) - pojedini posao ima definirano vrijeme kada do-

lazi u sustav, odnosno kada postaje dostupan za obradu

– rok izvršavanja posla (d_j) - za svaki posao definiran je rok do kada treba biti izvršen, te se na temelju njega određuje koliko posao kasni.

Minimizacija ukupnog težinskog kašnjenja definirana je formulom:

$$TWT(s) = \sum_{j=1}^{n} max(C_j - d_j, 0) \tag{2.1}$$

gdje C_j predstavlja vrijeme završetka posla j, a s predstavlja rješenje, odnosno raspored poslova po strojevima.

3. Metode

Metode opisane u ovom poglavlju podijeljene su u tri cjeline. Prvo je obrađena jedna metoda iscrpnog pretraživanja u kombinaciji s jednostavnim pravilom za pronalazak početnog rješenja, odnosno gornje granice. Nakon toga slijedi opis korištenih heuristika. Treća cjelina opisuje metaheuristike prilagođene za rad s problemom statičkog raspoređivanja. Navedeni su izvorni radovi koji opisuju metode, no moguće su razlike u implementaciji u odnosu na korištenu literaturu. Opisi metoda uključuju implementacijske detalje u obliku pseudokodova. Svim metodama zajednička je funkcija dobrote koja je jednaka funkciji cilja, odnosno kriteriju minimizacije koji je definiran formulom 2.1.

3.1. Branch and bound

Cilj korištenja iscrpne pretrage je pronalazak optimalnih rješenja zadanih problema. Budući da je u pitanju NP-težak problem, optimalna rješenja mogu se pronaći samo za male instance problema. Jedan od mogućih pristupa je branch and bound. Za algoritam branch and bound potrebno je odrediti gornju granicu pomoću heuristike ili ju procijeniti na neki drugi način. U prvom slučaju koristi se vrijednost dobivena evaluiranjem rasporeda kojeg je generirala heuristika ATC. Implementiran je i drugi način koji koristi vrijednosti dobivene genetskim algoritmom.

Branch and bound je postupak iscrpne pretrage koji pretražuje sve moguće kombinacije raspoređivanja poslova na strojeve. Algoritam rekurzivno obilazi stablo u kojem svaki čvor ima m djece pri čemu svako dijete predstavlja jedan stroj. Dubina stabla je n. Na svakoj razini stabla trenutni posao raspoređuje se na jedan stroj. Algoritam je implementiran zapravo kao branch and cut. U svakom čvoru evaluira se parcijalni raspored. Ako je njegova vrijednost funkcije dobrote veća od gornje granice, cijelo

podstablo se odbacuje. Drugim riječima, ako u nekom trenutku vrijednost funkcije dobrote parcijalnog rasporeda ima veću vrijednost od gornje granice, nema ga smisla graditi do kraja jer konačna vrijednost funkcije dobrote cijelog rasporeda može biti samo jednaka ili veća. Budući da se radi o problemu minimizacije, optimalno rješenje ima dobrotu koja je sigurno jednaka ili manja od gornje granice. Što je metoda za dobivanje gornje granice bolja, to će manje kombinacija biti potrebno obići postupkom branch and bound. Također, kada je pronađeno rješenje s dobrotom manjom od gornje granice, ona se postavlja kao gornja granica. Time se dodatno može smanjiti prostor pretrage.

Budući da za veće instance problema nema garancije da će se algoritam završiti u nekom prihvatljivom vremenu, izvođenje je vremenski ograničeno. Pri pokretanju zadaje se broj sati. Ako se u tom vremenu ne pronađe niti jedno potpuno rješenje, postupak vraća prazan raspored. Inače, rezultat pretrage je najbolji do tad nađeni raspored. Također, pretraga se prekida u slučaju da je nađen raspored koji ima vrijednost funkcije dobrote jednaku nuli jer kod problema minimizacije ukupnog težinskog kašnjenja nula sigurno predstavlja globalni minimum.

Postupak branch and bound prikazan je algoritmima 1 i 2. Rekurzija prima listu poslova koja određuje kojim redoslijedom se poslovi raspoređuju na strojeve. Za zadan redoslijed, obilaze se sve mogućnosti raspoređivanja poslova po strojevima. Ako je broj poslova jednak n, takvih lista poslova, odnosno permutacija, ima $n!$. Algoritam 1 generira permutacije redom te za svaku poziva rekurzivni postupak $bnbRecursion$ prikazan algoritmom 2. Za početnu permutaciju uzima se redoslijed poslova koji nije nasumičan. Konkretno, niz poslova sortira se po vremenu dolaska u sustav. Veća je vjerojatnost generiranja boljeg rasporeda ako se prvo raspoređuju poslovi koji najranije dolaze u sustav. Zbog vremenske složenosti, kod većih instanci problema, pretragu nije moguće izvršiti do kraja stoga ima smisla težiti tome da prve istražene kombinacije nisu nasumične. Drugim riječima, ovakvom optimizacijom ne smanjuje se prostor pretrage, već povećava vjerojatnost da je prostor boljih rješenja istražen ranije. Osim sortiranja poslova po vremenu dolaska u sustav, implementirana je i druga varijanta. Problem je optimiran genetskim algoritmom te se za početnu permutaciju uzima najbolje nađeno rješenje.

Algoritam 1 BranchAndBound

Input: $upperBound$

1: $permutation \leftarrow$ početna permutacija poslova
2: **while** nisu obiđene sve permutacije **do**
3: $\quad bnbRecursion(permutation, upperBound)$
4: $\quad$ **if** pronađen minimum $== 0$ **then**
5: $\quad\quad$ **break**
6: $\quad permutation \leftarrow$ generiraj iduću permutaciju
7: **return** najbolji pronađeni raspored

Algoritam 2 bnbRecursion

Input: $permutation$, $upperBound$

1: $depth \leftarrow depth + 1$
2: **if** prekoračeno vremensko ograničenje **then**
3: $\quad$ **return**
4: **if** $depth == n$ **then**
5: $\quad$ ažuriraj $bestSchedule$ i $bestFitness$
6: $\quad$ **if** $bestFitness < upperBound$ **then**
7: $\quad\quad upperBound \leftarrow bestFitness$
8: $\quad$ **if** pronađen globalni minimum **then**
9: $\quad\quad$ **return**
10: **else**
11: $\quad job \leftarrow$ trenutni posao određen pomoću $depth$ i $permutation$
12: $\quad$ **for all** $i \in m$ **do**
13: $\quad\quad$ dodaj posao job na stroj i
14: $\quad\quad$ **if** dobrota parcijalnog rasporeda $> upperBound$ **then**
15: $\quad\quad\quad$ makni posao job sa stroja i
16: $\quad\quad$ **else**
17: $\quad\quad\quad bnbRecursion(permutation, upperBound)$
18: $\quad\quad\quad$ **if** pronađen globalni minimum **then**
19: $\quad\quad\quad\quad$ **return**
20: $\quad\quad\quad$ makni posao job sa stroja i
21: $\quad$ **return**

3.1.1. ATC

ATC, ili punim nazivom *Apparent Tardiness Cost*, je pravilo raspoređivanja za rješavanje problema minimizacije ukupnog težinskog kašnjenja. Heuristiku su razvili Vepsalainen i Morton (1987) te Ow i Morton (1989). ATC generira raspored računajući prioritete poslova u trenutcima kada postoji slobodan resurs, odnosno stroj. Prioritet dodjeljivanja posla stroju mijenja se kroz vrijeme po sljedećoj formuli (Lee (1997)):

$$I_{i,j}(t) = \frac{w_j}{p_{i,j}} exp\left[-\frac{max(d_j - p_{i,j} - t, 0)}{k\overline{p}}\right] \tag{3.1}$$

Oznake i njihovo značenje:

j	posao
i	stroj
t	trenutno vrijeme
$I_{i,j}(t)$	prioritet raspoređivanja posla j na stroj i u trenutku t
w_j	težina posla (*weight*)
$p_{i,j}$	vrijeme izvođenja posla j na stroju i (*processing time*)
d_j	do kada posao treba biti obavljen (*due date*)
k	slobodni koeficijent
$\overline{p}$	prosječno trajanje izvođenja poslova koji još nisu raspoređeni

Vrijednost parametra k određuje se eksperimentalno. Po Vepsalainen i Morton (1987), ta vrijednost je najčešće između 1 i 3 kod problema raspoređivanja na jednom stroju.

Heuristika ATC prikazana je algoritmom 3. U svakom trenutku kada stroj postane slobodan, ATC računa prioritete svih neraspoređenih poslova i bira onog s najvećim prioritetom (linije 5 do 9). U slučaju da najbolji pronađeni stroj za posao s najvećim prioritetom nije slobodan u trenutku t, posao se svejedno miče s liste $unscheduedJobs$ i pokuša se ponovo rasporediti u idućoj iteraciji *while* petlje. Heuristika vraća izgrađeni raspored.

Raspored dobiven ovom heuristikom je dobra početna točka za daljnju pretragu stoga se koristi kao početno rješenje u nekim kasnije opisanim heuristikama i metaheuristikama.

Algoritam 3 ATC

1: **while** neraspoređeni poslovi **do**
2: $t \leftarrow$ trenutak kada su bar jedan posao i bar jedan stroj slobodni
3: $unscheduledJobs \leftarrow$ poslovi koji su došli u sustav prije ili u trenutku t i nisu još raspoređeni
4: **while** $unscheduledJobs \neq \emptyset$ **do**
5: **for all** $j \in unscheduledJobs$ **do**
6: **for all** $i \in m$ **do**
7: $\bar{p} \leftarrow$ srednja vrijednost vremena trajanja neraspoređenih poslova, bez trenutnog posla *j*
8: $priority \leftarrow \frac{w_j}{p_{i,j}} * exp\left[-\frac{max(d_j - p_{i,j} - t, 0)}{k\bar{p}}\right]$
9: $bestJob \leftarrow$ posao s najvećim prioritetom
10: $bestMachine \leftarrow$ stroj s najmanjim vremenom izvršavanja nakon dodavanja posla $bestJob$ na taj stroj
11: **if** $bestMachine$ slobodan u trenutku t **then**
12: rasporedi $bestJob$ na $bestMachine$
13: **return** raspored

3.2. Heuristike

Postoje razne heuristike za rješavanje problema raspoređivanja. U fokusu ovog rada su jednostavne heuristike koje koriste operatore lokalne pretrage koji primjerice mijenjaju redoslijed izvođenja poslova ili dodjele poslova resursima.

3.2.1. GRASP

GRASP ili *Greedy Randomized Adaptive Search Procedure* je iterativni postupak lokalne pretrage predložen od Feo i Resende (1995). Sastoji se od dvije faze: konstrukcija rješenja i faza poboljšavanja rješenja. Nasumična i pohlepna konstrukcija rješenja služi kao početno rješenje za drugu fazu koja lokalnom pretragom poboljšava rješenje, odnosno traži lokalni optimum. Postupak konstrukcije i lokalne pretrage ponavlja se do kriterija zaustavljanja. Osnovni kriterij zaustavljanja je zadani broj iteracija. U slučaju da trajanje izvođenja premaši definirano vremensko ograničenje, pretraga se također zaustavlja. Rezultat metode je najbolji pronađeni raspored.

Implementirane su tri heuristike temeljene na GRASP metodi kako je opisano u radu Paulo de C. M. Nogueira et al. (2014). Prva je klasični GRASP algoritam, zatim hibridna heuristika koja spaja GRASP i PR (Glover (1996)), i konačno kombinacija GRASP-a i ILS heuristike (Lourenço i Stützle (2003)). PR, ili punim nazivom *Path Relinking*, je strategija intenzifikacije koja istražuje prostor između dva dobra rješenja. ILS (*Iterated Local Search*) je iterativni postupak koji primjenjuje perturbacije na lokalni optimum.

Metoda GRASP prikazana je algoritmom 4. Funkcija f predstavlja funkciju dobrote, a $GreedyRandomizedConstruction$ je opisan algoritmom 5.

