UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Tatjana Davidović

Raspoređivanje zadataka na višeprocesorske sisteme primenom metaheuristika

Doktorska disertacija

Beograd, februar 2006.

Članovi komisije:

prof. dr Dušan Tošić

dr Nenad Mladenović

prof. dr Đorđe Dugošija

prof. dr Gordana Pavlović–Lažetić

Rezime

U disertaciji se razmatra problem statičkog raspoređivanja zadataka na višeprocesorski sistem proizvoljne strukture. Uzima se u obzir i vreme potrebno za prenos podataka između procesora. Problem raspoređivanja spada u klasu NP-teških problema u većini slučajeva. Samo za nekoliko najjednostavnijih varijanti moguće je pronaći optimalno rešenje u polinomnom vremenu. Matematički model po prvi put je formulisan za ovu varijantu problema raspoređivanja. Predloženi su i zvanični test primeri koji sadrže probleme različitih karakteristika i pogodni su za poređenje metaheurističkih metoda.

Na rešavanje ovog problema, primenjuje se nekoliko poznatih metaheurističkih metoda. Mnoge od njih po prvi put su primenjene na ovu varijantu problema raspoređivanja. Korišćeno je višestartno lokalno pretraživanje, genetski algoritam, tabu pretraživanje i metoda promenljivih okolina. Sve metode implementirane su u odnosu na istu reprezentaciju rešenja. Postignuto je značajno poboljšanje rezultata dobijenih kostruktivnim heurističkim metodama, kao i onih koji predstavljaju njihovu popravku primenom lokalnog pretraživanja u odnosu na neke okoline. Implementirano je i nekoliko strategija za paralelizaciju metode promenljivih okolina.

Ključne reči: Statičko raspoređivanje zadataka, komunikaciono kašnjenje, lokalno pretraživanje, metaheuristike, genetski algoritam, tabu pretraživanje, metoda promenljivih okolina.

Abstract

The dissertation deals with the problem of static scheduling of tasks to the arbitrary structured multiprocessor systems. The time needed to transfer data between processors, i.e. communication delay is taken into account. The scheduling problem is known to be NP-complete in most of the cases. The polynomial time algorithms for finding the optimal solution is known only for several very simple cases. For the first time, mathematical model is developed for this specific variant of the scheduling problem. The benchmarks for the analysis and comparison of heuristic and meta-heuristic methods are proposed.

Several well-known meta-heuristic methods (most of them for the first time) are applied to solve this particular problem variant. These meta-heuristics are multistart local search, genetic algorithms, tabu search and variable neighborhood search. All methods are implemented using the same solution representation. A significant improvement is achieved with respect to constructive heuristic methods as well as to the local optima with respect to different neighborhood structures. In addition, several strategies for the parallelization of variable neighborhood search method are implemented.

Keywords: Static Scheduling, Communication Delay, Local Search, Meta-heuristics, Genetic Algorithms, Tabu Search, Variable Neighborhood Search.

Predgovor

U ovom radu razmatra se jedan značajan problem u računarstvu: raspoređivanje zadataka na procesore koji ulaze u sastav nekog zadatog višeprocesorskog sistema. Za svakog korisnika višeprocesorskog sistema važno je da se njegovi programi izvrše korektno i za što kraće vreme. Svaki takav program sastoji se od modula (zadataka) od kojih će se svaki izvršiti na nekom od procesora. Dodeljivanje zadataka procesorima i određivanje redosleda njihovog izvršavanja naziva se raspoređivanje zadataka.

Cilj ovoga istraživanja je da se razvije efektivna i efikasna metoda za statičko raspoređivanje zadataka na proizvoljne višeprocesorske sisteme uz razmatranje zahtevane komunikacije između procesora. Efektivnost se odnosi na kvalitet dobijene raspodele, dok efikasnost podrazumeva dobijanje odgovarajuće raspodele u što kraćem vremenu.

Za rešavanje problema statičkog raspoređivanja zadataka na procesore nekog višeprocesorskog sistema, u ovoj disertaciji primenjuju se tri poznate metaheurističke metode: metoda promenljivih okolina (Variable Neighborhood Search, VNS), tabu pretraživanje (Tabu Search, TS) i genetski algoritam (Genetic Algorithm, GA). Neke od ovih metoda po prvi put se primenjuju za rešavanje ovog problema, dok su druge već korišćene, ali uglavnom ne u slučaju koji uzima u obzir i komunikaciju među procesorima. Razmatrana je i mogućnost paralelizacije izvršavanja metaheurističkih metoda u cilju povećanja kako brzine izvršavanja tako i kvaliteta dobijene raspodele.

Rad je nastao u Matematičkom Institutu SANU u Beogradu kao nastavak istraživanja vezanih za primene paralelizacije u robotici, koja su bila sadržaj magistarske teze autora. Deo istraživanja, koji se odnosi na paralelizaciju metaheurističkih metoda, obavljen je u Centru za istraživanja u transportu (Centre de Recherche sur les Transports, CRT), pri Fakultetu za matematiku, informatiku i operaciona istraživanja Univerziteta u Montrealu. Veći deo ovde iznetog materijala je u obliku radova saopšten na različitim naučnim skupovima i pripremljen za objavljivanje u časopisima.

Prvi rezultat opisan u disertaciji je formulacija matematičkog modela razmatranog problema. Matematička formulacija nije se mogla pronaći u literaturi u toliko opštem obliku. Po prvi put, razvijen je model koji uključuje vreme komunikacije, tj. vreme koje se troši na razmenu relevantnih međurezultata između procesora, kao i specifičnosti veza među procesorima definisanih arhitekturom višeprocesorskog sistema.

Jezgro disertacije je implementacija tri poznate metaheurističke metode (VNS, TS, GA) bazirana na istoj reprezentaciji rešenja. Neke od ovih metoda već su primenjivane za rešavanje problema raspoređivanja, pa čak i varijante slične onoj koja se u ovom radu razmatra, ali su zasnovane na različitoj reprezentaciji rešenja, tako da ih je nemoguće direktno porediti. Implementacija svih metoda u odnosu na istu reprezentaciju rešenja omogućava efikasno poređenje, kako osnovnih verzija metoda, tako i njihovih poboljšanja i kombinacija (hibrida). Hibridizacija metoda dobro je poznata u literaturi i trenutno je trend u poboljšanju efikasnosti metaheurističkih metoda. Stoga je, kao jedan od rezultata, u okviru ove disertacije po prvi put definisana jedna od mogućih hibridizacija TS i VNS metode.

Poređenjem osnovnih varijanti metoda utvrđeno je da se VNS pokazala najpogodnijom za rešavanje razmatranog problema. Sistematska pretraga u okolini "dobrih" rešenja i postepeno udaljavanje od lokalnih minimuma daje mnogo bolje rezultate od – na slučajnosti zasnovanog – ukrštanja jedinki ma kako "dobre" one bile ili od uspinjanja iz lokalnog minimuma u potrazi za prevojnom tačkom koja bi omogućila spust u neki "bolji" lokalni minimum. Pogodnim poboljšanjima svaka od metoda se može popraviti (što je u okviru ovog rada i učinjeno za VNS), te je stoga realno očekivati da gornji zaključak ne mora da važi u opštem slučaju. Sem toga moguće je da bi se prilikom korišćenja drugačije reprezentacije rešenja metode različito ponašale. Ova otvorena pitanja dobra su podloga za nastavak istraživanja u ovoj oblasti i mogu biti predmet nekih budućih radova.

Obzirom da ne postoje adekvatni zvanični test primeri (benchmarks) koji bi omogućili poređenje heurističkih i metaheurističkih metoda, na slučajan način generisani su grafovi zadataka različitih veličina (broja čvorova) i gustina (broja lukova), kao i oni sa poznatim optimalnim rešenjem. Primeri grafova generisani za potrebe ovog rada, predloženi su za zvanične test primere i nalaze se raspoloživi na Internetu (`http://www.mi.sanu.ac.yu/~tanjad`). Ovako dobijeni skupovi grafova zadataka pokazali su se veoma raznovrsnim, tako da među njima ima kako lakih, tako i teških primera za svaku od implementiranih metaheurističkih metoda.

U okviru ovoga rada, analizirane su postojeće paralelne metaheurističke metode i predloženo više strategija za paralelizaciju metode promenljivih okolina (VNS) koje su omogućile značajno poboljšanje rezultata dobijenih sekvencijalnom implementacijom.

Osim detaljnih eksperimentalnih rezultata primene sekvencijalnih i paralelnih metaheurističkih metoda za rešavanje problema statičkog raspoređivanja zadataka na višeprocesorske sisteme, kao proizvod disertacije ostaju i softverski paketi. Programi za generisanje slučajnih težinskih usmerenih acikličkih grafova mogu se koristiti i za druge namene, tj. za istraživanja u drugim oblastima direktno povezanim sa teorijom grafova. Solidno dokumentovana implementacija metaheurističkih metoda omogućava jednostavne modifikacije, te će stoga predstavljati dobru bazu za buduća poboljšanja i korekcije.

Na ovom mestu želim da se zahvalim svima koji su mi pomogli da posao oko pripreme i izrade doktorske disertacije privedem kraju. Pre svega, mentorima dr Nenadu Mladenoviću, naučnom savetniku Matematičkog instituta SANU i dr Dušanu Tošiću, redovnom profesoru Matematičkog fakulteta u Beogradu na stručnoj pomoći u toku izrade ovog rada, zatim dr Zoranu Markoviću, direktoru Matematičkog instituta SANU, na bezgraničnom strpljenju i razumevanju za specifične potrebe koje su se pojavile u vezi sa izradom ove disertacije, kao i akademiku Dragošu Cvetkoviću, redovnom profesoru Elektrotehničkog fakulteta u Beogradu, na korisnim savetima i sugestijama u pogledu dopune sadržaja rada. Zahvalnost dugujem i prof. Teodoru Gabrielu Crainicu za nesebičnu pomoć i podršku prilikom rada na tezi. Prof. Nelson Maculan, Federal University od Rio de Janeiro, dr Leo Liberti, Politecnico di Milano i Prof. Pierre Hansen, GERAD, Montreal bili su, ne samo koautori, već i dobre vile koje su bdele nada mnom i hrabrile me u teškim trenucima. U dobre vile ubrojila bih još i prof. Nedu Bokan, prof. Veru Kovačević-Vujčić, dr Slobodanku Janković, prof. Katicu (Stevanović) Hedrih, dr Miodraga Kapetanovića, dr Miodraga Mihaljevića, kao i ostale saradnike Matematičkog instituta SANU koji su mi na razne načine pomagali u toku rada na disertaciji. Ogroman doprinos izradi disertacije dala je i moja mala porodica odričući se moga prisustva sa punim razumevanjem. Posebnu zahvalnost zaslužuju i radnici službe obezbeđenja SANU koji su mi omogućili nesmetan rad u zgradi SANU, kao i društvo sa plaže za brigu o mojim dečacima u danima kada ja nisam bila pored njih.

Sadržaj

Pregled slika

Pregled tabela

Spisak korišćenih oznaka i skraćenica

DAG	-	usmereni aciklički graf (Directed Acyclic Graph)
TG	-	graf zadataka (Task Graph)
VPS	-	višeprocesorski sistem
MSPCD	-	problem raspoređivanja na višeprocesorske sisteme uz razmatranje komunikacionog kašnjenja (Multiprocessor Scheduling Problem with Communication Delays)
LP	-	linearno programiranje (Linear Programming)
MILP	-	mešovito celobrojno linearno programiranje (Mixed-Integer Linear Programming)
y_{jk}^{s}	-	binarna promenljiva matematičkog modela, indikator da je j s-ti zadatak na procesoru k
t_j	-	realna promenljiva modela, vreme početka izvršavanja zadatka i
T_{max}	-	funkcija cilja modela, vreme izvršavanja poslednjeg zadatka u grafu
LS	-	lokalno pretraživanje (Local Search)
MLS	-	višestartno lokalno pretraživanje (Multistart Local Search)
VNS	-	metoda promenljivih okolina (Variable Neighborhood Search)
GA	-	genetski algoritam (Genetic Algorithm)
TS	-	tabu pretraživanje (Tabu Search)
n	-	broj zadataka u grafu zadataka
l_i	-	dužina izvršavanja zadatka i
c_{ij}	-	količina podataka za prenošenje između zadataka i i j
p	-	broj procesora u višeprocesorskom sistemu
$D_{p\times p}$	-	matrica rastojanja među procesorima
ccr	-	odnos vremena izračunavanja i brzine komunikacije (communication-to-computation ratio)

N_g	-	broj generacija za izvršavanje GA
N_p	-	veličina populacije za GA
p_{xover}	-	verovatnoća ukrštanja kod GA
p_m	-	verovatnoća mutacije kod GA
$NTABU$	-	maksimalna dužina tabu liste u TS
k_{max}	-	maksimalni broj okolina za VNS
k_{step}	-	korak za uvećanje indeksa okoline u VNS
$plato$	-	verovatnoća prihvatanja novog najboljeg rešenja koje ima istu vrednost funkcije cilja
SL	-	dužina heurističke raspodele
SL_{opt}	-	dužina optimalne raspodele ($= T_{max}$)
t_{max}	-	maksimalno vreme izvršavanja neke metaheurističke metode
n_{iter}	-	maksimalni broj iteracija neke metaheurističke metode
$nimp_{iter}$	-	maksimalni broj iteracija između dve popravke najboljeg rešenja
x	-	rešenje problema raspoređivanja (dopustiva permutacija zadataka)
$\mathcal{N}(x)$	-	okolina rešenja x
Swap	-	okolina definisana premeštanjem zadatka u permutaciji sa jednog mesta na drugo
IntCh	-	okolina definisana zamenom pozicija zadacima iz izabranog para uz očuvanje dopustivosti rezultujuće permutacije
FOR	-	smer pretraživanja unapred (udesno, FORward)
BACK	-	smer pretraživanja unazad (ulevo, BACKward)
FI	-	pretraživanje samo do prvog poboljšanja (First Improvement)
BI	-	detaljno pretraživanje okolina u cilju nalaženja najboljeg rešenja u tim okolinama (Best Improvement)

1. Uvod

Predmet ovoga rada je paralelizacija sekvencijalnih programa, preciznije, raspoređivanje zadataka (poslova, modula) procesorima na izvršavanje. Problem je od značaja ne samo u računarstvu, već i u drugim oblastima (industriji, robotici, medicini, saobraćaju).

Pojava višeprocesorskih sistema pružila je nove mogućnosti za povećavanje brzine izvršavanja programa. Osim toga otvorilo se novo polje u istraživačkoj delatnosti, čiji je cilj efikasno iskorišćavanje paralelnih računara. Paralelizacija postojećih sekvencijalnih programa i generisanje novih, paralelnih, postale su značajne oblasti istraživanja. Ciljevi ove disertacije su efikasno iskorišćenje višeprocesorskih sistema i maksimalno ubrzavanje izvršavanja programa pravilnim dodeljivanjem poslova (zadataka) procesorima unutar paralelnog računara.

Prilikom rešavanja problema raspoređivanja polazi se od programa koji će se izvršavati na višeprocesorskom sistemu, i on se predstavlja kao skup aktivnosti (modula, funkcija, zadataka) koje treba izvršiti. Podrazumeva se da u takvom programu nema petlji i grananja, tj. da su ti problemi već razrešeni razmotavanjem tela petlje i da je izvršen izbor grane koja treba da se izvrši (paralelizacije petlji i grananja su posebna oblast istraživanja i neće biti ovde razmatrani). Najčešće postoje zavisnosti među tim zadacima tako da se oni ne mogu izvršavati proizvoljnim redom. Obzirom da ne moraju svi zadaci biti međusobno zavisni, relacija zavisnosti (prethođenja) između zadataka je relacija parcijalnog poretka, tako da se polazni program može predstaviti u obliku usmerenog acikličnog grafa (Directed Acyclic Graph, DAG) čiji čvorovi su zadaci, a lukovi spajaju međusobno zavisne zadatke; luk vodi od zadatka-prethodnika ka zadatku-sledbeniku. Taj graf se naziva *graf zadataka* (Task Graph, TG). Paralelizacija ovako predstavljenog sekvencijalnog programa naziva se *(statičko) raspoređivanje zadataka* procesorima na izvršenje (Multiprocessor Scheduling Problem), i sastoji se u određivanju na kom procesoru će se svaki od zadataka izvršavati i u kom vremenskom

trenutku će početi njegovo izvršavanje. Naziv *statičko* označava da su sve informacije u vezi programa koji se paralelizuje unapred zadate. Ako to nije slučaj radi se o *dinamičkom raspoređivanju zadataka*, a tu spadaju paralelizacije petlji i grananja i razna raspoređivanja u realnom vremenu.