Algoritam 4 GRASP

Input: α

while $!stoppingCriteria$ **do**
  $s_0 \leftarrow GreedyRandomizedConstruction(\alpha)$
  $localOptimum \leftarrow false$
  while $!localOptimum$ **do**
    $neighbour \leftarrow$ najbolji susjed iz susjedstva od s_0
    if $f(neighbour) < f(s_0)$ **then**
      $s_0 \leftarrow neihgbour$
    else if $f(neighbour) == f(s_0)$ **then**
      $localOptimum \leftarrow true$
return najbolji pronađeni raspored

Konstrukcija rješenja

Prva faza GRASP algoritma konstruira raspored tako da u svakoj iteraciji jedan posao smjesti na jedan od strojeva. Pri konstrukciji koriste se lista kandidata C i ograničena lista kandidata RC. Lista C nastaje tako da se svaki neraspoređeni posao privremeno rasporedi na trenutno izgrađeni parcijalni raspored. Redoslijed kojim se poslovi obilaze određen vremenom dolaska poslova u sustav. Pri privremenom raspoređivanju, bira se onaj stroj na kojem parcijalni raspored s dodanim poslom ima najmanju vrijednost funkcije dobrote. Svi neraspoređeni poslovi se zatim sortiraju po vrijednostima funkcije dobrote. Tako lista C na prvom mjestu ima posao, i odgovarajući stroj, čijim se dodavanjem u izgrađeni parcijalni raspored stvara najmanji trošak. Lista RC uzima određen udio najboljih poslova iz liste C. Pritom se pod najbolje poslove smatraju oni koji stvaraju najmanji porast dobrote rješenja. Veličina l liste RC određena je parametrom α i računa se po formuli:

$$l = max(1, \alpha * |C|), \qquad \alpha \in [0, 1] \tag{3.2}$$

Parametar α kontrolira udio nasumičnosti i pohlepnosti tijekom konstrukcije rasporeda. U svakoj iteraciji posao se bira nasumično iz liste RC i stavlja na odgovarajući stroj. Ako je $\alpha = 0$, lista RC ima duljinu 1 i sadrži samo prvi posao iz liste C. Tada se gradi pohlepno rješenje - u svakoj iteraciji bira se najbolji posao. Ako je $\alpha = 1$, lista RC je jednaka listi C te se u svakom koraku posao bira potpuno nasumično.

Postupak konstrukcije rasporeda prikazan je algoritmom 5, preuzetim iz Paulo de C. M. Nogueira et al. (2014).

Algoritam 5 GreedyRandomizedConstruction

Input: α

$s \leftarrow$ prazan raspored
$C \leftarrow$ svi neraspoređeni poslovi
while $C \neq \emptyset$ **do**
    for all $j \in C$ **do**
        privremeno rasporedi j na onaj stroj i gdje je vrijednost funkcije dobrote parcijalnog raspored $f(s)$ najmanja
    sortiraj C uzlazno po vrijednosti $f(s)$
    $RC \leftarrow$ uzmi redom $max(1, \alpha * |C|)$ poslova iz C
    nasumično odaberi posao j' iz RC
    rasporedi posao j' na odgovarajući stroj i'
    makni posao j' iz liste C
return s

Lokalna pretraga

Postupak lokalne pretrage počinje od rješenja konstruiranog u prvoj fazi GRASP metode. Pretražuje se susjedstvo rješenja i bira najbolji susjed. Lokalni operator generira susjeda umetanjem posla u raspored tako da se odabrani posao uklanja s originalne pozicije i ubacuje neko drugo mjesto u rasporedu. Susjedstvo je skup susjeda nastalih pojedinačnim umetanjima nad trenutnim rješenjem. Ako je pronađen susjed koji je bolji od trenutnog rješenja, on se postavlja za trenutno rješenje i lokalna pretraga se nastavlja. Kada je trenutno rješenje najbolje u susjedstvu, pretraga staje i odlazi se u novu iteraciju GRASP algoritma.

Veličina susjedstva nastalog umetanjima posla je $(n^2 - n + m)$ za paran broj strojeva i $(n^2 + m)$ za neparan broj strojeva. Sljedeći primjer preuzet iz Paulo de C. M. Nogueira et al. (2014) prikazuje susjedstvo za problem s četiri posla i dva stroja.

Početno rješenje:
$s = [[2, 1], [3, 4]]$

Susjedstvo nastalo umetanjem:

$s1 = [[1,2],[3,4]]$ $s2 = [[1],[2,3,4]]$ $s3 = [[1],[3,2,4]]$ $s4 = [[1],[3,4,2]]$
$s5 = [[2],[1,3,4]$ $s6 = [[2],[3,1,4]]$ $s7 = [[2],[3,4,1]]$ $s8 = [[3,2,1],[4]]$
$s9 = [[2,3,1],[4]]$ $s10 = [[2,1,3],[4]]$ $s11 = [[2,1],[4,3]]$ $s12 = [[4,2,1],[3]]$
$s13 = [[2,4,1],[3]]$ $s14 = [[2,1,4],[3]]$

3.2.2. GRASP i PR

Path Relinking istražuje puteve u lokalnom prostoru između dva dobra rješenja. Hibridna heuristika koja spaja GRASP i PR koristi memoriju za spremanje najboljih nađenih rješenja. Nakon što GRASP provede lokalnu pretragu, uzima se dobiveni lokalni optimum i jedno od najboljih, odnosno elitnih rješenja. Novo rješenje generira se na putu između ta dva rješenja. Dodavanje PR procedure u GRASP služi za poboljšavanje performansi.

Za provođenje PR procedure potrebni su početno rješenje s_0 i ciljno rješenje s_g, pri čemu vrijedi $s_0 \neq s_g$. Klasični PR postupak generira put između s_0 i s_g pomoću izmjena koje radi lokalni operator. Izmjene teže unijeti atribute ciljnog rješenja u početno rješenje. Među susjedima odabire se onaj koji najbolje unaprjeđuje rješenje i on služi kao početno rješenje u idućem koraku.

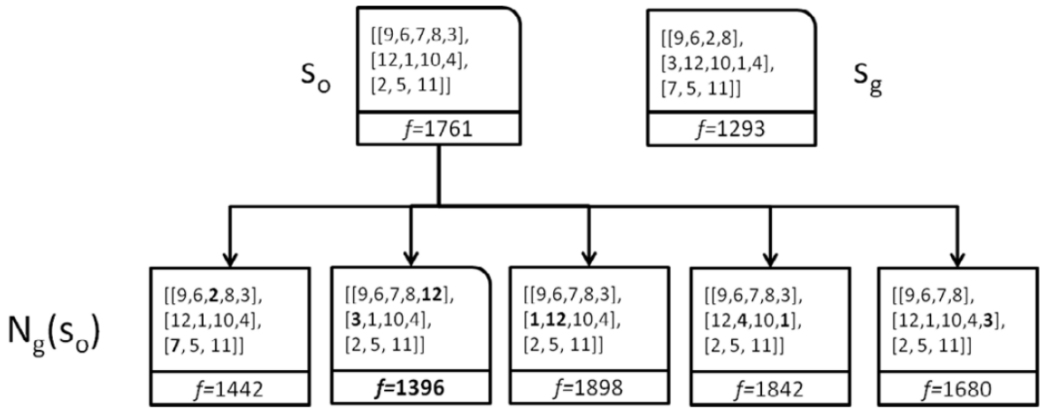
U ovom radu koristi se varijanta PR procedure koja se zove *Mixed Path Relinking* (Resendel M.G. (2005)). U prvom koraku se iz s_0 prema s_g stvara međurezultat s_1. U idućem koraku s_g postaje početno rješenje, a s_1 ciljno rješenje. Među njima se opet generira novi međurezultat s_2. U idućem koraku s_1 postaje početno rješenje, a s_2 ciljno rješenje. Ovaj postupak ponavlja se dok početno i ciljno rješenje ne postanu jednaki. Prednost *Mixed Path Relinking*-a je istraživanje lokalnog prostora oba rješenja.

Generiranje susjednih rješenja provodi se kao u radu Paulo de C. M. Nogueira et al. (2014). Koriste se zamjena i umetanje poslova. Zamjena služi za premještanje posla koji se u s_0 nalazi na drugačijoj poziciji nego u s_g. Neki posao j ima poziciju x u rasporedu s_0 i poziciju y u s_g. Na rasporedu s_0 izvršava se zamjena koja posao j stavlja na poziciju y, a posao j' s pozicije y stavlja na poziciju x. Umetanje služi za uklanjanje posla sa stroja i te stavljanje na drugi stroj i' kada u se na stroju i u s_0 nalazi više poslova u odnosu na isti stroj u rasporedu s_g. U tom slučaju, zadnji posao sa stroja i se uklanja i dodaje na stroj i' s manjim brojem poslova. Rezultat umetanja poslova

su nizovi poslova jednakih duljina kao u rasporedu s_g.

Iz Paulo de C. M. Nogueira et al. (2014) preuzet je primjer koji ilustrira nastanak susjednih rješenja. Početno rješenje je $s_0 = [[9, 6, 7, 8, 3], [12, 1, 10, 4], [2, 5, 11]]$, ciljno rješenje je $sg = [[9, 6, 2, 8], [3, 12, 10, 1, 4], [7, 5, 11]]$. Poslovi iz s_0 koji imaju drugačije pozicije u odnosu na s_g su : 7, 3, 12, 1, 4 i 2. Zamjene koje se izvršavaju su $7 \leftrightarrow 2$, $3 \leftrightarrow 12$, $12 \leftrightarrow 1$, $1 \leftrightarrow 4$. Poslovi nad kojima se ne provodi zamjena su 4 i 2. Za posao 4 zamjena nije moguća jer na stroju 2 ne postoji posao na petoj poziciji s kojim bi se posao 4 trebao zamijeniti. Zamjenom posla 2 nastalo bi rješenje koje već postoji. Budući da se na stroju 1 u s_0 nalazi više poslova nego na istom stroju u s_g, zadnji posao, u ovom slučaju 3, umetnut je na stroj 2, također na zadnje mjesto.

Slika 3.1 prikazuje generirano susjedstvo N_g za zadane s_0 i s_g. Kao međurezultat odabire se susjed s najmanjom vrijednost funkcije dobrote, u ovom primjeru to je raspored s vrijednosti $f = 1396$.



Slika 3.1: Susjedstvo u jednom koraku *Mixed Path Relinking* procedure (Paulo de C. M. Nogueira et al., 2014)

Postupak *Mixed Path Relinking* prikazan je algoritmom 6. Zamjena i premještanje poslova izvršavaju se na prethodno opisan način.

Hibridna heuristika koja spaja GRASP i PR prikazana je algoritmom 7, preuzetim iz Paulo de C. M. Nogueira et al. (2014).

Funkcija $EliteSetUpdate$ prikazana je algoritmom 8 , preuzetim iz Paulo de C. M. Nogueira et al. (2014). U set elitnih rješenja E sprema se najviše $eSize$ najboljih rješenja. Rješenje s dodaje se u E ako je bolje od barem jednog elitnog rješenja. U slučaju da je set pun, s se dodaje, a izbacuje se najlošije rješenje.