Značajan faktor prilikom raspoređivanja predstavlja vreme potrebno da se podaci (međurezultati) prenesu sa jednog procesora na drugi. U većini radova koji se sreću u literaturi smatra se da je to vreme malo i da se može zanemariti. Ukoliko se to vreme eksplicitno razmatra prilikom rešavanja problema raspoređivanja (kao što je to slučaj u ovome radu), dobija se varijanta problema raspoređivanja koja se naziva *raspoređivanje zadataka sa komunikacionim kašnjenjem* (Multiprocessor Scheduling Problem with Communication Delays, MSPCD).

U vezi sa tim je i način povezivanja procesora u višeprocesorski sistem na koji se vrši raspoređivanje. U literaturi se uglavnom pretpostavlja da postoji direktna veza između svaka dva procesora, tj. da je sistem *potpuno povezan*. To najčešće nije slučaj, poznate su arhitekture zvezde, hiperkocke, rešetke i razne druge, a u ovom radu se i taj parametar problema uzima u obzir.

Efikasni algoritmi paralelizacije ugrađuju se u prevodioce savremenih računara za generisanje mašinskih kodova koji će što efikasnije iskoristiti sve resurse kojima ti računari raspolažu: veći broj aritmetičko-logičkih jedinica, cash-memorije, zasebni koprocesori za kontrolu ulazno/izlaznih operacija, tj. mogućnost istovremenog izvršavanja računskih i ulazno/izlaznih operacija i sl. [1]. Značaj problema paralelizacije vidi se i u sve široj primeni paralelnih računara kako u univerzitetskim krugovima, tako i u privredi (industriji). Algoritmi koji se primenjuju za paralelizaciju sekvencijalnih programa, mogu se primeniti i na širu klasu problema raspoređivanja (redosled obrade na pojedinim industrijskim mašinama, raspoređivanje nastavnika po odeljenjima, raspoređivanje smenskih radnika – lekari, medicinske sestre, vozači, i sl.)

Teorijski je dokazano [59, 133] da ovaj problem pripada klasi NP-teških problema, čak i u većini specijalnih jednostavnijih slučajeva (raspoređivanje na dva procesora, zanemarivanje komunikacije, nepostojanje zavisnosti među zadacima, itd.), tako da se ne može očekivati da se njegovo optimalno rešenje u opštem slučaju dobije u kratkom vremenu. Tokom proteklih decenija istraživanja u ovoj oblasti išla su u dva osnovna pravca: 1) izdvajanje slučajeva za koje postoji algoritam za nalaženje optimalnog rešenja u polinomnom vremenu i 2) razvoj heurističkih metoda za nalaženje što boljeg suboptimalnog rešenja. Svaki od ovih pristupa detaljnije je opisan u glavi 3. Poslednjih godina pojavljuju se u literaturi radovi u kojima se razvijaju metaheurističke

metode za rešavanje ovog problema. Njihov osnovni cilj je da iterativnim postupkom poprave neko polazno (heurističko) rešenje.

Primena metaheuristika (genetski algoritmi, neuralne mreže, tabu pretraživanje, metoda promenljivih okolina, simulativno zamrzavanje, mravlje kolonije, itd.), prilikom rešavanja optimizacionih problema treba da obezbedi dobijanje kvalitetnijeg suboptimalnog rešenja od onog koje daju klasične, konstruktivne heuristike za razumno (dostižno) vreme izvršavanja. Primenom metaheurističkih metoda najčešće se omogućuje značajna popravka nekog početnog rešenja dobijenog klasičnim heurističkim metodama, uz razuman utrošak vremena za sopstveno izvršavanje.

U ovome radu razmatra se problem statičkog raspoređivanja zadataka na procesore nekog višeprocesorskog sistema pri čemu se uzima u obzir vreme potrebno za razmenu podataka među procesorima i arhitektura veza između procesora. Primenjuju se tri poznate metaheurističke metode: metoda promenljivih okolina (Variable Neighborhood Search, VNS), tabu pretraživanje (Tabu Search, TS) i genetski algoritam (Genetic Algorithm, GA). Detaljniji opis ovih metaheurističkih metoda dat je u glavi 2. Neke od ovih metoda već su korišćene za rešavanje pojedinih varijanti problema raspoređivanja, što je navedeno u pregledu literature. U većini slučajeva vreme potrebno za prenos podataka između procesora je zanemarivano. Sem toga, uvođene su različite pretpostavke o strukturi višeprocesorskog sistema na koji se vrši raspoređivanje i korišćene su razne reprezentacije rešenja polaznog problema. U cilju lakše analize i upoređivanja, u okviru rada na ovoj disertaciji izvršena, je implementacija svih metoda u odnosu na istu reprezentaciju rešenja. Takođe, struktura višeprocesorskog sistema nije fiksna, već se zadaje kao ulazni parametar.

U daljem tekstu detaljnije su opisani problemi raspoređivanja i data je njihova klasifikacija. U Glavi 2. obrazložen je metaheuristički prilaz rešavanju problema kombinatorne optimizacije, detaljno su objašnjene metaheurističke metode koje se u ovom radu primenjuju. U sledećoj glavi data je formalna postavka problema koji je predmet ovog rada i pregled specijalnih slučajeva koji se u literaturi razmatraju. Detaljno su opisani pristupi koji se koriste za rešavanje problema paralelizacije, dat je pregled odgovarajuće literature sa posebnim osvrtom na originalne rezultate autora. Glava 4. sadrži pregled radova u kojima se koristi metaheuristički pristup za rešavanje problema raspoređivanja, zatim opis konkretne implementacije metaheurističkih metoda za primenu na problem paralelizacije, kao i rezultate raspoređivanja slučajno generisanih grafova zadataka metaheurističkim metodama, njihovo poređenje

i poboljšanja. Strategije paralelizacije metaheurističke metode promenljivih okolina i rezultati primene paralelnih verzija i poređenja međusobno i sa sekvencijalnim varijantama opisani su u glavi 5. U glavi 6. sumirani su rezultati prikazani u ovom radu i date smernice za nastavak istraživanja u ovoj naučnoj oblasti. Kao prvi dodatak opisan je generator grafova zadataka sa zadatim karakteristikama koji je korišćen za dobijanje primera za testiranje i poređenje razvijenih heuristika. Drugi dodatak sadrži pomoćne tabele sa rezultatima paralelnih izvršavanja metode promenljivih okolina.

1.1. Osnovni zadaci paralelizacije sekvencijalnih programa

Kao što je već napomenuto, problem pisanja programa koji bi se izvršavali na paralelnim računarima aktuelan je već nekoliko decenija. On se obično rešava na dva osnovna načina [118]: paralelizacijom postojećih sekvencijalnih programa ili pisanjem novih, u osnovi paralelnih.

Mogućnost da se dve ili više stvari rade istovremeno, je potpuno prirodna i prisutna u svakodnevnom životu (razni uređaji, kao na primer satovi, parne mašine se sastoje iz većeg broja komponenti koje sve rade zajedno i istovremeno; pa i sam ljudski organizam: svi imamo dve ruke koje obično u istom trenutku prave različite pokrete, istovremeno možemo i da pišemo i da gledamo u tekst, pa čak i da pri tom slušamo neku muziku) [19]. Međutim, za većinu problema koji se u svakodnevnom životu javljaju već postoje (manje ili više efikasni) sekvencijalni algoritmi za njihovo rešavanje. Imajući to na umu, postaje jasno zašto se većina istraživača (pa i autor ovog rada) opredeljuje za pristup zasnovan na paralelizaciji postojećih sekvencijalnih algoritama, a samo manji broj njih radi na generisanju paralelnih načina za rešavanje konkretnih problema.

Prvi radovi iz oblasti paralelizacije stariji su od 40 godina [13, 64, 79, 94, 117, 138]. Neki od njih odnose se na industrijske procese (proizvodne linije), dok drugi zaista razmatraju upotrebu paralelnih računara.

U okviru rada na paralelizaciji sekvencijalnih programa potrebno je razrešiti nekoliko osnovnih problema (zadataka):

- granulacija, tj. podela polaznog programa na module koji predstavljaju zadatke za raspoređivanje;

- izbor metode za rešavanje problema raspoređivanja; optimalni-suboptimalni algoritmi, heurističke-metaheurističke metode;
- analiza grafa zadataka, određivanje optimalnog broja procesora i najboljeg algoritma za taj graf;
- redundantna izračunavanja (gde i kada ih uvoditi) i prekidi izvršavanja pojedinih zadataka (kako bi se izvršili neki drugi, trenutno značajniji).

U literaturi se navedeni problemi razmatraju uglavnom odvojeno. Problem granulacije (Partitioning problem) sastoji se u razmatranju polaznog programa (koji se paralelizuje) i definisanju zadataka iz kojih se on sastoji [9, 29]. Ti zadaci će se zatim pojaviti kao čvorovi grafa zadataka, tj. kao objekti raspoređivanja. Umesto serijske verzije programa, može se krenuti i od neke granulacije koja bi se popravila (deljenjem ili spajanjem zadataka), tako da se zatim u raspoređivanju dobije "bolji" rezultat. Kao izlaz tog razmatranja pojavljuje se graf zadataka koji je ulaz u algoritam raspoređivanja. Problem granulacije se veoma malo razmatra, do sada je jedino zaključeno da su fino granulisani zadaci pogodniji za izvršavanje na sistemima sa deljenom memorijom gde je brzina komunikacije velika i potrebno je obezbediti ravnomerno opterećenje procesora. Krupnija granulacija pogodnija je u slučaju kada se komunikacija ne može zanemariti, jer ona obezbeđuje dovoljno izračunavanja između slanja podataka. Pri tome se, naravno, gubi na ravnomernosti opterećenja jer je takve zadatke teže lepo upakovati [1, 29, 96, 139]. Problem granulacije se ne razmatra u ovom radu, ali se uzima u obzir kroz različite primere na kojima se testiraju razvijeni algoritmi raspoređivanja. U tom cilju napravljen je automatski generator grafova sa zadatim karakteristikama (gustina veza između zadataka, uniformna dužina izvršavanja i komunikacije, ograničenja na veze između zadataka, postojanje i poznavanje optimalne raspodele za zadatu arhitekturu, obezbeđivanje zadate širine grafa, postojanje nepovezanih podgrafova,...). Detaljan opis generatora dat je u prilogu A. Obzirom na nedostatak adekvatnih test primera u međunarodnim krugovima, grafovi zadataka generisani za potrebe istraživanja opisanih u ovom radu predloženi su za zvanične testove efikasnosti (benchmarks) heurističkih i metaheurističkih metoda raspoređivanja [39].

Najviše radova postoji na temu algoritama raspoređivanja i to uglavnom na uvođenju raznih heuristika za dobijanje zadovoljavajućeg suboptimalnog rešenja. Polazi se od pretpostavke da je zadat graf zadataka (ne

ulazeći u to da li je on najpogodniji ili postoji bolji) i vrši se njegovo raspoređivanje na zadatu višeprocesorsku arhitekturu. Problem raspoređivanja zadataka je, kao što je već spomenuto, NP-težak pa se u opštem slučaju optimalno rešenje ne može dobiti algoritmima polinomne složenosti. Stoga se razlikuju dve struje istraživača: oni koji traže klase grafova za koje se optimalna raspodela može dobiti u polinomnom vremenu [57, 85, 135] i oni koji razvijaju heurističke metode za što šire klase grafova kao na primer u radovima [35, 51, 93, 95, 125, 123]. Nedavno su se pojavili i metaheuristički pristupi zasnovani na opštim algoritmima kombinatorne optimizacije koji podrazumevaju iterativno raspoređivanje u cilju popravljanja nekog postojećeg suboptimalnog rešenja. Razvoj algoritama raspoređivanja baziranih na metaheurističkim metodama je osnovni predmet ovog rada.

U literaturi postoji malo radova i na temu analize grafa zadataka i izbora višeprocesorske arhitekture na osnovu toga. To se može opravdati činjenicom da se raspoređivanje vrši na raspoloživu višeprocesorsku arhitekturu koja je obično fiksne strukture. Naravno, to ne mora uvek da važi (na primer, neki posojeći višeprocesorski sistemi su rekonfigurabilni), a i ne moraju se iskoristiti uvek svi procesori. Optimizacija broja procesora je naročito važna pri raspoređivanju na klastere radnih stanica, gde više korisnika prijavljuje svoja izvršavanja. U tom slučaju, ukoliko je problem moguće rešiti za isto vreme uz angažovanje manjeg broja procesora, ostali se mogu upotrebiti za neka druga izvršavanja. Dakle, zadatak određivanja "optimalnog" broja procesora i te kako ima smisla. Analiza grafa zadataka u cilju određivanja najpogodnije heuristike za raspoređivanje takođe ima smisla imajući u vidu da različite heuristike ne daju dobre rezultate za sve strukture grafova zadataka i arhitekture na koje ih treba rasporediti. Realizacija izbora najpogodnije heuristike u datom slučaju omogućila bi klasifikaciju heurističkih metoda raspoređivanja koja još nije izvršena (postoje radovi koji se time bave [34, 67], ali je problem i dalje otvoren).

Uvođenje redundantnih izračunavanja (task duplication) zastupljeno je u literaturi jer može bitno da skrati vreme izvršavanja paralelnog programa. Osnovna ideja je da se izvršavanjem jednog zadatka na većem broju procesora izbegne slanje njegovih rezultata kroz višeprocesorski sistem jer se tako može značajno skratiti paralelno izvršavanje. Međutim, to je veoma složen problem i nema mnogo pravila kako se to radi. U ovom radu takve mogućnosti se ne razmatraju, kao ni eventualna prekidanja zadataka (task preemption) u toku njihovog izvršavanja, a više detalja o rešavanju ovih problema može se naći u [10, 22, 141]. Ovi problemi dobijaju na značaju u dinamičkom raspoređi-

vanju, tj. raspoređivanju u realnom vremenu, o čemu će biti malo više reči u sledećem odeljku.

1.2. Problem raspoređivanja zadataka na višeprocesorske sisteme

Kao što je već napomenuto, problem raspoređivanja zadataka na višeprocesorske sisteme sastoji se u sledećem: dat je program koji treba paralelizovati, izdeljen na module (zadatke); treba odrediti koji će se zadatak na kom procesoru izvršavati i kada će početi njegovo izvršavanje, a cilj je da se minimizira ukupno vreme izvršavanja celog programa.

Problem raspoređivanja zadataka na višeprocesorske sisteme može se, na najvišem nivou, podeliti na statički i dinamički. Kod statičkog raspoređivanja su svi podaci potrebni za raspodelu unapred poznati, tj. polazni program je fiksan u smislu da su svi moduli unapred zadati, da se "tačno" zna dužina njihovog izvršavanja, relacija zavisnosti među njima i količina zahtevane komunikacije i tokom raspodele se ništa ne može promeniti. Osnovna odlika dinamičkog raspoređivanja je da se tokom same raspodele otkrivaju neki novi podaci koji bitno utiču na raspodelu. Dakle, dinamičko raspoređivanje imamo u slučaju da graf zadataka nije zadat unapred, nije fiksan već se menja u toku same raspodele (petlje koje generišu onoliko zadataka kolika je vrednost brojača koja se recimo učitava na početku izvršavanja, zatim grananja koja aktiviraju samo neke zadatke, dok se oni sa "lažne" grane uopšte ne izvršavaju). To su takozvana raspoređivanja u realnom vremenu, u trenutku kada je izvršavanje polaznog programa već počelo i karakteristična su za dinamičke industrijske procese (fleksibilne robotizovane proizvodne ćelije, upravljanje letilicama i navođenim vozilima, nuklearne elektrane, meteorološke aplikacije, i sl.). Za dinamičko raspoređivanje karakteristično je i vremensko ograničenje za samu raspodelu, jer ako se zadatak ne rasporedi i ne završi u predviđenom roku, onemogućeno je izvršavanje celog programa, što može imati katastrofalne posledice u pojedinim primenama (rušenje letilice, nuklearnu katastrofu).

Specifičnost dinamičkog raspoređivanja je da u opštem slučaju rešenje problema raspoređivanja ne mora da postoji, a u nekim slučajevima, čak i kada postoji, nije sigurno da ćemo ga dobiti, baš zbog toga što nam nisu poznate sve informacije. Drugim rečima to samo znači da na početku ras-

poređivanja nije izvesno da li će raspodela uspeti ili ne. Dinamičko raspoređivanje je stoga izuzetno zanimljiv i izazovan problem kojim se bave mnogi istraživači, a ovde su izdvojeni neki od radova u kojima se ta problematika razmatra [6, 21, 50, 100, 101, 110, 119, 120, 126].

Predmet ovoga rada ipak ostaje u domenu statičkog raspoređivanja koje je već dovoljno složeno za rešavanje, a postoji sve veća potreba u svim granama ljudske delatnosti za nalaženjem što efikasnijih rešenja. Statička paralelizacija može se pojaviti kao značajan deo raspoređivanja u dinamičkim slučajevima, te je stoga izuzetno važno da se obezbedi brzo i efikasno dobijanje što boljeg rešenja tog dela, kako bi se obezbedilo što više vremena i efikasnih međurezultata potrebnih za rešavanje celog problema. Na primer, kod heuristike dvostrukog horizonta, primenjene na rešavanje problema navođenja vozila u realnom vremenu [101] u jednoj fazi se rešava problem raspoređivanja vozila pod pretpostavkom da je trenutno stanje konačno, dakle rešava se problem statičkog raspoređivanja.

Zadatak statičke paralelizacije formulisan je kao raspoređivanje grafa zadataka za izvršavanje na višeprocesorskom sistemu proizvoljne arhitekture. Opisano je njegovo optimalno rešenje u smislu najkraćeg vremena potrebnog za izvršavanje svih zadataka iz grafa, uključujući i vreme koje se troši na prenos podataka između zadataka koji se izvršavaju na različitim procesorima. Osnovni doprinosi rada su matematički model jedne varijante problema statičkog raspoređivanja baziran na linearnom programiranju; detaljna klasifikacija problema raspoređivanja u odnosu, kako na strukturu modula koji se raspoređuju i njihov međusobni odnos, tako i na arhitekturu višeprocesorskog sistema na koji se vrši raspoređivanje; razvoj i analiza heurističkih i metaheurističkih algoritama za nalaženje što boljeg suboptimalnog rešenja. Na kraju se predlaže paralelizacija metaheurističkih metoda sa ciljem povećavanja ne samo brzine njihovog izvršavanja, nego i kvaliteta dobijenog rešenja.

2. Pregled nekih metaheuristika

U ovoj glavi opisane su metaheurističke metode koje su u ovom radu korišćene za rešavanje problema statičkog raspoređivanja zadataka na višeprocesorske sisteme. Posebna pažnja posvećena je metodi promenljivih okolina koja je jedna od novijih i do sada je na našem jeziku opisana samo u disertaciji dr Dragana Urošević [134]. Osim toga, glavni rezultati rada vezani su za analizu i modifikacije ove metode kao i za definiciju hibrida koji se dobija kombinovanjem metode promenljivih okolina i tabu pretraživanja. Najpre je data opšta formulacija problema optimizacije i definicije odgovarajućih pojmova, istaknute su različite metode rešavanja optimizacionih problema i naglašen značaj primene metaheuristika. Nakon opisa metaheuristika korišćenih u ovoj disertaciji, na kraju ove glave, opisane su ukratko neke od metoda dobijene kombinovanjem osnovnih verzija metaheuristika.

2.1. Postavka problema kombinatorne optimizacije

U opštem slučaju zadatak optimizacije se definiše na sledeći način [27, 32, 131]:

Definicija 1. *Neka je $f : S \to R$ realna funkcija definisana na skupu S i neka je $X \subseteq S$ neki zadati skup. Problem je naći*

$$\min f(x)$$

pod uslovom (ograničenjem)

$$x \in X.$$

Problem maksimizacije $f(x)$ se može svesti na minimizaciju $-f(x)$, tako da se neće posebno razmatrati.

Ako je X konačan ili prebrojiv skup, problem optimizacije je *problem kombinatorne* ili *diskretne optimizacije*, dok je u suprotnom reč o *kontinualnoj optimizaciji*. Elementi skupa S predstavljaju (potencijalna) *rešenja* problema, dok se u skupu X nalaze *dopustiva rešenja* optimizacionog problema. Skupovi S i X nazivaju se, redom, *prostor rešenja* i *prostor dopustivih rešenja*.

Funkcija $f(x)$ predstavlja *funkciju cilja*, a zadatak je naći ono rešenje x za koje $f(x)$ ima najmanju moguću vrednost. Takvo $x^* \in X$ za koje važi $f(x^*) \leq f(x)$ za svako $x \in X$ naziva se *optimalno rešenje* ili *globalni minimum*. Sva ostala (dopustiva) rešenja nazivaju se još i suboptimalnim.

Optimalno rešenje može biti jedinstveno, ali može postojati i više međusobno različitih elemenata skupa X koji odgovaraju istoj (minimalnoj, optimalnoj) vrednosti funkcije cilja. Rešenje problema optimizacije sastoji se najčešće u pronalaženju jednog optimalnog rešenja, ređe se zahteva pronalaženje svih optimalnih rešenja.

Najopštije posmatrano, metode optimizacije se mogu podeliti na *egzaktne* i *približne* (koje se dalje dele na *heurističke* i *simulacione*). Egzaktne metode su najpoželjnije jer garantuju optimalnost dobijenog rešenja, ali je njihova primenljivost u praktičnim problemima najmanja, dok je rezultat dobijen primenom simulacije najnepouzdaniji, ali je broj zadataka koji se mogu rešavati ovom grupom metoda najveći.

Osnovni problem kod rešavanja zadatka kombinatorne optimizacije je što je broj (dopustivih) rešenja, iako konačan, izuzetno veliki. Stoga je za probleme većih dimenzija nemoguće primenjivati egzaktne metode rešavanja. U cilju rešavanja takvih problema, došlo je do razvoja heurističkih metoda. Heuristike predstavljaju konačan skup koraka kojima se dobijaju rešenja problema kombinatorne optimizacije (bez garancije njihove optimalnosti) za relativno kratko vreme. Osnovna prednost heurističkih metoda je njihova brzina, što omogućava dobijanje zadovoljavajućih rešenja za probleme velikih dimenzija kakvi se najčešće javljaju u realnim primenama. Nedostatak je što za dobijeno rešenje, ne samo da ne postoji garancija optimalnosti, već najčešće ni procena kvaliteta dobijenog rešenja. Međutim, za probleme velikih dimenzija osnovni cilj je da se neko rešenje dobije, bez obzira na njegov kvalitet. Kada rešenje već postoji, mogu se primeniti razne tehnike za njegovo poboljšanje, što je u suštini osnovna ideja prilikom nastajanja metaheuristika (o kojima će više reči biti u nastavku ove glave).

Heurističke metode mogu se podeliti u nekoliko grupa [131].

(i) *konstruktivne heuristike*;

(ii) *heuristike iterativnog poboljšavanja*;
(iii) *heuristike matematičkog programiranja*;
(iv) *dekompozicione heuristike*;
(v) *heuristike bazirane na podeli dopustivog skupa*;
(vi) *heuristike bazirane na restrikciji dopustivog skupa*;
(vii) *heuristike bazirane na relaksaciji*.

(i) *Konstruktivne metode*. Ova grupa metoda postepeno gradi (konstruiše) rešenje maksimalno koristeći specifična znanja svakog pojedinačnog zadatka.

(ii) *Iterativno poboljšavanje*. Savremeni naziv za ovu grupu je *metode lokalnog pretraživanja* i biće im više pažnje posvećeno kasnije u tekstu. Polazi se od proizvoljnog početnog rešenja, koje se postepeno poboljšava pretraživanjem njemu "bliskih" rešenja. Proces se ponavlja sve dok u "blizini" tekućeg rešenja postoji bolje rešenje od tekućeg. Naravno, problem odredjivanja bliskih rešenja je ključan, o čemu će kasnije biti reči.

(iii) *Matematičko programiranje*. Problem se formuliše kao zadatak matematičkog programiranja, pa se onda rešava približnim (a ne egzaktnim) metodama.

(iv) *Dekompozicione heuristike*. Početni problem se razbije (dekomponuje) na više manjih podproblema, čijim se čak ni egzaktnim rešavanjem ne mora garantovati optimalnost rešenja za polazni problem.

(v) *Podela dopustivog skupa*. Dopustivi skup se podeli na više podskupova, pa se delimičnim pretraživanjem rešenja u svakom od njih nadje najbolje. Heurističko rešenje je ono kod kojega je funkcija cilja najbolja.

(vi) *Restrikcija dopustivog skupa*. Restrikcijom se jednostavno eliminišu odredjeni podskupovi dopustivog skupa i tako smanjuje prostor pretrage. To omogućava da se novi zadatak lakše rešava.

(vi) *Relaksacija*. Suprotan pristup od restrikcije je relaksacija, kojom se dopustiv skup proširuje, ali tako da se omogućuje jednostavnije rešavanje novog problema.

Naravno, u rešavanju nekog konkretnog problema, moguće je kombinovati različite tipove heurističkih pristupa. Na primer, primenom konstruktivne heuristike pronađe se neko rešenje optimizacionog problema koje se zatim koristi kao polazno rešenje za iterativno poboljšavanje.

Kako su se heuristike pokazale kao praktično jedini način rešavanja optimizacionih zadataka, istraživačka populacija u poslednje vreme sve više pažnje posvećuje njihovom razvoju i usavršavanju. To je dovelo i do razvoja

univerzalnih heuristika ili tzv. metaheuristika. Klasične heuristike su, uglavnom, bile namenjene rešavanju nekih konkretnih, pojedinačnih problema i koristile su poznate osobine datog problema pri njegovom rešavanju. Metaheuristike, naprotiv, sastoje se od uopštenih skupova pravila koja se mogu primeniti za rešavanje raznovrsnih problema optimizacije. Metaheuristički pristupi u rešavanju optimizacionih problema zasnovani su na opštim algoritmima optimizacije koji podrazumevaju primenu iterativnih postupaka u cilju popravljanja nekog postojećeg rešenja.

Implementacija većine metaheurističkih metoda zasnovana je na pojmu dopustivog rešenja i definiciji njegove okoline (u cilju primene iterativnog poboljšavanja). Da bi se definisala okolina nekog rešenja mora se uvesti pojam rastojanja između dveju tačaka [8].

Definicija 2. *Neka je X proizvoljan skup tačaka. Ako se svakom uređenom paru (x, y) tačaka iz X pridruži realan broj $d(x, y)$ koji ima osobine:*

$$1^\circ \ \ 0 \leq d(x, y) < +\infty,$$
$$2^\circ \ \ d(x, y) = 0 \Leftrightarrow x = y,$$
$$3^\circ \ \ d(x, y) = d(y, x),$$
$$4^\circ \ \ d(x, y) \leq d(x, z) + d(z, y),$$

kaže se da je skup X snabdeven metrikom d. *Takav skup X, tj. uređeni par (X, d), naziva se* metričkim prostorom. *Vrednost $d(x, y)$ predstavlja* rastojanje *između tačaka x i y.*

Definicija 3. *Skup $\mathcal{N}_d(x) \subseteq X$ svih tačaka na rastojanju d od neke izabrane tačke $x \in X$ naziva se* d-okolina *tačke x. Elementi skupa $\mathcal{N}_d(x)$ nazivaju se* d-susedi *tačke x.*

Opštija definicija može obuhvatiti sve tačke na rastojanju manjem ili jednakom d. Ukoliko je vrednost d fiksirana, govori se o *okolini* tačke x koja sadrži sve njene *susede*.

U optimizacionim problemima uobičajeno se koristi manje formalna definicija okoline i suseda nekog rešenja x [112]. Pod okolinom rešenja x se podrazumeva $\mathcal{N}(x) \subseteq X$ koji je pridružen rešenju x po nekom pravilu. To je skup rešenja dobijenih od x primenom neke *elementarne transformacije*. Definicija elementarne transformacije zavisi od optimizacione metode koja se koristi kao i od njene konkretne implementacije. Obično se podrazumeva da rešenje x ne pripada svojoj okolini, tj. $x \notin \mathcal{N}(x)$. U ovom slučaju se takođe mogu posmatrati d-okoline nekog rešenja pri čemu je d prirodan broj (a ne realan

kao u definicijama 2 i 3). $\mathcal{N}_d(x)$ je sada skup svih rešenja koja se dobijaju primenom d puta (iste) elementarne transformacije na rešenje x.

Definicija 4. *Rešenje $x \in X$ naziva se* lokalni minimum *nekog optimizacionog problema ukoliko ne postoji $x' \in \mathcal{N}(x) \subseteq X$ takvo da važi $f(x') < f(x)$.*

Definicija 5. *Rešenje $x \in X$ naziva se* globalni minimum *nekog optimizacionog problema ukoliko ne postoji $x' \in X$ takvo da važi $f(x') < f(x)$.*

Lokalni minimumi se nazivaju *suboptimalna rešenja*. Obzirom na složenost optimizacionih zadataka, sve heurističke i metaheurističke metode usmerene su ka nalaženju što boljih suboptimalnih rešenja u što kraćem vremenu. Metoda iterativnog poboljšavanja kod koje je pretraživanje ograničeno na unapred definisanu okolinu rešenja u potrazi za lokalnim minimumom naziva se lokalno pretraživanje.

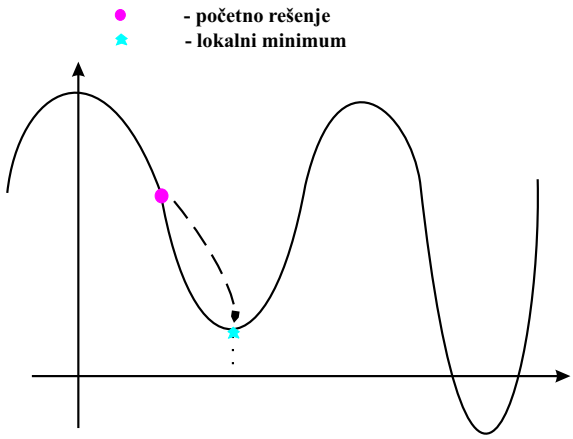
Lokalno pretraživanje (Local search, LS) podrazumeva da se za svako $x' \in \mathcal{N}(x)$ u okolini nekog početnog rešenja x, izračunava vrednost $f(x')$ i ukoliko je $f(x') < f_{min}$, čuva se novo $x_{min} = x'$ i $f_{min} = f(x')$. Kada se "obiđu" sva rešenja u okolini $\mathcal{N}(x)$, nastavlja se pretraga u okolini $\mathcal{N}(x_{min})$. Ispitivanje svih suseda nekog rešenja x_{min} naziva se pretraživanje okoline (neighborhood exploration, NE). NE predstavlja jednu iteraciju lokalnog pretraživanja. Proces lokalnog pretraživanja odvija se u iteracijama i zaustavlja se kada u okolini $\mathcal{N}(x_{min})$ ne postoji bolje rešenje.

Ilustracija izvršavanja lokalnog pretraživanja data je na slici 2.1. Pseudokod za LS može se prikazati u sledećem obliku.

1. *Inicijalizacija.* Izabrati početno rešenje x (slučajno ili primenom neke konstruktivne heuristike). Postaviti $x_{min} = x$ i $f_{min} = f(x)$.

2. **Ponavljaj**
   POPRAVKA = 0;
   $\forall x' \in \mathcal{N}(x_{min})$
      ako $(f(x') < f_{min})$ **onda**
         $x_{min} = x'$;
         $f_{min} = f(x')$;
         POPRAVKA = 1;
      kraj-ako
dok nije POPRAVKA == 0;

Ponekad je isuviše sporo, pa čak i nemoguće pretraživati celu okolinu



Sl. 2.1: Grafički prikaz metode lokalnog pretraživanja

$\mathcal{N}(x_{min})$. Stoga postoje razni načini (heuristike) kako da se odrede susedi koji "obećavaju" popravljanje vrednosti funkcije cilja, tj. da se na neki način smanji (redukuje) veličina okoline koja se pretražuje.

Osnovni nedostatak lokalnog pretraživanja je što se zaustavlja pri nailasku prvog lokalnog minimuma koji ne mora biti (i najčešće nije) globalni minimum, a to uveliko zavisi od početnog rešenja.

Sa druge strane, razvile su se mnoge metaheurističke metode optimizacije, najčešće po uzoru na neke poznate procese, prvenstveno u biologiji (genetski algoritmi (GA) i razne specijalne varijante evolutivnih algoritama (EA), neuralne mreže (NN), mravlje kolonije (AC)), u fizici (simulativno zamrzavanje (SA)), ali i metode inspirisane lokalnim pretraživanjem sa raznim idejama da se izbegne zamka lokalnog minimuma (višestartno lokalno pretraživanje (MLS), metoda promenljivih okolina (VNS), tabu pretraživanje (TS), Greedy Randomized Adaptive Search Procedure (GRASP)) i druge.