Algoritam 6 MixedPathRelinking

Input: s_0, s_g

$s_{best} \leftarrow \{\}$
while $s_0 \neq s_g$ **do**
    for all $j \in n$ **do**
        if posao j na istim pozicijama u s_0 i s_g **then**
            continue
        $neighbour \leftarrow$ napravi zamjenu poslova ako je moguća i ako ne stvara već postojeće rješenje (na rasporedu s_0)
        $s_{best} \leftarrow$ zapamti najboljeg pronađenog susjeda $neighbour$ (s najmanjom vrijednosti funkcije dobrote)
    for all $i \in m$ **do**
        if stroj i ima jednak broj poslova u s_0 i s_g **then**
            continue
        if stroj i ima više poslova u s_0 nego u s_g **then**
            pronađi drugi stroj i' koji na s_0 ima manje poslova nego u s_g, $i' \neg i$
            $neighbour \leftarrow$ zadnji posao sa stroja i premjesti na stroj i' (na rasporedu s_0)
        $s_{best} \leftarrow$ zapamti najboljeg pronađenog susjeda $neighbour$ (s najmanjom vrijednosti funkcije dobrote)
    $s_0 \leftarrow s_g$
    $s_g \leftarrow s_{best}$
return s_g

Algoritam 7 GRASP+PR

Input: $\alpha, eSize$

$E \leftarrow \emptyset$
$i \leftarrow 1$
while $!stoppingCriteria$ **do**
    $s \leftarrow GreedyRandomizedConstruction(\alpha)$
    $s_0 \leftarrow$ najbolji susjed iz susjedstva od s
    if $i \geq 2$ **then**
        $s_g \leftarrow$ nasumično odaberi rješenje iz E
        $s_0 \leftarrow MixedPathrelinking(s_0, s_g)$
        $E \leftarrow EliteSetUpdate(s_0, E, eSize)$
    else
        $E \leftarrow EliteSetUpdate(s_0, E, eSize)$
    $i \leftarrow i + 1$
return najbolji pronađeni raspored

Algoritam 8 EliteSetUpdate

Input: $s, E, eSize$

if $|E| == eSize$ **then**
    if $f(s) \leq max\{f(s')|s' \in E\}$**and**$s \notin E$ **then**
        ubaci s u E te izbaci najlošije rješenje
    $i \leftarrow i + 1$
else
    if $s \notin E$ **then**
        $E \leftarrow E \cup \{s\}$
return E

3.2.3. GRASP, ILS i PR

Iterated Local Search (ILS) je heuristika koja primjenjuje perturbacije na raspored dobiven lokalnom pretragom. U svakoj iteraciji, nakon perturbacije najboljeg rješenja, pokreće se proces lokalne pretrage. ILS završava kada se izvrši I_{ILS} iteracija bez poboljšanja. Upotreba heuristike GRASP u kombinaciji s metodom ILS pokazala se uspješnom na raznim kombinatoričkim problemima (Festa i Resende (2009)). U ovom radu koristi se heuristika koja kombinira GRASP, PR i ILS. Heuristika je slična metodi GRASP i PR, samo umjesto klasične lokalne pretrage koristi ILS heuristiku. Konkretno, u algoritmu 7 na mjestu gdje se dohvaća najbolji susjed (linija 5), ova hibridna heuristika poziva ILS metodu prikazanu algoritmom 9 (preuzeto iz Paulo de C. M. Nogueira et al. (2014)).

Algoritam 9 ILS

Input: s, d, I_{ILS}

1: $s^* \leftarrow$ najbolji susjed iz susjedstva od s
2: $i \leftarrow 0$
3: **while** $i < I_{ILS}$ **do**
4: $\quad s \leftarrow Perturbacija(s^*, d)$
5: $\quad s' \leftarrow$ najbolji susjed iz susjedstva od s
6: $\quad$ **if** $f(s') < f(s^*)$ **then**
7: $\quad\quad s^* \leftarrow s'$
8: $\quad\quad i \leftarrow 0$
9: $\quad i \leftarrow i + 1$
10: **return** najbolji pronađeni raspored (s^*)

Postupak perturbacije prolazi kroz dvije faze: destrukcija i konstrukcija, kako je predloženo od Ruiz i Stützle (2007). Kod destrukcije, d nasumično odabranih poslova miče se iz rasporeda. Tako nastali parcijalni raspored ulazi u fazu konstrukcije koja uklonjene poslove pohlepno vraća u raspored na bolje pozicije. Postupak vraća najbolji raspored nakon faze konstrukcije kako je prikazano algoritmom 10 (preuzeto iz Paulo de C. M. Nogueira et al. (2014)). U algoritmu se pod najboljim parcijalnim rasporedom misli na onaj s najmanjom vrijednosti funkcije dobrote. Trenutni posao j stavlja se na sve moguće pozicije u parcijalni raspored i odabire se ona pozicija za koju parcijalni raspored ima najmanju vrijednost funkcije dobrote.

Algoritam 10 Perturbacija

Input: s, d

1: $s_p \leftarrow s$
2: $R \leftarrow \emptyset$
3: **for all** $i \in d$ **do**
4: ukloni nasumično odabrani posao j s rasporeda s_p
5: $R \leftarrow R \cup \{j\}$
6: **for all** $j \in R$ **do**
7: $s_p \leftarrow$ najbolji parcijalni raspored nakon dodavanja posla j na sve moguće pozicije
8: **return** najbolji pronađeni raspored (s_p)

3.2.4. VNS

Variable Neighbourhood Search (VNS) razlikuje se od ostalih metoda lokalne pretrage po tome što koristi više od jednog susjedstva. Heuristika je predložena od Mladenović i Hansen (1997) te se pokazala kao dobra metoda za razne primjene (Hansen i Mladenovic (2001)). U svakoj iteraciji primjenjuje se jedna od dostupnih lokalnih pretraga, odnosno susjedstva. Kako proces ne bi trajao predugo, najbolje je koristiti tri susjedstva (Paula et al. (2007)).

U ovom radu implementirana su sljedeća susjedstva, kao i u Behnamian et al. (2009):

1. Umetanje posla s jednog stroja na drugi
2. Zamjena poslova na jednom stroju
3. Zamjena poslova na različitim strojevima

Umetanje posla uzima posao s najgoreg stroja i stavlja ga na najbolji stroj. Najgori stroj je onaj s najvećim troškom odnosno najvećom vrijednosti težinskog kašnjenja. Najbolji stroj je onaj koji ima najmanju vrijednost ukupnog težinskog kašnjenja. Vremenska složenost ovog postupka je O(n^2). Postupak je prikazan algoritmom 11, preuzetim iz Behnamian et al. (2009). U ovom i idućim algoritmima ulazni parametar s predstavlja rješenje za koje se traži susjedstvo, dok funkcija $f(s)$ predstavlja funkciju dobrote.

Algoritam 11 Susjedstvo1

Input: s

pronađi stroj i_1 s najvećom vrijednosti funkcije dobrote
pronađi stroj i_2 s najmanjom vrijednosti funkcije dobrote, $i_1 \neq i_2$
for all j na stroju i_1 **do**
    for all pozicija pos na stroju i_2 **do**
        $neighbour \leftarrow$ prebaci posao j sa stroja i_1 na poziciju pos na stroju i_2
        if $f(neighbour) < f(bestNeighbour)$ **then**
            ažuriraj $bestNeighbour$
return najbolji pronađeni raspored ($bestNeighbour$)

Zamjena poslova na jednom stroju obilazi sve kombinacije zamjene dva posla na istom stroju te vraća najbolju. Vremenska složenost algoritma je O($m * n^2$). Postupak je prikazan algoritmom 12, preuzetim iz Behnamian et al. (2009).

Algoritam 12 Susjedstvo2

Input: s

for all $i \in m$ **do**
    for all j_1 na stroju i **do**
        for all j_2 na stroju i, $j_1 \neq j_2$ **do**
            $neighboour \leftarrow$ zamjena poslova j_1 i j_2 u rasporedu s
            if $f(neighbour) < f(bestNeighbour)$ **then**
                ažuriraj $bestNeighbour$
return najbolji pronađeni raspored ($bestNeighbour$)

Zamjena poslova na dva stroja istražuje sve mogućnosti zamjene dva posla koji se nalaze na različitim strojevima. Vremenska složenost ove pretrage je veća od prethodnih i iznosi O($m^2 * n^2$). Postupak je prikazan algoritmom 13, preuzetim iz Behnamian et al. (2009).

Heuristika VNS u svakoj iteraciji teži korištenju najbrže metode generiranja susjedstva. Svaki put kada je pronađeno rješenje bolje od trenutno najboljeg, u idućoj iteraciji ponovo se koristi prvo susjedstvo. Ako nema poboljšanja, koristi se iduće susjedstvo. Kroz daljnje iteracije bez pronalaska boljeg rješenja, ciklički se mijenja koje susjedstvo se koristi. Na početku svake iteracije nasumično se bira jedan susjed iz susjedstva od trenutno najboljeg rješenja. Nad odabranim susjedom generira se novo susjedstvo i traži se najbolji susjed u tom susjedstvu. Kroz cijeli postupak uvijek se pamti najbolje

Algoritam 13 Susjedstvo3

Input: s

1: **for all** $i_1 \in m$ **do**
2: **for all** j_1 na stroju i **do**
3: **for all** $i_2 \in m, i_1 \neq i_2$ **do**
4: **for all** $j_2 \in i_2$ **do**
5: $neighbour \leftarrow$ zamjena poslova j_1 i j_2 u rasporedu s
6: **if** $f(neighbour) < f(bestNeighbour)$ **then**
7: ažuriraj $bestNeighbour$
8: **return** najbolji pronađeni raspored ($bestNeighbour$)

rješenje i od njega započinje svaka iteracija, no nasumičan odabir susjeda tog rješenja omogućava da pretraga nije potpuno pohlepna. Ne zapinje se odmah u lokalnom optimumu jer se istražuje nekoliko različitih susjedstva od rješenja koja nisu najbolja u svom susjedstvu.

Svaka implementirana metoda generiranja susjedstva ima definirano generiranje nasumičnog susjeda. Kod prve metode umetanja posla, nasumično se odabire jedan posao j i stroj i koji ne sadrži j. Zatim se nasumično odabire pozicija na stroju i na koju se ubacuje posao j. Kod zamjene poslova na istom stroju nasumično se odabiru stroj i te dva posla j_1 i j_2 sa stroja i. Nakon toga generira se susjed zamjenom poslova j_1 i j_2. Za generiranje nasumičnog susjeda zamjenom poslova na dva različita stroja, potrebno je nasumično odabrati: dva stroja i_1 i i_2, posao j_1 na stroju i_1 i posao j_2 na stroju i_2. Susjed nastaje zamjenom ta dva posla.

Kompletan postupak VNS lokalne pretrage prikazan je algoritmom 14, preuzetim iz Behnamian et al. (2009). Zaustavljanje algoritma određeno je brojem iteracija, no može se koristiti i vremensko ograničenje izvođenja. Varijabla l određuje koje se susjedstvo razmatra u pojedinoj iteraciji postupka. Kada se ne pronađe rješenje bolje od trenutnog, vrijednost l mijenja se ciklički (linija 11).

Algoritam 14 VNS

1: $S^* \leftarrow$ inicijalno rješenje, generirano nasumično
2: $l \leftarrow 1$
3: **while** !$stoppingCriteria$ **do**
4: $S \leftarrow S^*$
5: $S' \leftarrow$ pronađi nasumičnog susjeda iz susjedstva l
6: $S'' \leftarrow$ provedi lokalnu pretragu l nad rješenjem S'
7: **if** $f(S'') < f(S^*)$ **then**
8: $S^* \leftarrow S''$
9: $l \leftarrow 1$
10: **else**
11: $l \leftarrow l + 1$ modulo broj susjedstava
12: **return** najbolji pronađeni raspored (S^*)

3.2.5. VND

VND, punim nazivom *Variable Neighbourhood Descent*, je metoda lokalne pretrage predložena od Mladenović i Hansen (1997) te Hansen i Mladenovic (2001). Ideja je, kao i kod metode VNS, koristiti više lokalnih pretraga. Razlika ovdje je što je redoslijed izvođenja lokalnih pretraga deterministički. U radu Fanjul-Peyro i Ruiz (2010) korištena je heuristika koja spaja VND i IG (*Greedy Iterated*) ili RLS (*Restricted Local Search*) metode. IG je zapravo već opisan postupak perturbacije iz Ruiz i Stützle (2007). Dvije faze postupka su nasumično uklanjanje poslova iz rasporeda i pohlepno dodavanje poslova u raspored na bolje pozicije. RLS je jednostavnija modifikacija koja premješta jedan po jedan posao.

VND prvo stvara početno rješenje jednostavnom i brzom heuristikom. Svaki posao stavlja se na onaj stroj na kojem ima najmanje vrijeme izvršavanja. Iterativni postupak ponavlja se do kriterija zaustavljanja - zadani broj iteracija ili vremensko ograničenje. U svakoj iteraciji vrši se VND pretraga nakon koje se pronađeni lokalni optimum modificira jednom od dostupnih IG ili RLS metoda. U ovom radu implementirane su dvije lokalne pretrage unutar VND postupka, umetanje poslova i zamjena poslova, no moguće je koristiti proizvoljan niz metoda lokalne pretrage, pa tako i onih koje koristi ranije opisani postupak VNS. VND se zaustavlja kada je nađen lokalni optimum s obzirom na obje lokalne pretrage, odnosno oba susjedstva. Dobiveno rješenje više nije moguće poboljšati lokalnom pretragom stoga se koristi modifikacija rješenja s ciljem izlaska iz lokalnog optimuma.