One su postale značajno, a najčešće i jedino, sredstvo u rešavanju realnih problema baziranih na optimizaciji. Osnovni zahtev je dobijanje rešenja bliskih optimalnom u razumnom vremenu. Osim toga, mogu se primeniti za ubrzavanje tačnih metoda tako što će se koristiti za dobijanje dobrog početnog rešenja. Skorašnje primene potvrđuju da su se pokazale veoma uspešnim i kao sastavni delovi sistema za otkrivanje znanja u okviru veštačke inteligencije [18, 31]. U svetlu ovih primena mogu se definisati osobine koje

efikasne metaheuristike treba da poseduju kako bi obezbedile značaj i na praktičnom i na teorijskom planu [74]:

- *jednostavnost*: treba da budu zasnovane na jednostavnim i lako razumljivim pravilima;
- *preciznost*: koraci kojima se opisuje metaheuristička metoda treba da su formulisani preciznim, po mogućnosti matematičkim, terminima;
- *doslednost*: svi koraci metode treba da budu u skladu sa pravilima kojima je metaheuristika definisana;
- *efikasnost*: primena metaheuristike na neki konkretan problem treba da obezbedi dobijanje rešenja bliskih optimalnom za većinu realnih primera, naročito za zvanične test primere (benchmarks) raspoložive u toj klasi;
- *efektivnost*: za svaki konkretan problem, metoda mora da obezbedi optimalno ili rešenje blisko optimalnom u razumnom vremenu izvršavanja;
- *robusnost*: metoda treba da daje podjednako dobre rezultate za širok spektar primera iz iste klase, a ne samo za neke odabrane test primere;
- *jasnoća*: treba da bude jasno opisana kako bi se lako razumela i, što je još važnije, lako implementirala i koristila;
- *univerzalnost*: principi kojima je metoda definisana treba da budu opšteg karaktera kako bi se sa lakoćom primenjivala na nove probleme.

Monte-Carlo metoda. Kao najjednostavnija metaheuristička metoda pominje se metoda Monte-Carlo [55]. Suština ove metode je da se iz prostora dopustivih rešenja X bira slučajna tačka koja predstavlja novo rešenje. Ukoliko je to rešenje bolje od trenutno najboljeg rešenja, usvaja se kao novo najbolje, u protivnom se odbacuje. Ovaj postupak se ponavlja sve dok se ne zadovolji neki, unapred zadati, izlazni kriterijum. Pseudokod metode Monte-Carlo ima sledeći oblik:

1. *Inicijalizacija.* Izabrati početno rešenje x, $x_{opt}=x$, $f_{opt}=f(x)$.

2. **Ponavljaj**
   (a) *Pokušaj.* Izabrati slučajno rešenje x' u prostoru dopustivih rešenja X;
   (b) *Provera rešenja.* **ako** $f(x') < f(x_{opt})$ **onda** $x_{opt} = x'$, $f(x_{opt}) = f(x')$

dok nije zadovoljen kriterijum zaustavljanja.

Uobičajeni izlazni kriterijumi su broj pokušaja (broj slučajnih izbora novog rešenja, ili broj iteracija, kako se najčešće naziva) i broj pokušaja između dve popravke tekućeg najboljeg rešenja.

Metoda Monte-Carlo nastala je kao suprotnost detaljnom pretraživanju prostora dopustivih rešenja koje je praktično neizvodljivo zbog velikog broja takvih rešenja. Ova metoda veoma haotično pretražuje prostor dopustivih rešenja, te je stoga lako objasniti njenu neefikasnost prilikom traženja globalnog minimuma funkcije cilja, naročito u slučajevima problema velikih dimenzija ili problema kod kojih je prostor dopustivih rešenja izuzetno veliki. Stoga su se razvile mnoge metode koje pokušavaju da prevaziđu problem Monte-Carlo metode uvođenjem raznih pravila za sistematizaciju pretraživanja prostora dopustivih rešenja.

Osnovna ideja metaheurističkih metoda je da se polazi od jednog (ili više) mogućeg dopustivog rešenja (dobijenog konstruktivnom heurističkom metodom ili na slučajan način) i pokušava njegova popravka sve dok se ne zadovolji neki izlazni kriterijum. Obzirom da su konstruktivne heuristike ograničene na dobijanje (najčešće jednog) suboptimalnog rešenja i da bi u mnoštvu postojećih heuristika bilo teško izabrati najpogodniju za dati problem (da ne govorimo o skupoći izvršavanja svih) značaj metaheuristika kao alata za efikasno popravljanje postojećih rešenja je evidentan. Prefiks *meta-* označava da je reč o uopštenim heuristikama, koje ne zavise od prirode problema koji se rešava. Osim toga, ovaj prefiks govori i da se kod svake metode heuristike pojavljuju na više nivoa u procesu izvršavanja. Drugim rečima jedna heuristika kontroliše izvršavanje neke druge heuristike.

Kao što je već više puta istaknuto, metaheuristike predstavljaju uopšteni skup pravila za rešavanje optimizacionih zadataka. Najčešće su bazirane na nekoj opštoj ideji ili analogiji sa prirodnim procesima (u fizici, biologiji, medicini,...). Te ideje se zatim razvijaju, modifikuju, proširuju, a često i kombinuju (hibridizacija) u cilju povećanja njihove efikasnosti. To ponekad

može iskomplikovati primenu metode jer se njihova pravila usložnjavaju, parametri umnožavaju, a samim tim se prikriva, pa i gubi osnovna ideja metode. U ovom radu opisane su osnovne varijante četiri poznate metaheurističke metode (višestartno lokalno pretraživanje, metoda promenljivih okolina, tabu pretraživanje i genetski algoritam), njihova implementacija i primena na problem raspoređivanja, a dat je i osvrt na neke modifikacije i kombinacije pomenutih metoda.

Danas su metaheuristike opšte prihvaćeno sredstvo za rešavanje problema kombinatorne i globalne optimizacije. Njihova uspešnost i rasprostranjenost može biti posledica raznih činilaca: lake su za razumevanje (obzirom na to da su bazirane na nekim prirodnim ili industrijskim procesima), generalnost primene (opšti skup pravila se lako može primeniti na raznovrsne probleme), fleksibilnost (u smislu jednostavne ugradnje specifičnosti konkretnog problema, a samim tim i elemenata veštačke inteligencije), mogućnost efikasne kontrole odnosa kvaliteta dobijenog rešenja i efikasnosti implementacije ili vremena izvršavanja. O masovnosti njihove primene najbolje govori mnoštvo opštih i specijalizovanih naučnih skupova posvećenih razvoju i primenama metaheurističkih metoda: Metaheuristic International Conference (MIC), European Workshop On Evolutionary Computation in Combinatorial Optimization (EVOCOP), razne mini konferencije i radne grupe u okviru međunarodnog udruženja društava operacionih istraživača evropskih zemalja (The Association of European Operational Research Societies, EURO)
`http://www.euro-online.org/`
ili Institut za operaciona istraživanja i poslovodne nauke (Institute for Operations Research and the Management Science, INFORMS)
`http://www.informs.org/Conf/`, i druge. Takođe se otvaraju nove stranice na INTERNET mreži na kojima se mogu pronaći najnovije informacije vezane za metaheuristike i njihove prmene. EU/ME the European chapter on metaheuristics
`http://143.129.203.3/eume/welcome.htm?eume.main.html&1`
je najpoznatija stranica opšteg karaktera koja sadrži informacije o istraživačima, o oblasti primene metaheuristika i njihovim referencama. Osim toga, postoje i prezentacije pojedinačnih metaheuristika:
- mravlje kolonije, Ant Colonies (AC)
`http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html`,
- genetski algoritmi, Genetic Algorithms (GA)
`http://www-illigal.ge.uiuc.edu/index.php3`,
memetski algoritmi, - Memetic Algorithms (MA)

http://www.densis.fee.unicamp.br/ moscato/memetic_home.html,
- neuralne mreže, Neural Networks (NN)
http://ieee-nns.org/,
- tabu pretraživanje, Tabu Search (TS)
http://www.tabusearch.net/
- metoda promenljivih okolina, Variable Neighborhood Search (VNS)
http://www.mi.sanu.ac.yu/VNS/VNS.HTM
http://vnsheuristic.ull.es/en/intro/index.php
i druge.

U ovom radu izabrane su najrasprostranjenije metaheuristike, uglavnom bazirane na lokalnom pretraživanju (MLS, VNS i TS) i biološkim procesima (GA). To su ujedno i metaheuristike koje su u literaturi već korišćene za rešavanje nekih varijanti problema raspoređivanja, tako da je moguće porediti originalne rezultate sa već publikovanim.

2.2. Višestartno lokalno pretraživanje

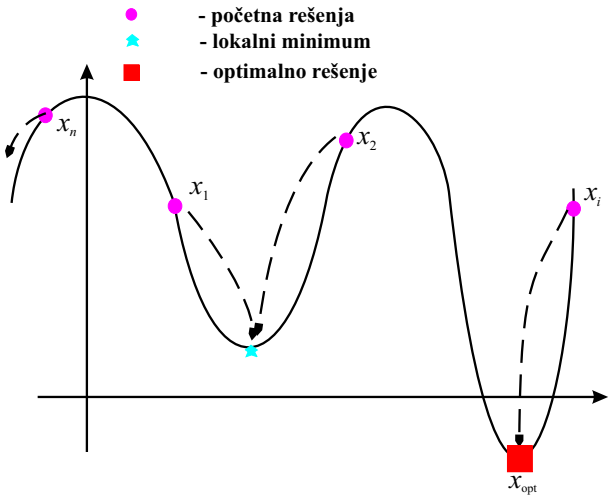
Da bi se izbegla zamka lokalnog minimuma najjednostavniji način je da se LS procedura restartuje iz novog početnog rešenja. Ta metoda naziva se *višestartno lokalno pretraživanje* (Multistart Local Search, MLS).

Višestartno lokalno pretraživanje realizuje se ponavljanjem LS procedure svaki put iz novog slučajnog rešenja. Svi dobijeni lokalni minimumi se upoređuju i najbolji među njima se proglašava za konačno rešenje. Procedura se zaustavlja kada se zadovolji neki izlazni kriterijum (broj ponavljanja, zadato vreme, broj ponavljanja bez poboljšanja i sl.).

Na slici 2.2 dat je grafički prikaz ove metode, dok se pseudokod može zapisati u sledećem obliku:

Ponavljaj
1. *Inicializacija.* Izabrati početno rešenje x (slučajno).
Ako je prva iteracija $x_{opt} = x$, $f_{opt} = f(x)$.
2. *LS.* Primeniti LS proceduru počev od x;
neka je x' dobijeni lokalni optimum.
3. *Provera rešenja.* **ako** $f(x') < f(x_{opt})$
onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$.

dok nije zadovoljen kriterijum zaustavljanja.



Sl. 2.2: Grafički prikaz metode višestartnog lokalnog pretraživanja

MLS predstavlja najjednostavniju metaheurističku metodu baziranu na lokalnom pretraživanju. Na žalost, ova metoda podleže *centralnoj graničnoj teoremi* (central limit theorem) što znači da je pokazano da ovako dobijeni minimumi teže srednjoj vrednosti lokalnih minimuma, a ne globalnom minimumu, tj. optimalnom rešenju.

2.3. Metoda promenljivih okolina

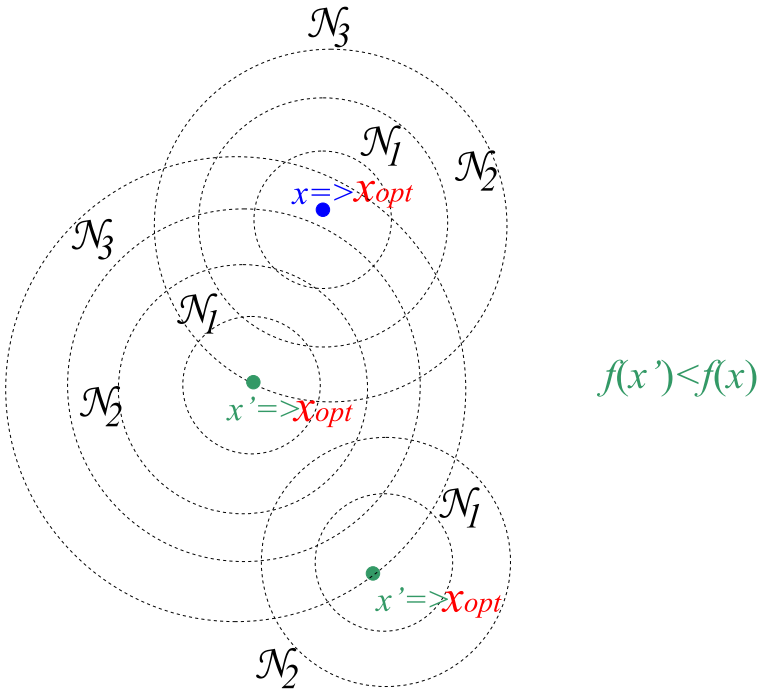
Metodu promenljivih okolina (Variable Neighborhood Search, VNS) predložili su Mladenović i Hansen [103], a prvi put ideja metode izložena je 1995. godine [102]. Osnovna ideja metode je veoma jednostavna: sistematska promena okolina unutar lokalnog pretraživanja. Dakle, neophodno je uvesti više okolina, bilo da se menja metrika u odnosu na koju se definiše okolina, bilo da se povećava rastojanje u odnosu na istu metriku. Metoda je do sada uspešno primenjena na veliki broj problema kombinatorne optimizacije i veštačke inteligencije. Predloženo je nekoliko modifikacija i poboljšanja, uglavnom namenjenih uspešnom rešavanju problema velikih dimenzija [70, 71, 72, 73, 74]. VNS metaheuristika zasnovana je na tri jednostavne činjenice [74]:

(i) lokalni minimum u odnosu na jednu okolinu ne mora biti i lokalni minimum u odnosu na neku drugu okolinu;

(*ii*) globalni minimum je lokalni minimum o odnosu na sve okoline;

(*iii*) za većinu problema lokalni minimumi u odnosu na razne okoline su međusobno bliski.

Prva činjenica opravdava uvođenje više okolina. Druga činjenica ne garantuje da rešenje koje je lokalni minimum u odnosu na svaku od izabranih okolina predstavlja i globalni minimum, već nešto malo slabije: ako neko rešenje nije lokalni minimum u odnosu na neku okolinu, onda sigurno nije globalni minimum, stoga takođe ima smisla uvođenje više okolina. Kao posledica činjenice 3 (koja je empirijska, a ne teoretska) javlja se potreba detaljnog istraživanja najbliže okoline lokalnog minimuma jer se tu očekuje popravljanje tekućeg najboljeg rešenja.



Sl. 2.3: Upotreba više okolina u lokalnom pretraživanju

Definicija 5. *Neka je $x \in X$ proizvoljno dopustivo rešenje i neka je $\mathcal{N}_k$, $(k = 1, \ldots, k_{max})$, konačna kolekcija unapred definisanih okolina. Tada je $\mathcal{N}_k(x)$*

skup rešenja u k-toj okolini od x, tj. skup rešenja koja se u odnosu na usvojenu metriku (*broj transformacija*) *nalaze na rastojanju k od rešenja x.*

Slika 2.3 ilustruje upotrebu više okolina u rešavanju optimizacionih problema. Mnoge metaheurističke metode bazirane na lokalnom pretraživanju koriste jednu, najviše dve okoline.

Upotreba više okolina postavlja dodatna pitanja na koja treba odgovoriti u okviru konkretne implementacije VNS algoritma [73]:

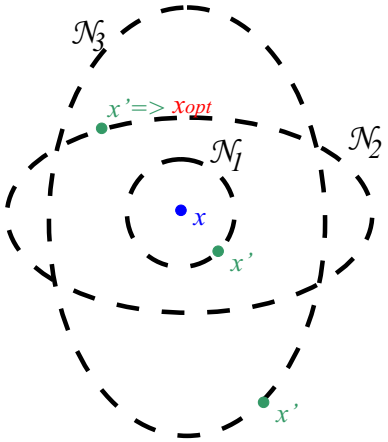
- *izbor okolina koje će se koristiti*: zavisi od toga koje se metrike mogu definisani i primeniti u datom slučaju;
- *uređenje (redosled) okolina*: definisano je usvojenom metrikom i treba da bude takvo da u odnosu na tu metriku rastojanje među rešenjima raste;
- *veličine okolina*: odnosi se na restrikcije, tj. smanjenja okolina uočavanjem i razmatranjem samo onih suseda koji potencijalno mogu da obezbede poboljšanje;
- *strategije pretraživanja*: koje mogu biti do prvog poboljšanja (FI) ili naći najbolje u okolini (BI);
- *spust i penjanje*: odnos između koraka koji obezbeđuju poboljšanje i ostalih (koji omogućavaju izbegavanje zamke lokalnog minimuma);
- *smer pretraživanja*: određuje redosled kojim će se pretraživati okoline (kod VNS metode on je definisan parametrima k_{min}, k_{max} i k_{step} o kojima će više reči biti kasnije);
- *intenzifikacija i diversifikacija pretraživanja*: načini na koje će se pretraživanje koncentrisati u okolinama u kojima se očekuje poboljšanje, ili usmeriti pretraga ka novim, neistraženim regionima prostora rešenja čime se povećava šansa nalaženja globalnog minimuma.

Pri konkretnoj implementaciji VNS metode polazi se od tri ranije navedene činjenice ($(i), (ii)$ i (iii)). Ove tri činjenice mogu se iskoristiti na tri različita načina: deterministički, stohastički ili kombinovano. Time se dobijaju tri prve verzije metode promenljivih okolina.

1) **Metoda promenljivog spusta**. Kada se koristi deterministička varijanta, dobija se metoda koju nazivamo *metoda promenljivog spusta* (Variable

Neighborhood Descent, VND). Ona se sastoji u tome da se izabere k_{max} okolina, $\mathcal{N}_k$, $k = 1, 2, ..., k_{max}$, odredi početno rešenje x i startuje LS procedura u odnosu na svaku od okolina, a počev od izabranog rešenja x. Ukoliko je $k_{max} = 1$, reč je o običnom lokalnom pretraživanju.

Grafički prikaz upotrebe više okolina u okviru VND metode dat je na slici 2.4. Grafičkom ilustracijom je istaknuto da okoline ne moraju biti "ugnježdene", tj. da ne moraju biti indukovane iz iste metrike (ili, ukoliko je reč o definiciji okoline u odnosu na neke zadate transformacije rešenja, okoline se mogu definisati primenom različitih transformacija).

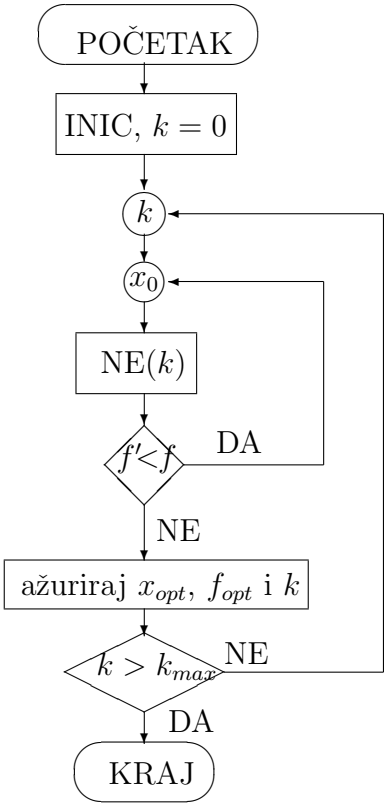


Sl. 2.4: Upotreba više okolina u metodi promenljivog spusta (VND)

Konkretno, VND znači da se pretražuje okolina $\mathcal{N}_1(x)$ dok god u njoj postoji mogućnost popravljanja trenutno najboljeg rešenja. Kada se odredi lokalni minimum x' u odnosu na okolinu $\mathcal{N}_1$, ažurira se tekuće najbolje rešenje (postavi $x_{opt} = x'$), startuje se LS procedura u odnosu na okolinu $\mathcal{N}_2(x_{opt})$. Čim dođe do popravke trenutno najboljeg rešenja x_{opt}, pretraživanje se vraća u okolinu $\mathcal{N}_1$ novog najboljeg rešenja. Ukoliko se x_{opt} ne popravi, procedura lokalnog pretraživanja se usmerava na narednu neispitanu okolinu $\mathcal{N}_k(x_{opt})$. VND procedura se završava ako je nemoguće popraviti trenutno najbolje rešenje x_{opt} ni u jednoj okolini, tj. ako je x_{opt} lokalni minimum u odnosu na sve okoline $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k_{max}}$. Blok dijagram izvršavanja VND metode prikazan je na slici 2.5 pri čemu NE(k) predstavlja pretraživanje k-te okoline datog rešenja, tj. svaku pojedinačnu iteraciju lokalnog pretraživanja u k-toj okolini. Pseudokod sa osnovnim koracima metode izgleda ovako:

1. *Inicializacija.* Izabrati početno rešenje x, $x_{opt} = x$, $f_{opt} = f(x)$.

2. **Ponavljaj**
   (a) k=1
   (b) **Ponavljaj**
      i. *LS.* Primeniti LS proceduru počev od x u okolini $\mathcal{N}_k$; neka je x' dobijeni lokalni optimum.
      ii. *Provera rešenja.* **ako** $f(x') < f(x_{opt})$
      onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, $k = 1$
      inače $k = k + 1$
   dok je $k <= k_{max}$
dok ima poboljšanja.



Sl. 2.5: Blok-dijagram metode promenljivog spusta

2) **Redukovana metoda promenljivih okolina**. U slučaju stohastičke definicije metode koraci su utoliko jednostavniji što ne postoji proces lokalnog pretraživanja. Dakle metoda se sastoji u sistematskoj promeni okolina i izboru jednog slučajnog rešenja u svakoj od okolina. Koraci odlučivanja bazirani su na tom jednom slučajnom rešenju. Metoda koja se na ovaj način dobija naziva se *redukovana metoda promenljivih okolina* (Reduced Variable Neighborhood Search, RVNS).

RVNS je izuzetno korisna kod primera velikih dimenzija jer se izbegava složena i dugotrajna LS procedura. Ova metoda veoma liči na Monte-Carlo metodu [55], mada je donekle sistematičnija. Dok Monte-Carlo bira slučajno rešenje u celom prostoru pretraživanja, RVNS se u svakom koraku ograničava na neku, strogo definisanu okolinu. Ipak, obzirom na stepen slučajnosti, najbolji rezultati se postižu kombinacijom ove metode sa nekom drugom varijantom. Na primer, RVNS se koristi za dobijanje početnog rešenja, a zatim se primenjuje neka varijanta koja sistematično pretražuje okoline tako dobijenog početnog rešenja [24].

Pseudokod RVNS metode može se zapisati u sledećem obliku:

1. *Inicializacija.* Izabrati početno rešenje x, $x_{opt}=x$, $f_{opt}=f(x)$.

2. **Ponavljaj**
   (a) k=1
   (b) **Ponavljaj**
      i. *Razmrdavanje.* Izabrati slučajno rešenje x' u okolini $\mathcal{N}_k(x)$;
      ii. *Provera rešenja.* **ako** $f(x') < f(x_{opt})$
         onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, $k = 1$
         inače $k = k + 1$
   dok nije $k = k_{max}$

dok nije zadovoljen kriterijum zaustavljanja.

Uobičajeni kriterijum zaustavljanja u ovom slučaju je $nimp_{iter}$, maksimalni broj iteracija između dva poboljšanja. Naravno, mogu se koristiti i svi ostali navedeni kriterijumi, zavisno od konkretne primene i zahteva koje treba zadovoljiti prilikom rešavanja svakog pojedinačnog optimizacionog zadatka.

3) **Metoda promenljivih okolina**. Kombinacijom prethodna dva principa, tj. sistematskom (determinističkom) promenom okolina, slučajnim izborom početnog rešenja u tekućoj okolini i primenom LS procedure počev od

tog, slučajnog rešenja dobija se *osnovna metoda promenljivih okolina* ((Basic) Variable Neighborhood Search, VNS). Ovo je najrasprostranjenija varijanta metode promenljivih okolina jer obezbeđuje više preduslova za dobijanje kvalitetnijih konačnih rešenja.

Osnovni koraci VNS metode sadržani su u petlji u okviru koje se menja indeks okoline k, određuje slučajno rešenje iz k-okoline, izvršava se procedura lokalnog pretraživanja i proverava kvalitet dobijenog lokalnog minimuma. Ovi koraci se ponavljaju sve dok ne bude zadovoljen neki kriterijum zaustavljanja. Početno rešenje u okolini k generiše se na slučajan način kako bi se obezbedilo pretraživanje različitih regiona prilikom sledećeg razmatranja okoline k. Okoline se mogu razlikovati po osnovu rastojanja (broja transformacija) ili po osnovu metrike (vrste transformacija). Važno je napomenuti da okoline za razmrdavanje (izbor slučajnog rešenja) i lokalno pretraživanje ne moraju biti istog tipa.

Opisani koraci mogu se ilustrovati pseudokodom na sledeći način:

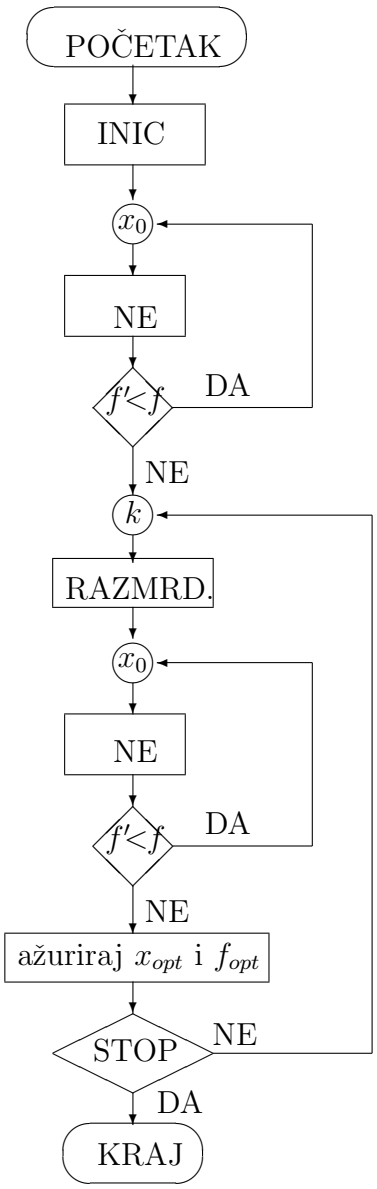
Inicializacija. Izabrati početno rešenje $x \in X$;
definisati kriterijum zaustavljanja; STOP = 0.

Ponavljaj
1. $k = 1$;
2. **Ponavljaj**
   (a) *Razmrdavanje.* Generisati slučajno rešenje x' u k-toj okolini od x, $(x' \in N_k(x))$;
   (b) *LS.* Primeniti neku proceduru lokalnog pretraživanja počev od x'; označiti sa x'' dobijeni lokalni minimum;
   (c) *Provera rešenja.* Ako je lokalni minimum bolji od trenutnog minimuma, preći u to rešenje $(x = x'')$; nastaviti od novog početnog rešenja u okolini $\mathcal{N}_1$ $(k = 1)$; inače preći u sledeću okolinu, tj. $k = k + 1$.
   (d) *Provera završetka.* Ako je zadovoljen kriterijum zaustavljanja, STOP = 1.

   dok nije $k == k_{max}$ ili STOP == 1;

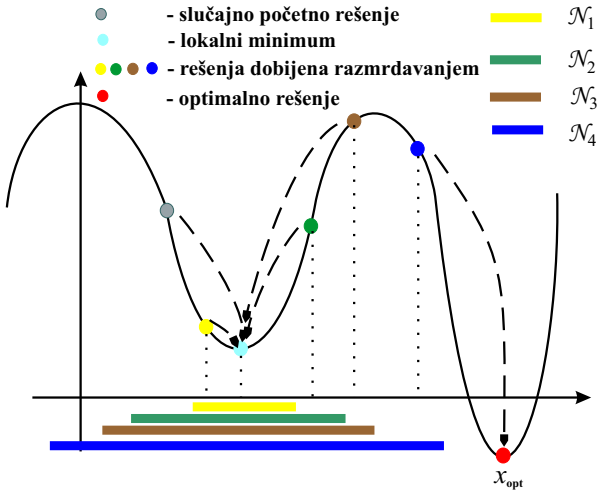
dok nije STOP == 1;

Na slici 2.6 predstavljen je blok-dijagram VNS metode. Ovde je sa NE označena jedna iteracija lokalnog pretraživanja. Slike 2.7 i 2.8 sadrže moguće

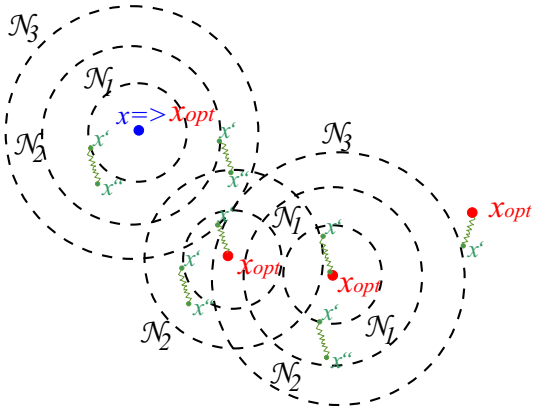


Sl. 2.6: Blok-dijagram VNS metode

grafičke prikaze primene VNS metode na minimizaciju funkcija u ravni (R^2) i prostoru (R^3).



Sl. 2.7: Ilustracija izvršavanja VNS metode za funkciju jedne promenljive



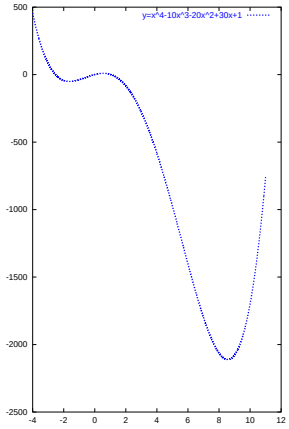
Sl. 2.8: Ilustracija izvršavanja VNS metode u prostoru

Metoda VNS u toku svog izvršavanja forsira "spust", uvek se prelazi u bolje rešenje i to u prvo na koje se naiđe, tj. metoda je *First–Improvement* (FI) karaktera. Osnovni parametar VNS metode je k_{max}–maksimalan broj okolina [72, 103]. Kriterijum zaustavljanja može biti maksimalno dozvoljeno vreme izvršavanja, maksimalni broj iteracija, tj. koliko puta se dostigne k_{max},

ili broj iteracija između dve popravke globalno najboljeg rešenja. Osim već pomenute kombinacije VNS metode sa RVNS, moguće je kombinovati i VNS sa VND tako što bi se LS procedura zamenila sa VND. Pri tome okoline u kojima se bira početno rešenje i okoline u okviru kojih se izvršava VND procedura mogu biti definisane na različite načine.

Za ilustraciju primene VNS metode (kao i ostalih metaheuristika koje se ovde opisuju) može poslužiti sledeći primer.

Primer. Naći minimum funkcije $y(x) = x^4-10x^3-20x^2+30x+1$ na intervalu $[-4, 11]$, pri čemu je x celobrojno. Grafik funkcije $y(x)$ dat je na slici, a relevantne vrednosti funkcije prikazane su tabelarno. Evidentno je da ova funkcija ima lokalni i globalni minimum, te stoga postoji opasnost da se procedura lokalnog pretraživanja zaustavi u lokalnom minimumu. Dakle, neophodno je primeniti neku od metaheurističkih metoda za rešavanje ovog zadatka.



x	-4	-3	-2	-1	0	1	2	3
$f(x)$	457	82	-43	-38	1	2	-83	-278
x	4	5	6	7	8	9	10	11
$f(x)$	-583	-974	-1403	-1798	-2063	-2078	-1699	-758

Da bi se primenila VNS metoda, najprirodnije je rešenje definisati kao argument funkcije (x), a niz okolina $\mathcal{N}_k(x) = \{x-k, x+k\}$, tj. par tačaka na rastojanju k od datog rešenja x. Procedura lokalnog pretraživanja izvršava se u $\mathcal{N}_1(x)$, dok se za razmrdavanje koriste sve raspoložive okoline. Neka se početno rešenje bira slučajno i neka je ono $x_0 = 0$. Vrednost funkcije u toj tački je $f(x_0) = f(0) = 1$. Procedura inicijalnog lokalnog pretraživanja zaustavlja se u lokalnom minimumu $x_{min} = -2$. Naime, polazeći od x_0 i njegove okoline $\mathcal{N}_1 = \{-1, +1\}$, u prvoj iteraciji LS procedura će izabrati tačku $x_1 = -1$ jer je tu vrednost funkcije manja. Zatim će se, u drugoj iteraciji, LS pomeriti u tačku x_1 i u odnosu na njenu okolinu $\mathcal{N}_1 = \{-2, 0\}$ izabrati novo, poboljšano rešenje $x_2 = -2$. U narednoj iteraciji, LS procedura se zaustavlja jer u okolini $\mathcal{N}_1 = \{-3, -1\}$ nema poboljšanja.

Izvršavanje VNS metode nastavlja se razmrdavanjem u okolini $\mathcal{N}_1(x_{min})$ pri čemu se na slučajan način bira jedna od tačaka iz te okoline. Bilo

koji izbor vraća LS proceduru u lokalni minimum $x_{min} = -2$. Stoga se povećava indeks okoline za razmrdavanje i vrši slučajan izbor tačke iz okoline $\mathcal{N}_2(x_{min}) = \{-4, 0\}$. Kao i u prethodnom koraku, procedura lokalnog pretraživanja iz bilo koje od ovih dveju tačaka zaustaviće se u već određenom lokalnom minimumu. Okolina $\mathcal{N}_3(x_{min})$ sadrži (u ovom primeru) samo jednu tačku, međutim, LS procedura koja od nje započinje svoje izvršavanje završava opet u x_{min}. Na redu je, dakle, okolina $\mathcal{N}_4(x_{min}) = \{2\}$. Polazeći od ove tačke, lokalno pretraživanje uspeva da pronađe globalni minimum $x_{opt} = 9$. Naravno, nije uvek ovako lako utvrditi da je to optimalno rešenje i stoga se, u složenijim primerima, izvršavanje VNS metode nastavlja sve dok se ne zadovolji neki kriterijum zaustavljanja.