Lokalna pretraga ubacivanjem posla istražuje susjedstvo dobiveno na sljedeći način. Za svaki posao isprobavaju se sve moguće pozicije na drugim strojevima te se bira ona s najmanjom vrijednosti funkcije dobrote. Tako nastaje jedan susjed. Budući da se umetanje radi za svaki posao, ukupno nastaje n susjeda. Ako je bar jedan od n susjeda bolji od trenutnog rješenja, on se postavlja za trenutno rješenje i odlazi se u novu iteraciju. Lokalna pretraga staje kada je nađen lokalni optimum, odnosno kada niti jedan susjed nije bolji od trenutnog rješenja. Opisani postupak prikazan je algoritmom 15 preuzetim iz Fanjul-Peyro i Ruiz (2010). U ovom i daljnjim algoritmima, funkcija $f(s)$ predstavlja funkciju dobrote rješenja s.

Algoritam 15 InsertionLocalSearch

Input: s

1: $improved \leftarrow true$
2: **while** $improved$ **do**
3: $improved \leftarrow false$
4: **for all** $j \in n$ **do**
5: $i \leftarrow$ stroj na kojem se nalazi posao j
6: pronađi stroj l i na njemu poziciju pos gdje ubacivanje posla j daje najmanju vrijednost funkcije dobrote, $l \neq i$
7: $s' \leftarrow$ susjed nastao ubacivanjem posla j na stroj l na poziciju pos rasporeda s
8: **if** $f(s') < f(s)$ **then**
9: $improved \leftarrow true$
10: $s \leftarrow s'$
11: **return** najbolji pronađeni raspored (s)

Druga implementirana lokalna pretraga generira susjede zamjenom dva posla. Zamjene poslova koje se razmatraju uključuje sve kombinacije dva posla s različitih strojeva. Kako je ovo susjedstvo veće od onog dobivenog umetanjem poslova, može se koristiti ubrzanje koje ne traži najbolju zamjenu. U tom slučaju prihvaća se prva zamjena koja poboljšava raspored i odlazi se u novu iteraciju, odnosno na novi posao. Implementirana procedura prikazana je algoritmom 16. Algoritam prikazuje postupak bez ubrzanja.

Implementacija IG modifikacije implementirana je jednako kao perturbacija prikazana algoritmom 10. RLS modifikacija pretražuje manja, ograničena susjedstva. Za razliku od IG modifikacije, ovdje se uzima jedan po jedan posao i prebacuje na neki drugi stroj. Implementirane su dvije modifikacije, *No Same Place* (NSP) i *Virtual* (VIR). Razlika između NSP i VIR je to što je u VIR metodi omogućeno umetanje posla na isti stroj. NSP pretražuje umetanja posla na sve strojeve osim onog na kojem se posao nalazi. Ulazna vrijednost za obje metode je parametar d koji određuje koliko poslova će biti premješteno. VIR i NSP su prikazani algoritmom 17, preuzetim iz Fanjul-Peyro i Ruiz (2010).

Algoritam 16 InterchangeLocalSearch

Input: s

1: $improved \leftarrow true$
2: **while** $improved$ **do**
3: $improved \leftarrow false$
4: **for all** $j \in n$ **do**
5: $i \leftarrow$ stroj na kojem se nalazi posao j
6: na stroju različitom od i pronađi posao j' čijom se zamjenom s poslom j dobiva bolji raspored
7: $s' \leftarrow$ susjed nastao zamjenom poslova j i j' na rasporedu s
8: **if** $f(s') < f(s)$ **then**
9: $improved \leftarrow true$
10: $s \leftarrow s'$
11: **return** najbolji pronađeni raspored (s)

Algoritam 17 NSP/VIR

Input: s, d

1: **for all** $k \in d$ **do**
2: $i \leftarrow$ nasumično odabran stroj
3: $j \leftarrow$ nasumično odabran posao sa stroja i
4: **if** algoritam NSP **then**
5: $l \leftarrow$ pronađi stroj $l \neq i$ i na njemu poziciju pos gdje ubacivanje posla j daje najmanju vrijednost funkcije dobrote
6: **if** algoritam VIR **then**
7: $l \leftarrow$ pronađi stroj l i na njemu poziciju pos gdje ubacivanje posla j daje najmanju vrijednost funkcije dobrote
8: **return** najbolji pronađeni raspored (s^*)

Heuristika je prikazana algoritmom 18, preuzetim iz Fanjul-Peyro i Ruiz (2010). Lokalne pretrage ubacivanjem poslova i zamjenom poslova implementirane su redom funkcijama $insertionLocalSearch()$ i $interchangeLocalSearch()$. Funkcija $solutionModification()$ predstavlja jednu od opisanih modifikacija, IG, NSP ili VIR.

Algoritam 18 VND

$s \leftarrow$ inicijalno rješenje
$s^* \leftarrow$ najbolje pronađeno rješenje
while $!stoppingCriteria$ **do**
    $improved \leftarrow true$
    while $improved$ **do**
        $improved \leftarrow false$
        $s' \leftarrow insertionLocalSearch(s)$
        $s'' \leftarrow interchangeLocalSearch(s')$
        if $s'' \neq s'$ **then**
            $improved \leftarrow true$
            $s \leftarrow s''$
    if $f(s'') \leq f(s^*)$ **then**
        $s^* \leftarrow s''$
    $s \leftarrow solutionModification(s^*)$
return najbolji pronađeni raspored (s^*)

3.3. Metaheuristike

Metaheuristika (Glover i Kochenberger (2003), Talbi (2009)) je skup algoritamskih koncepata koji se koristi za definiranje heurističkih metoda primjenjivih na širok skup problema. Može se reći da je metaheuristika heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja (Dorigo et al. (2006), Čupić (2012)).

U ovom radu su korištene tri metaheuristike - optimizacija kolonijom mrava, simulirano kaljenje i tabu pretraživanje. Cilj je ustanoviti efikasnost jednostavnih, općenitih metaheuristika u usporedbi s heuristikama posebno izrađenim za rješavanje problema raspoređivanja.

3.3.1. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava je metaheuristika inspirirana ponašanjem mrava u prirodi, konkretno njihovom sposobnošću da pronađu najkraći put do izvora hrane. Kretanjem kroz prostor mravi ostavljaju za sobom feromonski trag koji služi drugim mravima kako bi ih mogli slijediti. U slučaju da se pred mravima nalaze dva puta, jedan duži od drugog, inicijalno će polovica mrava ići jednim putem, druga polovica drugim putem. Feromoni koje ostavljaju za sobom isparavaju. Na dužem putu, proces isparavanja odvija se brže. Kako mravi instinktivno slijede miris feromona, više mrava će privući upravo kraći put, s više feromona. Tako će nakon nekog vremena većina mrava slijediti isti trag feromona odnosno kraći put.

Postupak optimizacije prilagođen je problemu raspoređivanja slično kao u radu Arnaout et al. (2009). Problem se može opisati povezanim grafom u kojem bridovi predstavljaju vrijednosti feromona. Svaki mrav prolazi put od praznog do potpunog rasporeda tako da u svakom čvoru donosi odluku o raspoređivanju temeljenu na vjerojatnosti. Vjerojatnosti se računaju pomoću feromona i vidljivosti. Feromoni sadrže informaciju o dobroti puta. Bolji put je onaj koji rezultira rasporedom s manjom vrijednosti funkcije dobrote. Vidljivost je pohlepna heuristika, na primjer dodjeljivanje posla onom stroju na kojem ima najkraće vrijeme izvođenja. Opcionalno, nakon izgradnje rasporeda može se provesti lokalna pretraga za dodatno poboljšavanje rješenja. Nakon prolaska svakog mrava ažurira se vrijednost feromona kako bi se simuliralo isparava-

nje. Nakon što svi mravi završe, uzima se najbolji mrav (onaj koji je izgradio raspored s najmanjom vrijednosti funkcije dobrote) i ažuriraju feromoni. Ažuriranje feromona za najboljeg mrava povećava vrijednosti samo onih feromona koji se nalaze na putu kojim je najbolji mrav izgradio raspored. Na početku procesa sve vrijednosti feromona postavljaju se na neku početnu vrijednost i kroz svaku iteraciju smanjuju se isparavanjem i povećavaju za dobra rješenja. Konkretna implementacija i formule za računanje vjerojatnosti i ažuriranje feromona opisani su u nastavku.

Svaki mrav prolazi kroz dvije faze. Prva faza određuje na kojem stroju će se nalaziti pojedini posao. Druga faza određuje permutaciju poslova, odnosno redoslijed kojim će se poslovi stavljati na strojeve određene prvom fazom. Produkt prve faze je vektor S_1, a druga faza gradi vektor S_2. Za primjer se može uzeti problem s 12 poslova i 3 stroja. Poslovi su numerirani od 0 do 11, a strojevi od 0 do 2. Indeksi vektora S_1 predstavljaju poslove, a vrijednosti odgovaraju strojevima. Iz vektora $S_1 = [0, 1, 1, 0, 0, 2, 2, 1, 0, 1, 2, 0]$ vidi se da su poslovi 0, 3, 4, 8 i 11 raspoređeni na stoj 0, poslovi 1, 2, 7 i 9 na stroj 1 i poslovi 5, 6 i 10 na stroj 2. Vektorom $S_2 = [10, 4, 7, 2, 8, 9, 5, 6, 1, 11, 0, 3]$ određen je redoslijed poslova na strojevima. Prvi posao koji se raspoređuje je 10 na stroj 2, zatim posao 4 na stroj 0, 7 na 1, 2 na 1 i tako dalje. Konačni raspored s je:

0:	4	8	11	0	3
1:	7	2	9	1	
2:	10	5	6		

U obje faze mrav gradi vektore S_1 i S_2 pomoću feromona. Za prvu fazu koristi se matrica koja ima m redaka i n stupaca (oznaka τ^I). Svaki posao (stupac) može se rasporediti na jedan od strojeva (retci). Vrijednosti u matrici određuju naklonost raspoređivanja posla po strojevima. Druga faza također koristi dvodimenzionalnu strukturu s n redaka i n stupaca (oznaka τ^{II}). Retci predstavljaju poslove, a stupci pozicije u vektoru permutacija S_2. Pomoću ove matrice određuje se kojim redoslijedom je najbolje rasporediti poslove po strojevima. Osim feromona koristi se i pohlepna heuristika - vidljivost. Implementirana je samo za prvu fazu i obrnuto je proporcionalna vremenu izvođenja posla na stroju. U prvoj fazi, računanje vjerojatnosti odabira stroja za pojedini posao određeno je formulom 3.3 (Arnaout et al. (2009)).

$$\Pi_{i,j}^{I} = \frac{\left(\tau_{i,j}^{I}\right)^{\alpha} * \left(\eta_{j,i}\right)^{\beta}}{\sum_{l=1}^{m} \left(\tau_{l,j}^{I}\right)^{\alpha} * \left(\eta_{j,l}\right)^{\beta}} \tag{3.3}$$

Oznaka i predstavlja stroj, j označava posao, τ je oznaka za feromone, a η za vidljivost. Parametri $\alpha \in [0, 1]$ i $\beta \in [0, 1]$ određuju utjecaj feromona naspram vidljivosti i time usmjeravaju pretragu. Zbroj parametara α i β treba biti 1. Vidljivost η određena je formulom 3.4 (Arnaout et al. (2009)):

$$\eta_{j,i} = \frac{1}{p_{j,i}} \tag{3.4}$$

pri čemu $p_{j,i}$ predstavlja vrijeme izvođenja posla j na stroju i.

U drugoj fazi, računanje vjerojatnosti odabira pozicije posla u vektoru S_2 određeno je formulom 3.5:

$$\Pi_{j,i}^{II} = \frac{\tau_{j,i}^{II}}{\sum_{l=1}^{\psi} \tau_{l,i}^{II}} \tag{3.5}$$

gdje ψ označava neraspoređene poslove, i označava poziciju, a j posao. Kada se posao rasporedi potrebno ga je izbaciti iz daljnjih odabira kako ne bi došlo do višestrukog raspoređivanja istog posla. Stoga formula 3.5 za svaku poziciju i razmatra samo trenutno neraspoređene poslove.

Nakon definiranja svih struktura, parametara i formula potrebnih za izgradnju rasporeda, ostaju još formule koje određuju ažuriranje feromona. Nakon svakog mrava dolazi do isparavanja obje strukture feromona po formulama 3.6 i 3.7.

$$\tau_{i,j}^{I} = (1 - \rho) * \tau_{i,j}^{I} \tag{3.6}$$

$$\tau_{j,i}^{II} = (1 - \rho) * \tau_{j,i}^{II} \tag{3.7}$$

Parametar ρ je konstanta koja određuje brzinu isparavanja.

Nakon što svi mravi izgrade rasporede, uzima se najbolji mrav i ažuriraju se feromoni po formulama 3.8, 3.9, 3.10 i 3.11.

$$\tau_{i,j}^{I} = \tau_{i,j}^{I} + \Phi * \Delta\tau_{i,j}^{I,Best} \tag{3.8}$$

$$\tau_{j,i}^{II} = \tau_{j,i}^{II} + \Phi * \Delta\tau_{j,i}^{II,Best} \tag{3.9}$$

$$\Delta\tau_{i,j}^{I,Best} \begin{cases} \frac{1}{TWT^{Best}}, & \text{ako je najbolji mrav rasporedio posao } j \text{ na stroj } i \\ 0, & \text{inače} \end{cases} \tag{3.10}$$

$$\Delta\tau_{j,i}^{II,Best} \begin{cases} \frac{1}{TWT^{Best}}, & \text{ako je najbolji mrav stavio posao } j \text{ na poziciju } i \\ 0, & \text{inače} \end{cases} \tag{3.11}$$

U formulama 3.8 i 3.9 parametar Φ je konstanta koja određuje intenzifikaciju dobrih puteva u izgradnji rasporeda. Oznake $\Delta\tau_{i,j}^{I,Best}$ i $\Delta\tau_{j,i}^{II,Best}$ odnose se na najboljeg mrava (onog koji je izgradio raspored s najmanjom vrijednosti funkcije dobrote). Feromoni se povećavaju za iznos obrnuto proporcionalan vrijednosti funkcije dobrote najboljeg mrava (TWT^{Best}) i to samo za one korake (odabrane poslove, strojeve i pozicije) koje je napravio najbolji mrav. Ovime se postiže da bolja rješenja više utječu na vrijednosti feromona.

Lokalna pretraga integrirana je u optimizaciju kolonijom mrava kako bi dodatno poboljšala rješenja koja izgrađuju mravi. Postupak lokalne pretrage implementiran je kao u radu Arnaout et al. (2009). Dodatno je definiran je parametar k koji određuje koji udio mrava će nakon izgradnje rasporeda provoditi lokalnu pretragu. Za najboljeg mrava, lokalna pretraga se uvijek provodi. U svakoj iteraciji lokalne pretrage s jednakom vjerojatnosti odabire se S_1 ili S_2 i generira jedan susjed. Susjed od S_1 nastaje tako da se za mali udio poslova promijene strojevi. Nasumično se odabere oko 5% poslova i na njima se nasumično bira stroj različit od trenutnog. Susjed od S_2 nastaje zamjenom dva nasumično odabrana posla na istom stroju. Nakon izvršavanja jedne dviju opisanih promjena, provjerava se je li raspored nakon promjena bolji od trenutnog. Ako je, ta promjena se provodi nad trenutnom vrijednosti S_1 ili S_2. Algoritam

vraća najbolje pronađene S_1 i S_2 (one koji određuju raspored s najmanjom vrijednosti funkcije dobrote) u zadanom broju iteracija.

Cjelokupan postupak optimizacije kolonijom mrava uz dodanu optimizaciju lokalnom pretragom prikazan je algoritmima 19 i 20.

Algoritam 19 OptimizacijaKolonijomMrava

Input: $\tau^I, \tau^{II}, \phi, \rho, \alpha, ants, k, iterLS$

postavi feromone na početne vrijednosti (τ^I, τ^{II})
$\beta \leftarrow 1 - \alpha$
$antsLS \leftarrow$ za koliko mrava će se provoditi lokalna pretraga ($k * ants$)
while !$stoppingCriteria$ **do**
    for all $ant \in ants$ **do**
        $S_1 \leftarrow$ izračunaj vjerojatnosti po formuli 3.3 i uniformno odaberi jedan stroj za svaki posao
        $S_2 \leftarrow$ izračunaj vjerojatnosti po formuli 3.5 i uniformno odaberi jedan od neraspoređenih poslova za svaku poziciju
        if $ant <= antsLS$ **then**
            provedi lokalnu pretragu za S_1 i S_2
        ažuriraj feromone po formulama 3.6 i 3.7
        $s^* \leftarrow$ ažuriraj najbolji pronađeni raspored
        $bestS_1$, $bestS_2 \leftarrow$ ažuriraj najboljeg mrava u trenutnoj iteraciji
    provedi lokalnu pretragu za $bestS_1$ i $bestS_2$
    ažuriraj feromone po formulama 3.8, 3.9, 3.10 i 3.11
    $s^* \leftarrow$ ažuriraj najbolji pronađeni raspored
return najbolji pronađeni raspored (s^*)

Algoritam 20 LokalnaPretraga

Input: $S_1, S_2, iterLS$

$i \leftarrow 0$
while $i < iterLS$ **do**
    $rv \leftarrow$ nasumično generiran broj iz $U(0, 1)$
    if $rv < 0.5$ **then**
        generiraj susjeda od S_1 zamjenom 5% vrijednosti
    else
        generiraj susjeda od S_2 zamjenom dva posla na istom stroju (koristeći informacije iz S_1)
    $f_{new} \leftarrow$ vrijednost funkcije dobrote za novi raspored
    $f_{current} \leftarrow$ dobrota trenutnog rasporeda
    if $f_{new} < f_{current}$ **then**
        zamijeni S_1 ili S_2 susjedom
    $i \leftarrow i + 1$
return S_1, S_2

3.3.2. Simulirano kaljenje

Simulirano kaljenje je postupak predložen od Kirkpatrick et al. (1983), motiviran kaljenjem metala. Ova metaheuristika radi s jednim rješenjem nad kojim se u svakoj iteraciji pretražuje susjedstvo. Ako je pronađeno bolje rješenje (s manjom vrijednosti funkcije dobrote) ono se automatski prihvaća za trenutno rješenje. U suprotnom, računa se vjerojatnost prihvaćanja lošijeg rješenja definirana formulom 3.12:

$$P(s') = min(1, e^{\frac{f(s)-f(s')}{T}}) \tag{3.12}$$

gdje funkcija f predstavlja funkciju dobrote, s je trenutno i s' novo rješenje, a parametar T označava temperaturu. Ovakav način prihvaćanja rješenja osigurava kretanje prema prostoru boljih rješenja ako su pronađena, ali i pomaže u izbjegavanju lokalnih optimuma povremenim prihvaćanjem lošijih rješenja. Vjerojatnost prihvaćanja novog rješenja je veća što je razlika između vrijednosti dobrota trenutnog i novog rješenja manja. Također, vjerojatnost se smanjuje kako se temperatura T smanjuje. Početna vrijednost temperature T postavlja se na veću vrijednost čime se postiže diverzifikacija na početku pretrage. Temperatura se kroz postupak smanjuje i time se postiže intenzifikacija, odnosno pretraga prostora oko najboljeg rješenja. Proces smanjivanja temperature kroz iteracije, ili plan hlađenja, može se odvijati brže ili sporije. Postoji nekoliko različitih planova hlađenja koji se najčešće koriste. U ovom radu implementiran je plan hlađenja prikazan formulom 3.13:

$$T_k = T_0 * \alpha^k \tag{3.13}$$

gdje parametar k predstavlja trenutni korak algoritma, odnosno iteraciju.

Lokalna pretraga koja je korištena u procesu simuliranog kaljenja je prilagodba VNS metode. Prvo se nad trenutnim rješenjem bira nasumični susjed. Drugi korak je izgradnja cijelog susjedstva nad odabranim nasumičnim rješenjem te vraćanje najboljeg susjeda - onog s najmanjom vrijednosti funkcije dobrote. Prvi korak nasumično bira jedan od tri načina generiranja nasumičnog susjeda, isto kao kod VNS postupka (3.2.4). Ukratko, prvi način umeće nasumičan posao na nasumičan stroj, drugi radi zamjenu bilo koja dva posla na istom stroju, i treći postupak radi zamjenu bilo koja dva posla na različitim strojevima u rasporedu. Nad selektiranim rješenjem gradi se susjedstvo

kojeg čine sva rješenja dobivena zamjenom bilo koja dva posla u rasporedu. Najbolje rješenje iz susjedstva se prihvaća ako je bolje od trenutnog. Ako nije pronađeno bolje rješenje, s vjerojatnosti određenom formulom 3.12 prihvaća se nasumično odabran susjed iz prvog koraka. Prije odlaska u iduću iteraciju, smanjuje se temperatura po implementiranom planu hlađenja. Opisani postupak prikazan je algoritmom 21. Prije početka iterativnog postupka generira se početno rješenje. Implementirana su dva načina - nasumično i pomoću heuristike ATC. Kriteriji zaustavljanja su zadan broj iteracija i/ili vremensko ograničenje.

Algoritam 21 SimuliranoKaljenje

Input: T_0, α

$s \leftarrow$ generiraj početno rješenje (nasumično ili ATC)
$T \leftarrow T_0$
while $!stoppingCriteria$ **do**
    $randomNeighbour \leftarrow$ nasumično odabran susjed
    $bestNeighbour \leftarrow$ najbolji susjed iz susjedstva od $randomNeighbour$
    if $f(bestNeighbour) < f(s)$ **then**
        $s \leftarrow bestNeighbour$
    else
        $p \leftarrow min(1, e^{\frac{f(s)-f(randomNeighbour)}{T}})$
        $rv \leftarrow$ nasumično generiran broj iz $U(0,1)$
        if $rv \leq p$ **then**
            $s \leftarrow randomNeighbour$
    $T \leftarrow T * \alpha$
return najbolje pronađeno rješenje (s)

3.3.3. Tabu pretraživanje

Tabu pretraživanje je metaheuristika predložena od Glover (1990), popularna za rješavanje raznih optimizacijskih problema. Radi s jednim rješenjem tako da krene od početnog i u svakoj iteraciji bira najbolje rješenje iz susjedstva. Kako bi se spriječilo kruženje, odnosno vraćanje u već posjećena rješenja, pamti se lista zadnjih l posjećenih rješenja. U svakoj iteraciji pri odabiru novog trenutnog rješenja potrebno je provjeriti nalazi li se ono u listi zabranjenih rješenja, kraće tabu listi. Vrijeme zadržavanja rješenja u tabu listi određeno je veličinom liste (parametar l). Tabu lista može pamtiti cijela

rješenja ili njihove atribute poput izvršenih poteza koji mijenjaju rješenje. Postupak tabu pretraživanja sadrži nekoliko različitih načina generiranja susjedstva. Nakon evaluacije svih susjeda dobivenih metodama generiranja susjedstva, bira se najbolji susjed koji nije tabu, čak i ako je lošiji od trenutnog rješenje što omogućava izlazak iz lokalnih optimuma. Ako je pronađen susjed bolji od trenutnog rješenja, on se prihvaća neovisno o tabu listi.