Ovaj primer poslužiće i za ilustraciju izvršavanja ostalih metaheurističkih metoda (TS i GA).

Postoje brojne modifikacije i proširenja originalno definisane VNS metode [74]. One su uglavnom namenjene za rešavanje složenih problema i problema velikih dimenzija. Nekoliko jednostavnih modifikacija su: 1) prihvatanje lošijih rešenja sa nekom zadatom verovatnoćom, čime metoda postaje i penjanje i spust; 2) razmatranje svih k_{max} okolina pre nego što se donese odluka u koje rešenje preći, tj. implementacija *Best–Improvement* (BI) verzije metode; 3) umesto jednog početnog rešenja u okolini k, generiše se b slučajnih rešenja (b je novi parametar) i bira se najbolje među njima za početno rešenje LS procedure; 4) mogu se uvesti i parametri k_{min} i k_{step} sa značenjem da se ne polazi iz okoline $k = 1$, već k_{min} i da se indeks okoline ne uvećava za 1 nego za k_{step}. Ukoliko je priroda problema takva da postoji veliki broj rešenja sa istom vrednošću funkcije cilja, tj. postoje tzv. *platoi*, može se kao parametar uvesti i $p_{plateaux}$–verovatnoća prelaska u novo rešenje koje ima istu, trenutno minimalnu vrednost funkcije cilja [71, 72, 73].

Neke malo složenije modifikacije osnovnih verzija VNS metode bile bi: (a) Metoda promenljivih okolina sa dekompozicijom (Variable Neighborhood Decomposition Search, VNDS); (b) Adaptivna metoda promenljivih okolina (Skewed VNS, SVNS) i druge o čemu se više detalja može pronaći u [74].

2.4. Tabu pretraživanje

Tabu pretraživanje (Tabu Search, TS) metaheuristika opisana je detaljno u knjizi Glovera i Lagune [61]. Metodu su nezavisno razvili Glover [60] i Hansen [68] koji je formulisao heuristički princip najbržeg spusta/najsporijeg

uspona (steepest descent/mildest ascent). Osnovne ideje metode, kao i analiza efikasnosti i neki primeri primene TS metode, lepo su opisani u [32, 3, 112]. Ovde će samo ukratko biti sumirane osnovne ideje na kojima je metoda bazirana. Kao i MLS i VNS, i TS metoda zasniva proces pretraživanja na okolini tekućeg minimalnog rešenja. Kada se u toku pretraživanja naiđe na lokalni minimum, potrebno je definisati kriterijum izlaska iz njega. Taj kriterijum je kod TS metode zasnovan na upotrebi memorija. Za razliku od ostalih navedenih metoda koje čuvaju samo trenutno najbolje rešenje i odgovarajuću (trenutno minimalnu) vrednost funkcije cilja, TS metoda pamti kretanje preko rešenja posećenih u nekoliko prethodnih iteracija. Te informacije se koriste za izbor narednog rešenja kao i za modifikaciju definicije okoline u smislu izbacivanja nekih "zabranjenih" rešenja. Obzirom na to, jasno je da okolina $\mathcal{N}(x)$ rešenja x varira od iteracije do iteracije, te se stoga TS ubraja u procedure koje se nazivaju *tehnike dinamičkog pretraživanja okoline*. Preciznije, opis izvršvanja osnovne verzije TS metode sastoji se u sledećem: ova metoda sistematski prati proces minimizacije zasnovan na LS proceduri sve dok se ne dostigne lokalni minimum. Zatim se taj lokalni minimum isključuje iz okoline za pretraživanje i traži minimalno rešenje u tako modifikovanoj okolini. Isključena rešenja se pamte u listi nazvanoj *tabu lista* (TL) koja predstavlja zabranjena rešenja u toku nekoliko narednih iteracija (definisanih dužinom TL). Po isteku zabrane, ova rešenja vraćaju se u okolinu, a neka druga postaju zabranjena. Na osnovu dosada izloženog može se opisati pseudokod osnovne verzije TS metode.

Inicializacija. Izabrati početno rešenje $x \in X$;
postaviti $x_{opt} = x$, $TL = \emptyset$; izabrati kriterijum zaustavljanja.

Ponavljaj
1. *LS.* Naći $x' \in \mathcal{N}(x) \setminus TL$ za koje f ima minimalnu vrednost.
2. *Provera rešenja.* Ako je $f(x') < f(x_{opt})$ postaviti $x_{opt} = x'$
   Postaviti $x = x'$.
3. *Ažuriranje TL.* $TL = TL \cup \{x'\}$; **ako** $|TL| > NTABU$
   onda $TL = TL \setminus \{x^t\}$

pri čemu je x^t rešenje koje je najduže u tabu listi TL (FIFO princip).
dok nije zadovoljen kriterijum zaustavljanja.

Ako se koristi tabu lista promenljive dužine, kao prvi korak u pseudokodu TS metode dodaje se

1. *Ažurirati $|TL|$ (dužinu tabu liste TL).* Generisati slučajan broj iz intervala $(1, NTABU)$ koji predstavlja dužinu TL za tekuću iteraciju.

Kriterijumi zaustavljanja su uobičajeni, već pominjani: maksimalan broj iteracija, maksimalan broj iteracija između dve popravke najboljeg rešenja ili maksimalno dozvoljeno vreme izvršavanja metode.

Suština TS metode je u tome da se nakon dostignutog lokalnog minimuma posećuju rešenja koja ne vode poboljšanju vrednosti funkcije cilja sa idejom da se napusti okolina tog lokalnog minimuma i omogući potraga za novim lokalnim minimumima od kojih bi neki mogao biti i globalni. Da bi se sprečilo ponovno posećivanje nekog lokalnog minimuma, tzv. cikliranje, koristi se tabu lista (TL) u kojoj se čuvaju zabranjena rešenja.

Primer. Minimizacija funkcije $y(x) = x^4 - 10x^3 - 20x^2 + 30x + 1$ za $x \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ primenom TS metode počev od inicijalnog rešenja $x_0 = 0$ odvijala bi se na sledeći način. Kao i prilikom primene VNS metode, lokalno pretraživanje zaustavlja se u lokalnom minimumu $x_{min} = -2$. To rešenje ubacuje se u tabu listu i u njegovoj okolini određuje sledeće najbolje, a to je $x' = -1$. TS koristi samo okolinu $\mathcal{N}_1$, te u $\mathcal{N}_1(x') = \{-2, 0\}$ bira tačku $x'' = 0$ iako je lošija od tekućeg minimuma zato što je to rešenje koje nije zabranjeno (nije u tabu listi). Naravno, $x' = -1$ se ubacuje u tabu listu i nastavlja pretraživanje u odnosu na $x'' = 0$. Postupak se nastavlja i, kako je uvek tačka levo od tekućeg rešenja zabranjena, bira se desni kraj intervala što u naredna dva koraka izvlači pretragu iz zamke lokalnog minimuma i usmerava je ka optimalnom rešenju. U ovom primeru dovoljno je bilo da dužina tabu liste bude 2 da bi proces pretrage izbegao zamku lokalnog minimuma. Kod složenijih primera, dužinu tabu liste treba pažljivo izabrati.

Zabranjena rešenja se mogu pamtiti pomoću nekih atributa, kako bi se smanjila memorija potrebna za njihovo smeštanje. Stoga je uobičajeno reći da se u TL pamte *zabranjeni pokreti.* Uobičajeno je da se koristi i po nekoliko tabu lista i u tom slučaju, pokret ima "tabu status" ako se nalazi u svim listama, u suprotnom, pokret je dozvoljen. Osnovna uloga tabu liste je izbegavanje zamke lokalnog minimuma i sprečavanje cikliranja u okolini nekog rešenja.

Obzirom na činjenicu da se u tabu listi čuvaju atributi rešenja (zbog uštede u prostoru), moguće je da to dovede do znatne restrikcije u prostoru pretraživanja, tj. do zabrane svih pokreta koji imaju iste atribute kao i onaj koji se zaista zabranjuje, a samim tim do sprečavanja da se poseti mnogo više

rešenja, a ne samo ona dobijena u poslednjih $|TL|$ iteracija. Sprečavanje cikliranja, takođe, nije u potpunosti zagarantovano i zavisi od dužine tabu liste. To je parametar koji treba pažljivo birati (ukoliko je prevelik, diversifikacija pretraživanja je veća i može dovesi do tzv. *slučajnog kretanja* (*random walk*); ukoliko je dužina TL isuviše mala, veća je mogućnost cikliranja). Korisnik treba da bude svestan tih činjenica i da ih na najpovoljniji način uključi u implementaciju metode [128].

Da bi se prevazišli ovi nedostaci uvodi se mogućnost ukidanja tabu statusa nekom pokretu. Najjednostavniji način za to je korišćenje tabu liste promenljive dužine o čemu je već bilo reči. Sistematičniji način je definicija tzv. praga zadovoljivosti (aspiration level) kojim se opisuju "dobra" rešenja. Najjednostavniji primer je da se skida tabu status pokretu koji vodi u novo najbolje rešenje (u odnosu na funkciju cilja).

Još jedna specifičnost TS metode je upotreba različitih vrsta memorije. Postoje tzv. *kratkoročne memorije*, zatim *srednjoročne* i *dugoročne* memorije. Kratkoročne memorije se koriste prilikom formiranja tabu lista i služe za izbegavanje cikliranja pretraživanja u okolini nekog "dobrog" rešenja. Srednjoročne memorije imaju zadatak da obezbede intenzifikaciju pretraživanja u nekom regionu u kome se nalazi trenutno najbolje rešenje, tzv. proces intenzifikacije, dok je uloga dugoročnih memorija diversifikacija, tj. prenošenje procesa pretraživanja u još neistražene regione prostora dopustivih rešenja. Da bi se to obezbedilo, pamte se neki definisani "atributi" posećenih rešenja. U procesu intenzifikacije najčešće se biraju rešenja koja imaju veći broj promenljivih ili "atributa" zajedničkih sa trenutno najboljim rešenjem, dok se u procesu diversifikacije, favorizuju rešenja kod kojih su ti atributi različiti.

Način definicije okoline, izbora atributa koji se prate i pamte, kao i ostalih parametara TS metode (dužina tabu liste, izbor kriterijuma zaustavljanja, definicija praga zadovoljivosti), zavise od konkretnog problema koji se rešava kao i od same implementacije metode za neki dati problem. Opšta pravila metode dovoljno su fleksibilna da omogućavaju efikasne implementacije za najraznovrsnije probleme. Ukoliko se u implementaciju uključi i znanje o strukturi samog problema, moguće je dobiti inteligentno pretraživanje (pretraživanje sa elementima automatskog rezonovanja).

Postoji izuzetno veliki broj radova koji se bave primenom TS metode na rešavanje raznih problema kombinatorne i globalne optimizacije, ali veoma malo njih se bavi teorijskim razmatranjima njene konvergencije.

2.5. Genetski algoritam

Genetski algoritam (Genetic Algorithm, GA) predstavlja metaheuristiku koja koristi veći broj rešenja (populaciju) i bazirana je na principima prirodne evolucije. Metodu je prvi predložio J. Holland [76], dok je detaljan opis dat u [62]. Osnovne ideje metode mogu se naći i u [32, 52, 107, 112]. GA ne uključuje eksplicitno proceduru lokalnog pretraživanja, već vrši modifikacije rešenja po nekim, unapred zadatim, pravilima ili kombinovanje dva (ili više) rešenja u cilju generisanja novih i nalaženja među njima boljih od trenutno najboljeg rešenja. To je proces koji treba da simulira reprodukciju živih organizama u prirodi. Rešenja, koja su članovi neke *populacije* mešaju se i proizvode *potomke* koji bi trebalo da zadržavaju dobre osobine svojih *predaka*. Ovaj proces može se zamisliti kao LS u generalisanom smislu, pretražuje se okolina ne jednog rešenja, nego više rešenja iz populacije (ili čak svih rešenja iz populacije).

Osnovne ideje metode (po ugledu na prirodnu evoluciju) su da se izabere inicijalna populacija i da se zatim kroz niz generacija evoluiranja te populacije vrši popravka trenutno najboljeg rešenja primenom tzv. *operatora*, tj. transformacija polaznih rešenja i generisanja potomaka. Osnovni operatori su *ukrštanje*, *mutacija* i *selekcija* [32, 52, 62, 107, 112].

Jedna od bitnih odlika GA je da se operatori ne primenjuju u okviru samog prostora rešenja, već se najpre vrši kodiranje rešenja nizovima simbola nekog konačnog alfabeta. Najjednostavnije je binarno kodiranje ($0-1$ simbolima), ali su u upotrebi i složeniji načini kodiranja. U daljem razmatranju će se uvek pod rešenjem podrazumevati njegova reprezentacija pomoću izabranog koda. Takva reprezentacija naziva se *jedinka* populacije ili *hromozom*. Svaki od simbola u kodu naziva se *gen*. Ponekad je moguće da se pod genom podrazumeva grupa simbola (recimo, binarna reprezentacija cifara dekadnog sistema), ali se to ređe koristi.

Ukratko opisano, proces izvršavanja GA metode sastoji se u sledećem. Izabere se inicijalna populacija koja sadrži N_p jedinki (rešenja). Primenom genetskih operatora, polazna populacija evoluira kroz generacije kreirajući u svakoj narednoj generaciji populaciju iste veličine. Proces evolucije odvija se tako što se u svakoj generaciji primene redom operatori selekcije, ukrštanja i mutacije.

Detaljnije, u svakoj generaciji se najpre odredi "kvalitet" svakog rešenja (jedinke) izračunavanjem vrednosti funkcije prilagođenosti (fitness). Funkcija prilagođenosti najčešće se definiše kao normalizovano rastojanje konkretne

jedinke od najbolje jedinke u populaciji ili od trenutno najboljeg rešenja. Na osnovu njene vrednosti, različitim mehanizmima selekcije (rulet, turnir, uniformna, elitistička [62]) biraju se jedinke koje će dobiti šansu da pređu u narednu generaciju, bilo nepromenjene bilo kroz svoje potomke. Po ugledu na evolucione procese u prirodi, pretpostavlja se da će bolje prilagođene jedinke imati više šanse da dovedu do popravljanja trenutno najboljeg rešenja.

Rešenja se zatim grupišu u parove čijim se ukrštanjem dobijaju po dva potomka. Operator ukrštanja (crossover) može se definisati na razne načine, čak se može uključiti i više od dve jedinke i proizvesti više od dva potomka. Uobičajena procedura je da se izaberu dve jedinke, na slučajan način se izabere pozicija u kodu i izvrši zamena mesta elementima koda počev od te pozicije. Ilustracije radi, neka su jedinke neke populacije kodirane binarnim nizovima dužine 8 simbola. Izabrane su jedinke i pozicija 3. Rezultat ukrštanja dobija se na sledeći način

$$\begin{array}{cccccccc} 1 & 0 & 1| & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0| & 0 & 1 & 1 & 1 & 1 \end{array} \quad \rightarrow \quad \begin{array}{cccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Osim ovako definisanog operatora ukrštanja, može se koristiti i dvopozicioni, kod kojega se vrši zamena dela koda između tih dveju pozicija, pa čak i višepozicioni, gde se naizmenično razmenjuju delovi jedinki.

$$\begin{array}{cccccccc} 1 & 0| & 1 & 1| & 0 & 0| & 1 & 0 \\ & & \updownarrow & & & & \updownarrow & \\ 0 & 0| & 0 & 0| & 1 & 1| & 1 & 1 \end{array} \quad \rightarrow \quad \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Postoje i druge definicije operatora ukrštanja, a konkretan izbor najčešće zavisi od samog problema koji treba rešiti kao i od izbora kodiranja za rešenja.

Kako je broj mogućih izbora po dve jedinke za primenu operatora ukrštanja veliki (a ni u prirodi ne dolazi do ukrštanja svake jedinke sa svakom drugom) razvijeni su različiti mehanizmi izbora jedinki na koje će biti primenjen operator ukrštanja. Najčešće se jedinke biraju na osnovu redosleda (indeksa) u populaciji. Ponekad se operator ukrštanja primenjuje samo na jedan par jedinki i nakon primene sledećeg operatora (mutacije) odmah se formira nova generacija, koja se od prethodne razlikuje u samo dve jedinke [88]. Ovakav postupak je prilično restriktivan, ali može biti vrlo efikasan, naročito u početnim fazama izvršavanja algoritma.

Na svaki par jedinki, sa zadatom verovatnoćom p_{xover} primenjuje se operator ukrštanja. Primena operatora sa verovatnoćom znači da na neke jedinke operator neće biti primenjen i one će dobiti šansu da nepromenjene pređu u sledeću generaciju. Verovatnoća p_{xover} predstavlja okvirni procenat jedinki

koje će proizvesti potomke i time dati mogućnost da se u tekućoj generaciji pronađe novo najbolje rešenje. Osnovni cilj primene operatora ukrštanja treba da bude očuvanje dobrih osobina trenutno najkvalitetnijih jedinki.

Potomci, dobijeni primenom operatora ukrštanja, podvrgavaju se zatim mutaciji, sa nekom (malom) verovatnoćom p_m. Uloga mutacije je da obezbedi diversifikaciju prilikom pretraživanja prostora rešenja. Nasuprot ukrštanju koje teži zadržavanju osobina dobrih jedinki, rezultat mutacije treba da bude "promena" genetskog materijala koja će usmeriti pretragu ka još uvek neistraženim regionima. Imajući u vidu činjenicu da su lokalni minimumi međusobno bliski (činjenica (*iii*) na kojoj se zasniva VNS metoda), operator mutacije primenjuje se sa mnogo manjom verovatnoćom.

Najjednostavnija definicija operatora mutacije je da se na slučajno izabranoj poziciji u jedinki (kodu rešenja) trenutni simbol alfabeta zameni nekim drugim simbolom. Na primer, mutacija na poziciji 5 daje

$$1 \quad 0 \quad 1 \quad 1 \quad \underline{0} \quad 0 \quad 1 \quad 0 \quad \rightarrow \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0$$

Nakon završetka svih operacija nad jedinkama tekuće populacije dobija se novih N_p jedinki koje, zajedno sa jedinkama tekuće populacije, konkurišu za ulazak u sledeću generaciju. Znači, njima se računa vrednost funkcije fitnesa i na osnovu nje vrši selekcija jedinki za ulazak u novu generaciju. Kao što je već napomenuto, i operator selekcije može se definisati na različite načine. Najjednostavnija je elitistička selekcija koja fiksirani broj najboljih jedinki obavezno prenosi u novu generaciju. Uniformna selekcija izbacuje određeni broj najgorih jedinki i na njihovo mesto uvodi najbolje novodobijene jedinke. Međutim, manjkavost ovako definisanih operatora selekcije je što forsiraju intenzifikaciju pretraživanja. Najjednostavnija, rulet selekcija, za narednu generaciju bira jedinke na slučajan način, s tim što je definisana verovatnoća slučajnog izbora, proporcionalna vrednosti funkcije prilagođenosti. Ovim se postiže da se ipak veća šansa za "preživljavanje" daje bolje prilagođenim jedinkama. Manji stepen slučajnosti postignut je kod turnirske selekcije gde se u novu generaciju prenose najbolje jedinke iz slučajno izabrane grupe koja sadrži unapred zadati broj jedinki. Operator selekcije, u opštem slučaju, dozvoljava da se neka jedinka pojavi više puta u tekućoj populaciji. Međutim, da bi se forsirala raznovrsnost jedinki, potrebno je ukloniti višestruka pojavljivanja (ukoliko provera nije isuviše složena).

Kriterijum zaustavljanja, može biti maksimalni broj generacija, broj generacija bez popravke trenutno najboljeg rešenja, prevelika sličnost jedinki ili maksimalno dozvoljeno vreme izvršavanja (koje je najrasprostranjeniji kri-

terijum jer omogućava poređenje sa drugim metodama).

Na osnovu prethodnih obrazloženja, pseudokod GA bi imao sledeći oblik:

Inicijalizacija. Generisati Np različitih rešenja (pomoću konstruktivnih heuristika i/ili slučajno).

Ponavljaj

1. *Ocenjivanje.* Izračunati ocenu (fitness) svakog rešenja u populaciji
2. *Ažuriranje.* Zapamtiti x', rešenje sa najboljom ocenom (najmanjom vrednošću fitness-a).
3. *Kreiranje nove generacije.*
   (a) *Selekcija.* Izabrati rešenja koja ulaze u sastav nove generacije.
   (b) *Ukrštanje.* Sa zadatom verovatnoćom, izabrati parove rešenja i primeniti operaciju ukrštanja (rekombinaciju) čime se dobijaju potomci izabranih rešenja.
   (c) *Mutacija.* Svaku komponentu svakog od potomaka mutirati (modifikovati) sa zadatom verovatnoćom.

dok nije zadovoljen kriterijum zaustavljanja.

Primer. Kada se genetski algoritam primenjuje na minimizaciju funkcije $y(x) = x^4 - 10x^3 - 20x^2 + 30x + 1$ za celobrojno $x \in [-4, 11]$, rešenja se mogu kodirati binarnim nizovima dužine 4. To omogućava da se jednoznačno prikaže bilo koji ceo broj iz intervala $[0, 15]$. Da bi se rekonstruisalo rešenje ovog zadatka, dovoljno je od broja koji je kodiran binarnim nizom oduzeti 4 (time se interval $[0, 15]$ prevodi na $[-4, 11]$).

Neka populacija sadrži 4 jedinke i neka je inicijalna populacija 0100, 1010, 1001, 0010. Time su kodirana redom rešenja $x = 0$, $x = 6$, $x = 5$, $x = -2$. U ovoj populaciji, rešenje koje odgovara minimumum funkcije je $x_{min} = 6$. Primenom operatora ukrštanja na parove jedinki inicijalne populacije, može se dobiti sledeći skup novih jedinki

$$\begin{array}{cccc} 0 & 1| & 0 & 0 \\ 1 & 0| & 1 & 0 \end{array} \rightarrow \begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{cccc} 1 & 0 & 0| & 0 \\ 0 & 0 & 1| & 0 \end{array} \rightarrow \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}$$

Među njima su dve jednake, ali se to može regulisati primenom mutacije. Na primer, u trećoj jedinki se drugom genu promeni vrednost i dobija se 1100.

Ukoliko bi se ipak desilo da se i posle mutacije neke jedinke ponavljaju, one se jednostavno mogu eliminisati, a prilikom generisanja nove populacije izabrati više jedinki iz prethodne generacije. U ovom slučaju, novu generaciju neka sačinjavaju najbolja jedinka iz prethodne i tri nove dobre jedinke. To znači 1010, 1000, 0110 i 1100. Pri tome je popravljeno tekuće najbolje rešenje, jer ova populacija sadrži $x_{min} = 8$.

Ponavljanjem postupka, ukrštanja, mutacije i selekcije, od druge generacije dobija se treća. Na primer,

$$\begin{array}{cccc} 0 & 0| & 1 & 0 \\ 1 & 1| & 0 & 0 \end{array} \rightarrow \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array}$$

$$\begin{array}{cccc} 1| & 0 & 0 & 0 \\ 0| & 1 & 1 & 0 \end{array} \rightarrow \begin{array}{cccc} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}$$

Mutira se poslednji gen druge jedinke, a zatim se nova generacija formira od najbolje jedinke iz prethodne i tri dobre iz tekuće. Time se dobijaju sledeće jedinke 1100, 1000, 1111, 1110, koje odgovaraju rešenjima $x_1 = 8$, $x_2 = 4$, $x_3 = 11$, $x_4 = 10$. Tekuće najbolje rešenje u ovoj generaciji nije popravljeno. Za četvrtu generaciju, mogući scenario bio bi

$$\begin{array}{cccc} 1 & 1 & 0| & 0 \\ 1 & 1 & 1| & 1 \end{array} \rightarrow \begin{array}{cccc} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}$$

$$\begin{array}{cccc} 1 & 1| & 1 & 0 \\ 1 & 0| & 0 & 0 \end{array} \rightarrow \begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{array}$$

Radi unošenja raznovrsnosti, mutira se prvi gen treće jedinke, i izborom jedinke 1100 iz prethodne generacije i novodobijenih jedinki 1101, 1110 i 1010, formira se četvrta generacija. Ona sadrži $x_{min} = 9$, novo najbolje rešenje (kodirano jedinkom 1101). To je ujedno i globalni minimum, ali je to teško pokazati, te se postupak nastavlja do zadovoljenja kriterijuma zaustavljanja.

2.6. Kombinovanje metoda, hibridizacija

Bez obzira na efikasnost implementacije, nijedna metaheuristička metoda ne garantuje optimalnost dobijenog rešenja. Čak je izuzetno teško predvideti njegov kvalitet, tj. njegovo odstupanje od optimalnog rešenja. Odlika heurističkih (pa i metaheurističkih) metoda je da ne postoji univerzalna

metoda koja daje najbolja rešenja za sve probleme. Kod nekih problema pogodnija je jedna, za neke druge, efikasnijom se pokazuje druga heuristika. Ova karakteristika može se odnositi i na različite varijante istog problema.

Da bi se povećala efikasnost metaheurističkih metoda, u poslednje vreme u literaturi se pojavljuju kombinacije raznih metoda, tzv. hibridi. Strategije kombinovanja metoda i klasifikacija tako dobijenih hibrida detaljno su opisani u [127]. Najčešće se kombinuju GA i TS, a kombinacija GA i LS evoulirala je u novu metaheuristiku poznatu pod imenom *memetski algoritam* (Memetic Algorithm, [105]). Iako relativno nova, VNS metoda takođe se koristila za kombinovanje, a jedna od mogućih hibridizacija VNS i TS po prvi put je predložena u ovom radu. Jedan od rezultata dobijenih u toku izrade ovog rada je i kombinacija VNS metode sa LP opisana u [46], dok je hibridizacija ovih metoda po prvi put predložena u [75].

Primeri hibridizacije metaheurističkih metoda ukratko opisani u ovom radu ograničeni su na kombinacije prethodno opisanih metoda. Kombinacije ostalih metoda takođe postoje (na primer, TS i SA, VNS i GRASP), kao i kombinacije metaheurističkih sa egzaktnim metodama (TS i LP, GA i DP[1]), a njihov opis može se naći i preglednim člancima [74, 112, 127] i zbornicima sa međunarodnih konferencija o metaheuristikama (MIC [2]).

U radu [106] predložena je TS metoda koja se izvršava nad populacijom rešenja. U procesu diversifikacije, nova rešenja se dobijaju ukrštanjem parova rešenja iz populacije. Hibridizacija TS i VNS metoda moguća je na dva osnovna načina: primena TS unutar VNS metode i obratno, tj. upotreba ideja VNS metode u okviru TS [74]. U navedenom radu opisano je nekoliko primera implementiranih hibrida. VNTS metoda [121] dobijena je zamenom LS dela u okviru VNS metode tabu pretraživanjem. U svakoj od okolina, nakon procedure razmrdavanja, primenjuje se TS za dobijanje poboljšanja polaznog rešenja. Multi-level TS sastoji se u tome da se definiše nekoliko okolina (nivoa) i u okviru svake od tih okolina primenjuje TS nezavisno, tj. tabu liste su lokalne za svaki od nivoa. Varijanta ReVND koristi tabu listu za smanjivanje okolina koje ulaze u sastav VND metode. U okviru TS metode, VNS ideje se mogu koristiti na više načina. Jedan od njih je upotreba VND umesto LS, a moguće je i ideje operacije razmrdavanja koristiti za diversifikaciju u okviru TS metode.

Detaljan opis hibrida predloženih u ovom radu i rezultati njihove primene dati su na kraju glave 4.

[1]DP – dinamičko programiranje (Dynamical Programming)

3. Formulacija problema raspoređivanja zadataka na višeprocesorske sisteme

U ovoj glavi opisan je matematički model problema statičkog raspoređivanja zadataka na višeprocesorske sisteme. Date su dve formulacije, standardna kombinatorna koja koristi jezik teorije grafova i nova formulacija, zasnovana na matematičkom programiranju. Na početku je dat kratak uvod koji sadrži definicije pojmova iz teorije grafova i matematičke analize koji se koriste u formulaciji problema. Nakon formulacija, dat je opis nekih specijalnih slučajeva zajedno sa pregledom radova u kojima su predloženi algoritmi polinomne složenosti za njihovo rešavanje. Navedene su neke najpoznatije konstruktivne heuristike za suboptimalno rešavanje, predložene u literaturi, sa naglaskom na one koje se pojavljuju kao baza za implementaciju metaheurističkih metoda.

Postoji nekoliko mogućih formulacija problema statičkog raspoređivanja zadataka. Na primer, u [15, 98] korišćeni su skupovi instrukcija koje svaki zadatak zahteva, dok su višeprocesorski sistemi opisivani dužinama izvršavanja svake od tih instrukcija. Ovde su date dve formulacije, kombinatorna preko grafova i formalna matematička na osnovu linearnog programiranja. Formulacije se odnose na problem statičkog raspoređivanja na proizvoljne višeprocesorske sisteme koji se sastoje od identičnih procesora, uz razmatranje vremena koje je potrebno za komunikaciju između procesora. Ovaj problem se naziva *problem raspoređivanja sa komunikacionim kašnjenjem* (Multiprocessor Scheduling Problem with Communication Delays, MSPCD).

3.1. Uvodne definicije

Uobičajena formulacija problema raspoređivanja koja se sreće u literaturi i predstavlja osnovu za implementaciju metaheurističkih metoda korišćenih u ovom radu, zasniva se na pojmovima iz teorije grafova [30, 32]. Stoga se ovde navode definicije korišćenih pojmova.

Definicija 1. *Neka je V neprazan skup i $E \subseteq V \times V$ skup uređenih parova elemenata iz V. Uređen par $G = (V, E)$ se naziva* graf. *Elementi skupa V su* čvorovi *grafa, a elementi skupa E* grane.

Definicija 2. *Graf je* simetričan *ili* neorijentisan *ukoliko za svaki par $v_i, v_j \in V$ važi $(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E$. Tada kažemo da su grane neorijentisane. Ukoliko simetričnost ne važi, graf nazivamo* usmereni, orijentisani, *tj. razlikuju se polazni i dolazni čvor grane. Orijentisane grane nazivamo* lukovima.

Definicija 3. Put *u orijentisanom grafu je uređeni skup lukova takvih da je dolazni čvor svakog luka, polazni za njegovog sledbenika.* Dužina puta *je broj lukova koji ga obrazuju.*

U neorijentisanom grafu svaka grana se može shvatiti kao dvostrano orijentisana, pa put nije definisan samo nizom grana, nego se za svaku granu koja ulazi u posmatrani put mora naznačiti njena orijentacija u tom putu. Stoga se često za grafove koji sadrže neorijentisane grane put dužine k definiše kao naizmenični niz čvorova v_i i grana e_i oblika

$$v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1},$$

pri čemu je za $i = 1, 2, \ldots, k$ čvor v_i polazni, a v_{i+1} dolazni za granu e_i.

Put *povezuje* čvor v_i sa čvorom v_j ako je v_i polazni čvor prve grane u putu, a v_j dolazni čvor poslednje grane.

Definicija 4. Rastojanje *između dva čvora u neorijentisanom grafu je dužina najkraćeg puta koji ih povezuje.*

Definicija 5. *Neorijentisani graf je* povezan *ukoliko postoji put između svaka dva čvora. U protivnom, graf se naziva* nepovezan. *Graf je* potpuno povezan *ukoliko postoji grana između svaka dva čvora. Za orijentisani graf kažemo da je povezan, ako je povezan njemu odgovarajući neorijentisani, tj. graf koji se dobija dodavanjem svakom luku odogovarajućeg suprotno orijentisanog luka.*

Definicija 6. *Orijentisani graf se naziva* aciklički *ukoliko ne postoji put takav da počinje i završava u istom čvoru.*

Definicija 7. *U acikličkom grafu* (neposredni) prethodnik *nekog čvora v_j je svaki čvor v_i koji je polazni čvor (izvorna tačka) luka čiji je dolazni čor v_j.*

Skup neposrednih prethodnika čvora v_j, $Pred(v_j)$ je skup svih takvih čvorova, dakle $Pred(v_j) = \{v_i \in V \mid e_{ij} \in E\}$.

Analogno se definiše skup svih (neposrednih) sledbenika datog čvora v_i, $Succ(v_i) = \{v_j \in V \mid e_{ij} \in E\}$. Za čvor koji nema prethodnika kažemo da je *koren*, dok čvorove koji nemaju sledbenike nazivamo *listovima*.