Implementacija postupka tabu pretraživanja implementirana je kao u radu Lee et al. (2013). Inicijalno rješenje generira se pomoću heuristike ATC. U svakoj iteraciji nad trenutnim rješenjem generira se osam susjeda kako je opisano u nastavku:

1. **Umetanje grupe poslova**
   Nasumično se bira grupa poslova (niz uzastopnih poslova) sa stroja koji ima najveću vrijednost funkcije dobrote te umeće na nasumično odabranu poziciju na stroju s najmanjom vrijednosti funkcije dobrote

2. **Premještanje posla**
   Nasumično se bira posao sa stroja koji ima najveću vrijednost funkcije dobrote te umeće na nasumično odabranu poziciju na stroju s najmanjom vrijednosti funkcije dobrote

3. **Lančano umetanje grupa poslova**
   Nasumično se bira grupa poslova sa stroja koji ima najveću vrijednost funkcije dobrote te umeće na nasumično odabranu poziciju na nasumično selektirani stroj (posrednik). Sa stroja posrednika se nasumično bira grupa poslova i umeće na nasumično odabranu poziciju na stroju s najmanjom vrijednosti funkcije dobrote.

4. **Lančano umetanje grupe poslova i posla**
   Ova metoda je jednaka lančanom umetanju grupa poslova s razlikom da se sa stroja posrednika premješta jedan posao, a ne grupa poslova.

5. **Lančano umetanje poslova**
   Ova metoda je jednaka lančanom umetanju grupa poslova s razlikom da se premještaju pojedinačni poslovi (prvo sa stroja s najvećom vrijednosti funkcije dobrote na posrednički stoj, pa s njega na stoj s najmanjom vrijednosti funkcije dobrote).

6. **Zamjena grupa poslova**
   Nasumično odabrana grupa poslova sa stroja s najvećom vrijednosti funkcije dobrote zamjenjuje se s nasumično odabranom grupom poslova s nasumično selektiranog stroja, različitog od onog s najvećom vrijednosti funkcije dobrote.

7. **Zamjena poslova**
   Ova metoda je ista kao zamjena grupa poslova s time da se zamjenjuju pojedinačni poslovi. Ovdje se također zamjena radi na dva različita stroja.

8. **Zamjena grupa poslova na istom stroju**
   Radi se zamjena dva nasumično odabrana posla na jednom nasumično odabranom stroju.

Za svakog susjeda pamte se potezi, odnosno promjene u odnosu na originalno rješenje. Kada se susjed prihvaća za trenutno rješenje (ono za koje se pretraga nastavlja u idućoj iteraciji) odgovarajući potezi stavljaju se u tabu listu. Pri provjeri je li neki susjed tabu, uspoređuju se potezi vezani za njega sa svim potezima u tabu listi. Ako se dovoljno poteza poklapa, susjed je tabu i ne prihvaća se za trenutno rješenje. Broj poteza koji se trebaju poklopiti, odnosno biti isti, zadan je parametrom $p \in (0, 1]$. Jedan potez pamti posao koji je promijenio poziciju, stroj na koji je premješten i poziciju na koju je stavljen. Svih 8 metoda koje generiraju susjede za svaki posao koji mijenja svoju poziciju pamte potez s navedenim podatcima.

Postupak tabu pretraživanja prikazan je algoritmom 22. Kriterij zaustavljanja je određen brojem iteracija ili vremenskim ograničenjem. Ulazni parametar l određuje veličinu tabu liste. Pri ažuriranju tabu liste, ako je lista puna koristi se princip FIFO (*First In First Out*). Pri određivanju je li raspored tabu, koristi se parametar p koji određuje koliki postotak poteza vezanih za susjeda mora biti u tabu listi da bi on bio tabu, odnosno da se ne bi mogao postaviti za trenutno rješenje. Počevši od susjeda s najmanjom vrijednosti funkcije dobrote, bira se prvi koji nije tabu. U slučaju da je prvi susjed bolji od trenutnog rješenja, uzima se za trenutno rješenje bez obzira na tabu listu.

Algoritam 22 TabuPretraživanje

Input: l, p

$s \leftarrow$ inicijalno rješenje pomoću ATC heuristike
while $!stoppingCriteria$ **do**
    generiraj i susjede i odgovarajuće poteze iz 8 opisanih susjedstva
    $neighbourhood \leftarrow$ sortiraj susjede po rastućoj vrijednosti funkcije dobrote
    $bestNeighbour \leftarrow neighbourhood[0]$
    if $f(bestNeighbour) < f(s)$ **then**
        $s \leftarrow bestNeighbour$
        ažuriraj tabu listu dodavanjem poteza od $bestNeighbour$
    else
        $s \leftarrow$ prvi susjed iz $neighbourhood$ koji nije tabu
        ažuriraj tabu listu dodavanjem poteza od odabranog susjeda
return najbolje pronađeno rješenje (s)

4. Rezultati

Rješenja su razvijena u jeziku C++ koristeći neke funkcionalnosti programskog okvira ECF (Evolutionary Computing Framework). Implementirani postupci uspoređeni su s postojećim genetskim algoritmima i pravilima raspoređivanja na problemima različitih dimenzionalnosti. Korišteno je 60 instanci problema s različitim brojevima poslova i strojeva. Brojevi poslova su 12, 25, 50 i 100, a brojevi strojeva su 3, 6 i 10. Konkretne instance sastoje se od svih kombinacija brojeva poslova i strojeva, osim 10 strojeva i 12 poslova. Rezultati dobiveni za iscrpnu pretragu bili su vremenski ograničeni na 24 sata za svaku instancu problema. Ostale metode pokrenute su s vremenskim ograničenjem od 5 minuta za sve instance, odnosno 5 sekundi za svaku instancu. Svaka metoda pokrenuta je 30 puta na svim instancama. Izlazna vrijednost algoritma za neku instancu problema je najbolji raspored, odnosno najmanja vrijednost funkcije dobrote. Rezultat jednog pokretanja je suma po svim instancama problema. Konačno je dobiveno 30 suma za svaku metodu te se nad njima radila usporedba. Neke metode imaju implementirano više načina generiranja početnog rješenja, od kojih je jedan zajednički - ATC. Te metode su pokrenute s i bez korištenja ATC-a. Sve metode uspoređivat će se s ATC pravilom i genetskim algoritmom koji je također pokrenut unutar istog vremenskog ograničenja od 5 minuta za sve instance.

U daljnjem tekstu i tablicama korištene su kratice: ACO za optimizaciju kolonijom mrava, SA za simulirano kaljenje, TS za tabu pretraživanje, GA za genetski algoritam i ostale kratice uvedene u prethodnom poglavlju.

4.1. Rezultati iscrpne pretrage

Rezultati iscrpne pretrage uspoređeni su s rezultatima genetskog algoritma. Algoritam branch and bound nije uspio pronaći rješenja bolja od genetskog algoritma. Svi re-

zultati su ili jednako dobri kao oni od genetskog algoritma ili nisu uopće pronađeni. Kako se postupak pretrage prekida nakon 24 sata, u nekim slučajevima ne uspije se izgenerirati niti jedan kompletan raspored. To se dogodilo za 20 od 60 instanci problema, najviše za instance sa 6 strojeva i 100 poslova. Za ostalih 40 instanci pronađena su rješenja, od toga je 9 instanci prekinuto nakon 24 sata i uzeto je do tada najbolje nađeno rješenje.

Optimalno rješenje pronađeno je za 31 instancu, odnosno za $51.\dot{6}\%$ instanci problema. Za 15% instanci nisu obiđene sve kombinacije pa se ne može znati je li pronađeno rješenje optimalno i za $33.\dot{3}\%$ rješenje uopće nije pronađeno. Iz ovih rezultata očit je nedostatak iscrpne pretrage u odnosu na genetski algoritam. Za veće instance problema ne može se očekivati da će iscrpna pretraga uspjeti naći optimalno rješenje, bez obzira na implementirane optimizacije poput istraživanja prvo prostora boljih rješenja ili korištenja gornje granice. Svi pronađeni optimumi jednaki su rezultatima genetskog algoritma što pokazuje koliko je zapravo genetski algoritam dobar. Također, i za onih 15% instanci koje se nisu uspjele izvršiti do kraja, pronađeno rješenje je jednako rezultatu genetskog algoritma. To je posljedica toga što iscrpno pretraživanje počinje od prostora rješenja u kojem je genetski algoritam pronašao najbolje rješenje.

4.2. Optimizacija parametara algoritama

Prije pokretanja heurističkih algoritama, potrebno je odrediti što bolje vrijednosti ulaznih parametara. Postupak pronalaska optimalnih parametara izvodi se na jedan od dva načina. Prvi, jednostavniji način je definiranje nekoliko vrijednosti jednog parametra i fiksiranje ostalih parametara. Algoritam se onda pokreće 10 puta za svaku vrijednost na svim instancama. U jednom pokretanju sumira se svih 60 vrijednosti funkcija dobrote. Rezultat 10 pokretanja je 10 suma. Za svaku vrijednost parametra tada je vezano 10 suma i bira se ona vrijednost parametra koja ima najmanji medijan među sumama. Drugi način pretraživanja nad prostorom parametara je pretraživanje po rešetci. Ovaj postupak je dugotrajan i izveden samo za optimizaciju kolonijom mrava nad nekoliko parametara koji su međusobno zavisni.

Vrijednosti parametara nad kojima se radi pretraga odabrane su tako da pokrivaju rubne vrijednosti, a između njih su odabrane one koje su bile korištene u izvornoj literaturi ili intuitivno imaju smisla. Parametri korišteni pri pokretanju prikazani su tablicom 4.1.

Tablica 4.1: Parametri algoritama korišteni pri pokretanju

Metoda	Parametri			
ATC	$k = 0.05$			
GRASP	$\alpha = 1$			
GRASP+ATC	$\alpha = 1$			
GRASP+PR	$\alpha = 0.9$	$esize = 0.0\dot{6}$		
GRASP+PR+ATC	$\alpha = 1$	$esize = 0.\dot{6}$		
GRASP+PR+ILS	$\alpha = 1$	$esize = 0.\dot{6}$	$d = 0.125$	$iterILS = 75$
GRASP+PR+ILS+ATC	$\alpha = 1$	$esize = 0.\dot{6}$	$d = 0.125$	$iterILS = 75$
VND+IG	$d = 0.05$			
VND+NSP	$d = 0.0\dot{6}$			
VND+VIR	$d = 0.1$			
SA	$\alpha = 0.5$	$T_0 = 750$		
SA+ATC	$\alpha = 0.5$	$T_0 = 200$		
TS	$l = 90$	$p = 0.1$		
ACO	$\rho = 0.01$	$\phi = 0.7$	$iterLS = 20$	$\tau^I = 0.1$
	$\alpha = 0.8$	$k = 1$	$ants = 10$	$\tau^{II} = 0.1$

4.3. Rezultati heuristika i metaheuristika

Sve implementirane metode pokrenute su korištenjem ranije definiranih vrijednosti parametara s vremenskim ograničenjem od 5 minuta za svaku metodu. Cilj je bio svim algoritmima postaviti isti kriterij zaustavljanja i na temelju toga provesti međusobnu usporedbu i usporedbu s genetskim algoritmom.

Svi prikazani rezultati u ovom poglavlju dobiveni su sumiranjem izlaza algoritma po svim instancama problema, pri čemu je izlaz algoritma najmanja pronađena vrijednost funkcije dobrote. Za usporedbu koristite se sljedeće dvije vrijednosti:

Genetski algoritam: 9.43140634

ATC: 13.3821

Genetski algoritam je kompleksna i dugotrajna metoda koja daje jako dobre rezultate. Međutim, u nekim situacijama bitno je dobiti rezultate u kratkom vremenu. Tada je manje bitno jesu li oni optimalni, ali naravno teži se tome da budu što bolji. Proces iscrpne pretrage može dati optimalna rješenja, no kako se pokazalo to nije isplativo jer proces traje predugo. Za slučajeve kada je prioritet pronalazak rješenja u kratkom vremenu, koriste se pravila raspoređivanja, heuristike i metaheuristike. Najkraće vrijeme izvođenja potrebno je za pravila raspoređivanja jer grade samo jedan raspored. Primjer je ATC, koji na konkretnim instancama problema daje sumu 13.3821. Heuristike i metaheuristike se razlikuju po tome što kroz iterativni proces poboljšavaju početno rješenje. Te metode bi trebale dati rješenje bolje od pravila ATC, i pokušati se približiti rezultatu genetskog algoritma. Očekivano je stoga da se rezultati nalaze između vrijednosti 9.4 i 13.4. Naravno, to je pod pretpostavkom da sve implementirane metode rade dobro na konkretnom problemu kojim se bavi ovaj rad. No, neke korištene metode originalno su namijenjene drugačije definiranim problemima statičkog raspoređivanja, stoga je bilo potrebno prilagoditi implementaciju. Za jednostavnije metode to nije bio slučaj. Jedna od nadogradnji kompleksnijih postupaka je ta da se za početno rješenje uzima raspored generiran po pravilu ATC i od njega kreće pretraga prema boljim rješenjem.

Tablica 4.2 prikazuje rezultate od 30 pokretanja za svaku metodu, pri čemu je trajanje jednog pokretanja ograničeno na 5 minuta. Rezultati su u obliku suma vrijednosti funkcija dobrota po svih 60 instanci problema. Za svaku metodu zabilježeni su minimum, medijan i maksimum u 30 pokretanja. Dodatno, stupac *best* prikazuje sumu

Tablica 4.2: Usporedba rezultata po metodama

metoda	min	med	max	best
GA	9.5278	9.6772	9.9493	9.4314
VND+IG	10.4000	10.7483	11.2046	9.8218
VND+VIR	10.5048	10.8245	11.1249	10.0155
VND+NSP	13.2499	13.7840	14.5203	12.2617
VNS	9.8035	10.1437	10.6849	9.4639
GRASP+ATC	11.6852	11.8005	11.9383	11.5929
GRASP	110.1150	116.5735	121.2370	89.9420
GRASP+PR+ATC	11.1970	11.2700	11.3247	11.1589
GRASP+PR	25.1047	27.1770	28.8285	20.6035
GRASP+PR+ILS+ATC	9.5905	9.6601	9.7759	9.4691
GRASP+PR+ILS	9.8316	9.9342	10.3283	9.5146
SA	9.5759	9.7097	9.9366	9.4737
SA+ATC	9.5274	9.6165	9.8831	9.4475
TS	12.9549	13.2244	13.2892	12.5611
ACO	14.1345	15.7891	16.7021	11.6569

koja se dobije ako se za svaku instancu uzme najmanja vrijednost funkcije dobrote među svim pokretanjima. Prvi redak tablice prikazuje rezultate za genetski algoritam koji služe za usporedbu sa svim implementiranim metodama. Slika 4.1 prikazuje raspodjelu dobivenih suma u svim pokretanjima za sve metode osim verzija GRASP-a i GRASP-a uz PR koji ne koriste ATC.

VND

Metoda VND ima tri implementirane modifikacije. Postupci IG i VIR daju slične rezultate, dok je NSP lošiji. Razlika između rezultata od IG i VIR je mala. IG ima manji medijan i manje vrijednosti suma za 50% pokretanja što se može vidjeti iz grafa. Medijani za IG i VIR iznose 10.74 i 10.82 dok je medijan za NSP 13.78. Ova razlika može se objasniti time što je NSP modifikacija više ograničena. Konkretno, zabranjeno je umetanje posla na isti stroj. Ovakva modifikacija ima više smisla kada nema ograničenja dolaska poslova u sustav i zadanih rokova do kada poslovi trebaju biti izvršeni.

Ako su poslovi raspodijeljeni na dobar stroj, ali poredak nije optimalan, postupci IG i VIR to mogu ispraviti. IG pokazuje bolje rezultate jer radi veće promjene nad rasporedom. Nasumično uklanja pa pohlepno stavlja poslove u raspored, dok VIR premješta jedan po jedan posao. Zbog toga je raspon rezultata veći za VND uz modifikaciju IG. U usporedbi s genetskim algoritmom, rezultati su prihvatljivi, no iz eksperimenata se može vidjeti da neke druge metode poput VNS-a, GRASP-a uz PR i ILS te simuliranog kaljenja daju bolje rezultate. Također, u usporedbi s pravilom ATC, rezultati su bolji što je i očekivano, uz iznimku VND-a uz NSP. Ako se u tablici 4.2 promatra stupac *best*, VND uz IG je blizu rezultatu genetskog algoritma.

Može se zaključiti da postupak VND daje prihvatljive rezultate u kratkom vremenu uz jednu od dvije modifikacije, IG ili VIR.

VNS

Metodi VND najsličniji postupak je VNS. Daje bolje rezultate, s medijanom 10.14. Iako je ideja slična, korištenje više načina generiranja susjedstva, eksperimenti pokazuju da je VNS bolji. Prednost VNS-a je što nedeterminističkim redoslijedom koristi tri susjedstva. VND u svakoj iteraciji istim redoslijedom poziva dva postupka lokalne pretrage dok ne dođe do optimuma s obzirom na oba susjedstva. VNS koristi nasumično biranje susjeda što može pomoći u izbjegavanju lokalnih optimuma. Rezultati su bolji od pravila ATC i relativno blizu rezultatima genetskog algoritma. U tablici 4.2 ističe se vrijednost u stupcu *best* koja pokazuje da se sumiranjem najboljih rezultata po instanci dobivaju rezultati skoro jednako dobri najboljim rezultatima genetskog algoritma. U usporedbi s ostalim metodama, ako se promatraju medijani, VNS daje bolje rezultate od 9 među preostalih 13 metoda (ili varijacija metoda).

VNS je jednostavna i brza heuristika koja za problem statičkog raspoređivanja na nesrodne strojeve daje vrlo dobre rezultate.

GRASP

GRASP je implementiran u 6 verzija: običan GRASP, običan GRASP uz ATC, GRASP uz PR, GRASP uz PR i ATC, GRASP uz PR i ILS te GRASP uz PR, ILS i ATC. GRASP bez ikakvih dodatnih poboljšanja daje jako loša rješenja. Medijan je 116.57.

Razlog tome je što konstrukcija početnog rješenja ne stvara dobre rasporede. Dio procesa koji pohlepno raspoređuje poslove na parcijalni raspored nije dobro prilagođen za problem kojim se bavi ovaj rad. To se može vidjeti iz optimalnih vrijednosti parametra α dobivenih pretragom prostora parametara. Ta vrijednost je u jednom slučaju jednaka 0.9, a u ostalih 5 varijanti algoritma 1. Parametar α određuje pohlepnost naspram nasumičnosti pri izgradnji rasporeda. Postavljanjem parametra α na vrijednost 1 dobiva se potpuno nasumično izgrađen raspored. Ispada da je pohlepna izgradnja rasporeda lošija od nasumične.

Metoda koja spaja GRASP i PR daje bolje rezultate. Medijan je 27.17, što je u odnosu na 116.57 veliko poboljšanje, ali i dalje dosta loše. Optimizacija koja je dodana u GRASP i GRASP uz PR je korištenje rasporeda kojeg je generirala ATC heuristika u prvoj iteraciji procesa. Svaka iduća iteracija radi originalno zamišljen postupak konstrukcije početnog rješenja. očekivano, uz ATC, rezultati GRASP metode su dobri. Uspiju se dobiti rezultati bolje od samog ATC-a. Vrijednost medijana u tom slučaju je 11.80. Ovo poboljšanje rezultat je lokalne pretrage koja se radi u prvoj iteraciji nad rasporedom kojeg generira ATC. Svaka druga iteracija jednako je loša kao obični GRASP. Isto vrijedi i za GRASP uz PR uz dodatno generiranje početnog rješenja pomoću heuristike ATC. Medijan je bolji, i iznosi 11.27. Na slici 4.1 može se vidjeti utjecaj korištenja ATC-a u GRASP metodama. Dobiveni rezultati su u uskom intervalu jer nasumičnost nema velikog utjecaja na krajnja rješenja u većini pokretanja.

Tek se dodavanjem ILS procedure na GRASP uz PR dobivaju prihvatljiva rješenja. GRASP uz PR i ILS daje vrlo dobre rezultate što se može vidjeti po medijanu koji iznosi 9.93. Korištenje ATC-a za dobivanje početnog rješenja daje još bolje rezultate. Medijan u tom slučaju iznosi 9.66. To što su rezultati bez korištenja ATC-a jako dobri, govori da je heuristika GRASP uz PR i ILS dobro prilagođena za problem kojim se bavi ovaj rad. Promatranjem stupca *best* u tablici 4.2 vidljivo je da su vrijednosti vrlo blizu onima od genetskog algoritma. Na slici 4.1 može se vidjeti da obje varijante GRASP-a uz PR i ILS zajedno sa simuliranim kaljenjem daju najbolje rezultate.

Metoda GRASP u kombinaciji s procedurama PR i ILS se pokazala kao odličan izbor u rješavanju problema statičkog raspoređivanja na nesrodne strojeve, dok metode bez postupka ILS daju prihvatljiva rješenja samo ako se koristi heuristika ATC.

Simulirano kaljenje

Simulirano kaljenje (SA) izvedeno je u dvije varijante - s i bez ATC-a. ATC se, kao i u ostalim metodama, koristi za generiranje početnog rješenja. SA daje odlične rezultate i bez korištenja ATC-a, no uz ATC su ipak malo bolji. Medijan bez ATC-a je 9.71, uz ATC je 9.62. Simulirano kaljenje daje najbolje rezultate od svih metoda po vrijednostima minimuma, medijana i maksimuma, čak nadmašuje i genetski algoritam. Stupac *best* u tablici 4.2 je jedino gdje je genetski algoritam nadmašio ovu metodu. Iz grafa 4.1 vidi se da su rezultati od SA uz ATC jako blizu GRASP-u uz PR, ILS i ATC. Razlika je u raznolikosti rješenja. Polovica rješenja simuliranog kaljenja uz ATC je u većem intervalu vrijednosti u usporedbi s drugom spomenutom metodom. Mogući razlog tome je što simulirano kaljenje koristi više različitih načina generiranja nasumičnog susjeda nad kojim se provodi lokalna pretraga. Na taj način istražuje se veći prostor rješenja.

Simulirano kaljenje je brza i jednostavna metoda koja se kroz eksperimente pokazala kao najbolja metoda za rješavanje konkretnog problema kojim se bavi ovaj rad. U kombinaciji s heuristikom ATC, ne samo da nadmašuje sve ostale korištene metode, već je po nekim kriterijima bolja i od genetskog algoritma.

Tabu pretraživanje

Tabu pretraživanje (TS) u originalnoj implementaciji koristi pravilo ATC za generiranje početnog rješenja, stoga se nije razmatrala varijanta bez ATC-a. Kao što se može vidjeti iz tablice 4.2 i grafa 4.1 , rezultati nisu zadovoljavajući. Pokazuje karakteristike kao i neke druge metode koje ne rade dobro bez ATC-a, konkretno GRASP i GRASP uz PR. Svi rezultati su u malom intervalu vrijednosti, medijan je 13.22, a minimum i maksimum 12.95 i 13.29. Iako je ideja dobra, korištenje čak osam različitih načina generiranja susjeda i biranje najboljeg, eksperimenti su pokazali da pristup nije dobar. Problem bi mogao biti u atributima koje tabu lista pamti. Zbog efikasnosti, u tabu listi se ne pamte cijela rješenja, već atributi. U ovom slučaju, pamte se stroj i pozicija na koje je posao premješten. To možda nije dovoljno informacija za predstavljanje tabu poteza. Kod simuliranog kaljenja dobra karakteristika je što se nakon nasumično odabranog susjeda nad njim generira cijelo susjedstvo i bira najbolji susjed. Ovdje svaka od osam metoda vraća jednog nasumično odabranog susjeda. Jedini kriterij koji ko-

risti pohlepnost je odabir strojeva na kojima će se provoditi izmjene. Moguće je da ovom pretraživanju nedostaje proces koji će više usmjeriti pretragu prema boljim rješenjima. Osim toga, pohlepan odabir strojeva orijentiran je prema tome da se bira onaj stroj koji ima najveći iznos vrijednosti funkcije dobrote. Za kriterij ukupnog težinskog kašnjenja to možda nije prikladno. Ako se neki posao premjesti na drugi stroj, tamo može još više kasniti i iako se smanji zakašnjelost na jednom stroju, ukupna zakašnjelost raste. Tako se postupak može nepotrebno vrtjeti u krug umjesto da se razmotre zamjene poslova na nekim drugim strojevima.

Tabu pretraživanje nije ispunilo očekivanja i pokazalo se kao jedna od lošijih metoda koja ostavlja prostora za daljnja istraživanja i unaprjeđenja.

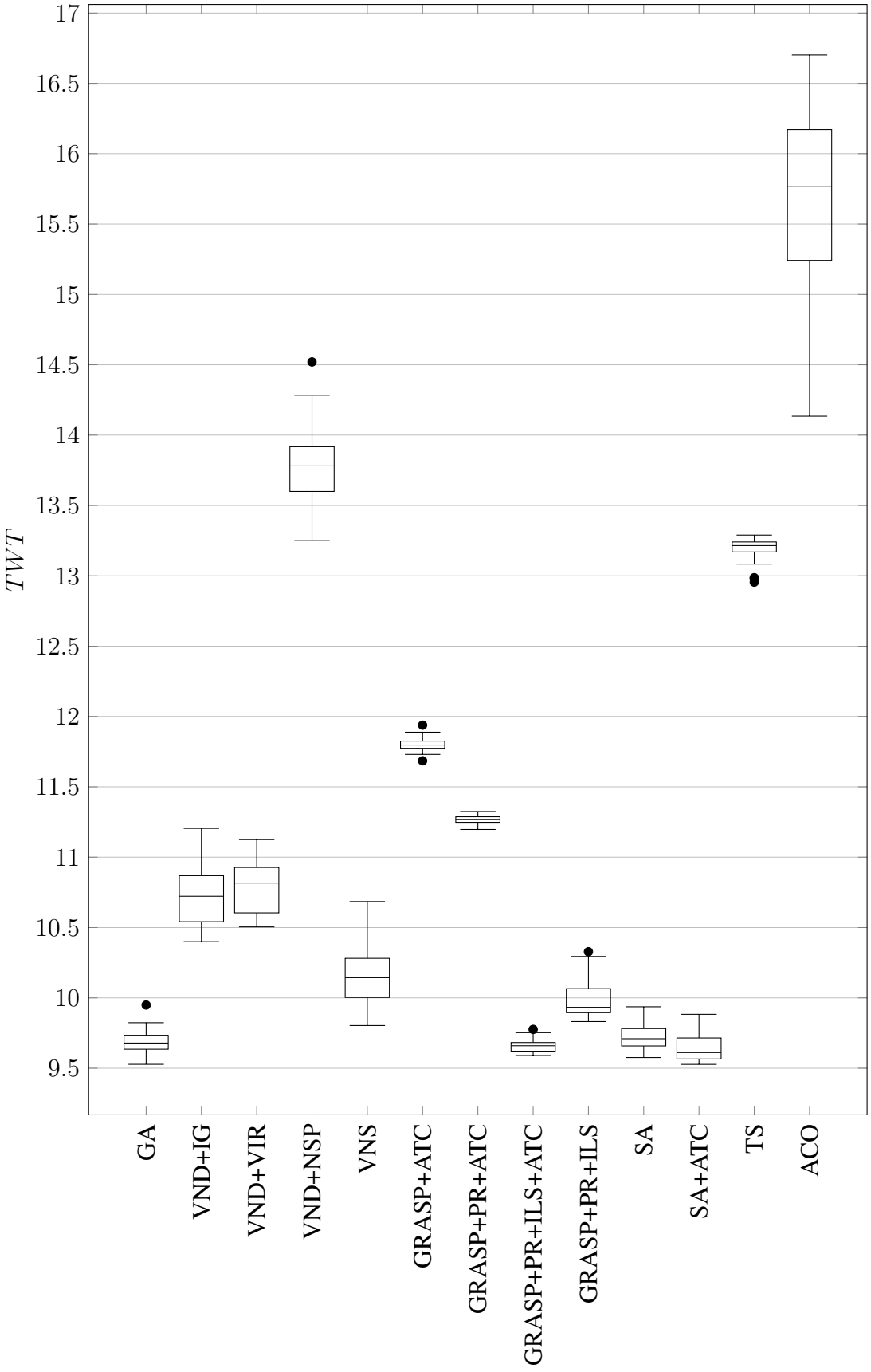
Optimizacija kolonijom mrava

Optimizacija kolonijom mrava (ACO) daje polovicu rezultata u rasponu od 15.3 do 16.8, s medijanom 15.80. U usporedbi s genetskim algoritmom, rezultati su znatno lošiji. Također, u usporedbi s većinom implementiranih metoda, optimizacija kolonijom mrava daje lošije rezultate, lošije čak i od ATC-a, jednostavnog pravila raspoređivanja. Na kvalitetu rezultata znatno utječu vrijednosti parametara s kojima se pokreću eksperimenti. Optimizacija kolonijom mrava ima više parametara od ostalih korištenih metoda. Također, ti parametri su međusobno zavisni. Jedan primjer je količina isparavanja u odnosu na pojačavanje feromona koji daju dobra rješenja. Ili broj mrava i količina isparavanja nakon prolaska svakog mrava. Iz tog razloga bio je proveden postupak pretraživanja po rešetci. No kako je to dugotrajan proces, prostor parametara nije bio detaljno istražen. Stoga, moguće poboljšanje algoritma moglo bi se izvesti iscrpnom pretragom prostora parametara. Drugi razlog za lošija rješenja je vremensko ograničenje izvođenja. Uz zadan veći broj iteracija, bez vremenskog ograničenja, postignuti su bolji rezultati. Konkretno, za 3000 iteracija i iste vrijednosti parametara, dobivena je suma 13.8211. U ovom i sličnim evolucijskim algoritmima, bolja rješenja se najčešće postižu nakon većeg broja iteracija ili povećanjem populacije, u ovom slučaju mrava. Ako se promatra stupac *best* u tablici 4.2, vidi se da je ta vrijednost dosta bolja od vrijednosti minimuma. To ukazuje na stohastičnost ove metode, što se može vidjeti na grafu 4.1 koji pokazuje da je raspon rezultata za metodu ACO puno veći nego kod ostalih metoda.

Optimizacija kolonijom mrava kao i tabu pretraživanje ostavlja prostora za poboljša-

nja, iako je moguće da samo bolji izbor parametara može znatno utjecati i dati bolje rezultate.

Slika 4.1: Prikaz rezultata za sva pokretanja



5. Zaključak

Cilj ovog rada bio je dati pregled i usporedbu raznih metoda rješavanja problema raspoređivanja uključujući iscrpnu pretragu i nekoliko problemu prilagođenih heuristika i metaheuristika. Konkretan problem kojim se ovaj rad bavi je statičko raspoređivanje u okruženju nesrodnih strojeva, a proučavane metode su jednostavne heuristike i metaheuristike poput VNS-a i GRASP-a te simuliranog kaljenja i tabu pretraživanja. Implementirane metode uspoređene su međusobno, ali i s dva postojeća rješenja iz literature - pravilom ATC i genetskim algoritmom. Eksperimenti su provedeni s jednakim vremenskim ograničenjem na instancama problema različitih dimenzionalnosti. Rezultati pokazuju da su učinkovitosti heuristike GRASP uz PR i ILS te metaheuristike simuliranog kaljenja vrlo blizu genetskom algoritmu koji daje najbolja rješenja. Neke metode poput običnog GRASP-a pokazale su se lošijima, čak i u usporedbi ATC-om, jednostavnim pravilom raspoređivanja, dok optimizacija kolonijom mrava i tabu pretraživanje ostavljaju prostora za unaprjeđenja u daljnjem radu. Eksperimenti provedeni u ovom radu služe kao dobra usporedba učinkovitosti algoritama te zajedno s iznesenim zapažanjima i zaključcima daju opširan pregled metoda za rješavanje statičkog raspoređivanja u okruženju nesrodnih strojeva.

LITERATURA

Jean-Paul Arnaout, Ghaith Rabadi, i Rami Musa. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21:693–701, 12 2009.

J. Behnamian, M. Zandieh, i S.M.T. Fatemi Ghomi. Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm. *Expert Systems with Applications*, 36(6):9637 – 9644, 2009.

Marco Dorigo, Mauro Birattari, i Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1:28–39, 12 2006.

J. Du i J.Y. Leung. Minimizing total tardiness on one machine is np-hard. *Mathematics of Operations Research*, 15:483–494, 1990.

Luis Fanjul-Peyro i Rubén Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55 – 69, 2010.

Thomas A. Feo i Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *J. Global Optimization*, 6:109–133, 1995.

Paola Festa i Mauricio G. C. Resende. An annotated bibliography of grasp–part ii: Applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.

Johnson D. Garey, M. *Computers and intractability: A guide to the theory of NP-completeness*. New York: W.H. Freeman, 1997.

F. Glover i G. A. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, New York, 2003.

Fred Glover. Tabu search—part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

Fred Glover. *Tabu search and adaptive memory programing – Advances, applications and challenges*. Kluwer, 1996.

Pierre Hansen i Nenad Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 02 2001.

Scott Kirkpatrick, Charles Daniel Gelatt, i Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220 4598:671–80, 1983.

Jae-Ho Lee, Jae Min Yu, i Dong-Ho Lee. A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: Minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, 69, 12 2013.

Y.H. Lee. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions (Institute of Industrial Engineers)*, 29:45–52, 01 1997.

O. C. Martin Lourenço, H. R. i T. Stützle. *Iterated local search*. Handbook of Metaheuristics. International Series in Operations Research and Management Science 57. Springer New York, 2003.

N. Mladenović i P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097 – 1100, 1997.

P.S. Ow i T.E. Morton. The single machine early/tardy problems. *Management Science*, 35:177–191, 1989.

Mateus Paula, Martin Ravetti, Geraldo Mateus, i Panos Pardalos. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, 18:101–115, 03 2007.

João Paulo de C. M. Nogueira, José Elias C. Arroyo, Harlem Mauricio M. Villadiego, i Luciana B. Gonçalves. Hybrid grasp heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. *Electronic Notes in Theoretical Computer Science*, 302:53–72, 02 2014.

M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, New York, USA, 2005.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, USA, third edition, 2008.

Ribeiro C.C. Resendel M.G. *GRASP with Path-Relinking: Recent Advances and Applications*. In: Ibaraki T., Nonobe K., Yagiura M. (eds) Metaheuristics: Progress as Real Problem Solvers. Operations Research/Computer Science Interfaces Series, vol. 32. Springer, Boston, MA, 2005.

Rubén Ruiz i Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177:2033–2049, 2007.

D. R. Sule. *Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*. CRC Press, Boca Raton, USA, second edition, 2008.

E.-G. Talbi. *Metaheuristics: From design to implementation*. Hoboken, N.J: John Wiley and Sons, 2009.

A. Vepsalainen i T.E. Morton. Priority rules for jobshops with weighted tardiness costs. *Management Science*, 33:1035–1047, 1987.

Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* 2012.

Metode za rješavanje statičkog problema raspoređivanja u okruženju nesrodnih strojeva

Sažetak

Ovim radom obuhvaćen je problem raspoređivanja u okruženju nesrodnih strojeva i metode koje se mogu iskoristiti za rješavanje statičke varijante problema. Problemu je prilagođen jedan postupak iscrpnog pretraživanja za dobivanje optimalnih rješenja. Ostvarene su i isprobane različite varijante problemski specifičnih heurističkih postupaka, kao i nekoliko metaheurističkih postupaka za rješavanje statičkog problema raspoređivanja. Za instance problema različitih veličina provedena je analiza učinkovitosti ostvarenih postupaka međusobnom usporedbom kao i usporedbom s postojećim rješenjima iz literature.

Ključne riječi: statičko raspoređivanje, nesrodni strojevi, iscrpna pretraga, ATC, heuristike, metaheuristike, usporedba učinkovitosti

Methods for solving static scheduling problem in the unrelated machines environment

Abstract

This thesis focuses on scheduling problem in the unrelated machines environment and methods that can be used to solve a static variant of the problem. One method of brute-force search is adapted for the problem with the goal of producing optimal solutions. Also, different problem specific heuristic and metaheuristic approaches are implemented and applied to the static scheduling problem. For instances of different dimensionalities, an analysis based on mutual comparison of methods is conducted, as well as comparison with existing methods from literature.

Keywords: static scheduling, unrelated machines, brute-force search, ATC, heuristics, metaheuristics, efficiency comparison