Definicija 8. *Graf se naziva* težinski *ukoliko su čvorovima i/ili granama pridruženi (realni) brojevi koje nazivamo* težinama. *Težinski graf predstavlja se uređenom četvorkom $\mathcal{G} = (V,\ E,\ L,\ C)$, gde je $V = \{v_1, \ldots, v_n\}$, skup čvorova grafa, $E = \{e_{ij} \mid v_i, v_j \in V\}$ označava skup lukova, $L = \{l_1, \ldots, l_n\}$ predstavlja težine čvorova grafa, a $C = \{c_{ij} \mid e_{ij} \in E\}$ je skup težina na lukovima.*

Definicija 9. Najduži put, kritični put*, predstavlja put koji vodi od korena do lista i ima najveću sumu težina na čvorovima i/ili lukovima.*

Definicija 10. *Najveći broj lukova na putu između nekog lista i nekog korena orijentisanog grafa naziva se* visina *grafa. Najveći broj lukova na putu između nekog čvora i korena predstavlja* visinu *tog čvora.*

Definicija 11. Širina (orijentisanog) grafa *definisana je kao najveći broj čvorova na istoj visini.*

Definicija 12. Gustina grafa ρ *definiše se kao količnik broja lukova u datom grafu i broja lukova u potpuno povezanom grafu sa istim brojem čvorova. Preciznije,*

$$\rho = \frac{|E|}{n(n-1)/2}$$

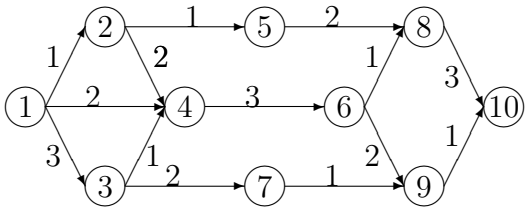
Definisani pojmovi koriste se direktno ili posredno prilikom formulacije i rešavanja problema raspoređivanja. Osim teorije grafova, koriste se i neki elementi teorije parcijalno uređenih skupova [132]. Lukovi u grafovima kojima se modelira problem raspoređivanja ustvari predstavljaju zavisnosti koje se moraju poštovati prilikom paralelnog izvršavanja programa. Oni definišu jednu relaciju parcijalnog uređenja koja omogućava smanjivanje prostora za pretraživanje prilikom primene metaheurističkih metoda.

3.2. Kombinatorna formulacija

Definisanje MSPCD pomoću grafova može se naći u više radova, na primer, [51, 88, 125]. Program koji treba paralelizovati zadaje se u obliku usmerenog acikličkog grafa (Directed Acyclic Graph, DAG) definisanog kao sledeća uređena četvorka $\mathcal{G} = (V,\ E,\ L,\ C)$, gde je $V = \{1, \ldots, n\}$, skup čvorova grafa,

koji definiše skup zadataka za raspoređivanje; $E = \{e_{ij} \mid i, j \in V\}$ označava skup lukova kojima se opisuje zavisnost među zadacima; $L = \{l_1, \ldots, l_n\}$ predstavlja težine čvorova grafa kojima se zadaju dužine izvršavanja zadataka; a $C = \{c_{ij} \mid e_{ij} \in E\}$, skup težina na lukovima, je količina komunikacije među zavisnim zadacima, komunikaciono kašnjenje, tj. količina podataka, rezultata zadatka i (prethodnika) koje zadatak j (sledbenik) koristi u svojim izračunavanjima. Komunikaciono kašnjenje $c_{ij} \in C$ predstavlja količinu podataka koja se fizički prenosi kroz višeprocesorski sistem ukoliko su zadaci i i j dodeljeni na izvršavanje različitim procesorima. Ako se oba zadatka izvršavaju na istom procesoru, komunikaciono kašnjenje je 0. Skupom E opisana je tzv. *relacija prethođenja* (*zavisnosti*) među zadacima. Zadatak i ne može početi svoje izvršavanje ukoliko svi njegovi prethodnici nisu završeni i svi potrebni podaci raspoloživi na procesoru k kome je taj zadatak dodeljen na izvršavanje. Skupom C opisuje se cena zavisnosti zadataka tj. vreme koje se mora potrošiti na prenos podataka (međurezultata) između procesora. Prekid izvršavanja i redundantna izvršavanja nisu dozvoljena.

Primer grafa zadataka dat je na slici 3.1. Težine na lukovima označavaju količinu podataka koja se razmenjuje među odgovarajućim zadacima. Brojevi u čvorovima označavaju indekse (redne brojeve) zadataka u grafu, dok su dužine izvršavanja date u tabeli.

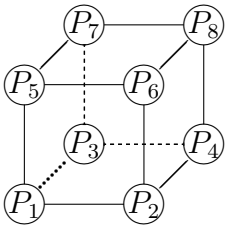


i	1	2	3	4	5	6	7	8	9	10
l_i	6	8	7	10	8	12	9	6	8	7

Sl. 3.1: Primer grafa zadataka

Pretpostavlja se da se višeprocesorski sistem $\mathcal{M}$ sastoji od p identičnih procesora od kojih svaki ima privatnu, lokalnu memoriju, a komuniciraju razmenjivanjem poruka preko dvosmernih kanala od kojih su svi istog kapaciteta. Višeprocesorski sistemi se mogu modelirati *neusmerenim grafovima* [125] ili *matricama rastojanja* (*distance matrix*) [35, 88]. Kod grafovske reprezentacije čvorovi predstavljaju procesore, a grane komunikacione kanale između pro-

cesora. Elemenat na mestu (h, k) matrice rastojanja $D = [d_{hk}]_{p\times p}$ predstavlja rastojanje između procesora h i k. Dobijena iz grafovske reprezentacije, matrica rastojanja sastoji se od rastojanja između čvorova kojima su predstavljena ta dva procesora. Lako se vidi da je matrica rastojanja simetrična matrica sa nulama po dijagonali. Na slici 3.2 prikazana je 3-dimenzionalna hiperkocka identičnih procesora i odgovarajuća matrica rastojanja ($p = 8$).



$$D = \begin{bmatrix} 0 & 1 & 1 & 2 & 1 & 2 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 1 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 1 & 2 & 2 & 3 & 0 & 1 & 1 & 2 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 2 & 3 & 1 & 2 & 1 & 2 & 0 & 1 \\ 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Sl. 3.2: 3-dimenziona hiperkocka i odgovarajuća matrica rastojanja

Zavisno od arhitekture višeprocesorskog sistema (strukture veza među procesorima definisane matricom rastojanja) na koju se vrši raspoređivanje, vreme komunikacije (communication delay) između zadataka se određuje na sledeći način

$$\gamma_{ij}^{hk} = c_{ij} \cdot d_{hk} \cdot ccr, \tag{3.0}$$

gde je c_{ij} količina podataka koju je potrebno preneti od zadatka i do zadatka j definisana skupom C, d_{hk} predstavlja rastojanje između procesora na kojima se ovi zadaci izvršavaju zadato odgovarajućim elementom matrice D, a ccr je parametar koji određuje brzinu komunikacije u okviru datog višeprocesorskog sistema i predstavlja količnik između vremena potrebnog da se izvrši jedna "elementarna operacija" i vremena potrebnog da se prenese jedinična količina podataka između dva susedna procesora. Ukoliko se zadaci izvršavaju na istom procesoru, tj. $h = k$, γ_{ij}^{kk} će biti jednako nuli, jer je $d_{kk} = 0$, za svako $k = 1, \ldots, p$.

Još jedna važna pretpostavka je da svaki procesor poseduje zasebne ulazno/izlazne procesne jedinice. To omogućava istovremeno izvršavanje aritmetičko-logičkih i ulazno-izlaznih operacija, tj. izračunavanja i komunikacije.

Problem raspoređivanja zadataka definisanih grafom $\mathcal{G}$ na višeprocesorsku arhitekturu $\mathcal{M}$ sastoji se u tome da se za svaki zadatak odredi indeks proce-

sora na kome će se on izvršavati i vremenski trenutak u kome će početi njegovo izvršavanje tako da se minimizira neka zadata funkcija cilja. Uobičajeno je (a to je i u ovom radu slučaj) da se kao funkcija cilja koristi vreme izvršavanja celog paralelnog programa $T_{max} = \max_i T_i,\ 1 \leq i \leq n$, pri čemu T_i označava vreme završetka zadatka i. T_{max} se naziva dužinom raspodele (schedule length, makespan) i često se označava sa SL. Poznate su i druge funkcije cilja, na primer, vreme komunikacije (communication delay) [123], ravnomerno opterećenje procesora (load balance) [5] i sl.

Ovako postavljen zadatak statičkog raspoređivanja nije najopštiji mogući. U ovom radu, razmatranja su ograničena na višeprocesorske sisteme koji sadrže identične procesore, tzv. homogene višeprocesorske sisteme. Uopštavanje je moguće dozvoljavanjem da ne budu svi procesori u višeprocesorskom sistemu istih karakteristika, tj. razmatranjem heterogenih višeprocesorskih sistema (kao što je to na primer slučaj u računarskim mrežama). Heterogene sisteme takođe je moguće opisati neusmerenim grafovima [98, 115, 125], u ovom slučaju težinskim, gde težine čvorova označavaju brzinu procesora u smislu izračunavanja, tj. izvršavanja računskih operacija, dok težine lukova govore o kapacitetu veza između procesora (komunikacionih kanala), tj. brzini prenosa podataka kroz višeprocesorski sistem.

Generalizacija sa homogenih na heterogene sisteme ne predstavlja veliki problem [125]. Da bi se opisala različita dužina izvršavanja nekog zadatka na različitim procesorima dovoljno je svakom zadatku (čvoru grafa zadataka) umesto težine pridružiti niz od p elemenata takav da je l_{ik} dužina izvršavanja zadatka i na procesoru k. Pri tome, dozvoljeno je da neki element l_{ik} bude ∞, sa značenjem da procesor k ne raspolaže svim resursima (na primer, nema štampač ili CD) potrebnim za izvršavanje zadatka i (tada se radi o tzv. namenskim heterogenim sistemima). Postoji i drugi pristup [98] sa opisivanjem skupa instrukcija koje svaki zadatak zahteva i dužinom izvršavanja svake od tih instrukcija na pojedinim procesorima. Slično predhodnom, vrednost ∞ za dužinu izvršavanja određene instrukcije na nekom od procesora, označava da se ta instrukcija ne može izvršiti na tom procesoru, te stoga ni jedan zadatak koji u svom skupu sadrži takvu instrukciju ne može biti dodeljen tom procesoru na izvršavanje.

Slično tome, umesto vremena starta razmatra se vreme završetka zadatka na svakom od procesora (jer pravilo da će se zadatak tamo gde najranije počinje, najranije i završiti, više ne važi) [115].

Drugo uopštenje koje se može naći u literaturi je kada zadaci (svi ili samo

pojedini) zahtevaju više od jednog procesora za svoje izvršavanje [15, 90]. Za svaki zadatak i zadaje se $\pi(i)$, broj koji označava koliko procesora taj zadatak zahteva za svoje izvršavanje. Prilikom raspoređivanja tog zadatka mora se voditi računa o tome da li postoji dovoljan broj slobodnih procesora kojima će se dodeliti taj zadatak na izvršavanje. Ovaj slučaj se u literaturi naziva *raspoređivanje višeprocesorskih zadataka* (*Multiprocessor Task Scheduling*) i ne treba ga mešati sa MSPCD što je oznaka za raspoređivanje sekvencijalnih zadataka (zadataka za čije izvršavanje nije potrebno više od jednog procesora) na višeprocesorske sisteme.

Mnogo češće od ovih, najopštijih slučajeva, u literaturi se razmatraju neki specijalni slučajevi problema statičkog raspoređivanja zadataka, o kojima će biti više reči u završnom odeljku ove glave, nakon matematičke formulacije problema u obliku u kome se razmatra u ovom radu.

3.3. Formulacija problema kao zadatka matematičkog programiranja

U literaturi postoje formalne, matematičke formulacije nekih specijalnih slučajeva problema raspoređivanja. Tako je u radu [117] data formulacija problema koji se odnosi na raspoređivanje nezavisnih zadataka. Zavisni zadaci uz zanemarivanje vremena komunikacije i strukture višeprocesorskog sistema modelirani su na razne načine u radovima [13, 94, 138].

Međutim, formulacija koja razmatra višeprocesorske sisteme proizvoljne konfiguracije i istovremeno uključuje komunikaciona kašnjenja nije pronađena. Najbliža formalna definicija jedne varijante problema raspoređivanja može se naći u [92]. U tom radu data je matematička formulacija problema raspoređivanja zavisnih zadataka na heterogenu višeprocesorsku arhitekturu uz pretpostavku da je vreme koje se troši na prenos podataka zanemarljivo malo.

Polazeći od te formulacije, problem koji se razmatra u ovom radu može se opisati restrikcijom polaznog modela datog u [92] u smislu razmatranja samo homogenih višeprocesorskih sistema (ali proizvoljne arhitekture), a zatim njegovim proširivanjem uzimanjem u obzir vremena potrebnog za prenos podataka između procesora (koje uključuje i brzinu prenosa i rastojanje među procesorima) na sledeći način.

Uočava se skup realnih promenljivih (u opštem slučaju necelobrojnih, a svakako ne 0-1 promenljivih), koje predstavljaju vremena početka izvršavanja

zadataka na procesorima. Vreme početka izvršavanja zadatka j označeno je sa t_j, dok T_j označava trenutak u kome je zadatak j već izvršen, tj. $T_j = t_j + l_j$.

Neka je

$$y_{jk}^s = \begin{cases} 1, & \text{ako se zadatak } j \text{ kao } s\text{-ti izvršava na procesoru } k, \\ 0, & \text{inače,} \end{cases}$$

za svako $j \in V$, $k \in \mathcal{M}$. Sada se MSPCD može formulisati ovako:

Naći

$$\min_{y,t} \max_{j \le n} T_j \tag{3.1}$$

pod uslovom da su zadovoljena sledeća ograničenja:

$$\sum_{k=1}^{p} \sum_{s=1}^{n} y_{jk}^s = 1, \quad j = 1, 2, \ldots, n \tag{3.2}$$

$$\sum_{j=1}^{n} y_{jk}^1 \le 1, \quad k = 1, 2, \ldots, p \tag{3.3}$$

$$\sum_{j=1}^{n} y_{jk}^s \le \sum_{j=1}^{n} y_{jk}^{s-1}, \quad k = 1, 2, \ldots, p, \quad s = 2, \ldots, n \tag{3.4}$$

$$\begin{aligned} t_j \;\; \ge \;\; & t_i + l_i + \sum_{h=1}^{p} \sum_{s=1}^{n} \sum_{k=1}^{p} \sum_{r=1}^{n} \gamma_{ij}^{hk} \cdot y_{ih}^s \cdot y_{jk}^r, \\ & i \in Pred(j), \quad j = 1, 2, \ldots, n \end{aligned} \tag{3.5}$$

$$\begin{aligned} t_j \;\; \ge \;\; & t_i + l_i - \alpha \cdot \left[2 - \left(y_{ik}^s + \sum_{r=s+1}^{n} y_{jk}^r \right) \right], \\ & k \in \mathcal{M}, \quad s = 1, 2, \ldots, n-1, \quad i,\ j \in V \end{aligned} \tag{3.6}$$

$$y_{jk}^s \in \{0, 1\}, \quad j = 1, 2, \ldots, n, \quad k = 1, 2, \ldots, p, \quad s = 1, 2, \ldots, n \tag{3.7}$$

$$t_j \ge 0, \quad j = 1, 2, \ldots, n \tag{3.8}$$

gde $\alpha \gg 0$ predstavlja pogodno izabran dovoljno veliki težinski koeficijenat, a γ_{ij}^{hk} predstavlja vreme komunikacije između zadataka i i j i definisano je sa (3.0).

Ograničenje 3.2 obezbeđuje da svaki zadatak bude dodeljen tačno jednom procesoru. Nejednakosti (3.3-3.4) znače da ni jedan procesor ne može

istovremeno izvršavati više od jednog zadatka. (3.3) znači da najviše jedan zadatak može biti prvi na procesoru k, dok (3.4) obezbeđuje da za svaki s-ti ($s \geq 2$) zadatak raspoređen na procesor k mora postojati $(s-1)$-vi zadatak raspoređen na isti procesor. Nejednakosti (3.5) izražavaju relacije zavisnosti: zadatak j ne može početi svoje izvršavanje dok se svi njegovi prethodnici ne izvrše i svi potrebni podaci ne prenesu sa procesora na kojima su zadaci-prethodnici izvršavani.

Ograničenja (3.6) definišu niz vremena početaka izvršavanja skupa zadataka dodeljenih istom procesoru. Ona izražavaju činjenicu da zadatak j mora da počne najmanje l_i jedinica vremena nakon početka zadatka i kad god se i izvršava pre j na istom procesoru k. Uloga koeficijenta α je da pokrije slučajeve kada se i izvršava posle j i kad se i i j ne izvršavaju na istom procesoru. U ovom drugom slučaju j može da počne pre nego što se i završio, pa čak i pre nego što je započeo, ukoliko su nezavisni, bez obzira na to što je j r-ti, a i, s-ti ($r > s$) zadatak na odgovarajućim procesorima.

Ovako data matematička formulacija predstavlja mešovito bilinearno programiranje: y_{ik}^s su 0-1 promenljive, a t_i su realne promenljive. Međutim, moguće je linearizovati dati model, tj. prebaciti ga u model linearnog programiranja (Linear Programming, LP) uvođenjem nove grupe 0-1 promenljivih $z_{ijhk}^{sr} = y_{ih}^s \cdot y_{jk}^r$.

Promenljive z_{ijhk}^{sr} treba da zadovolje sledeće, dodatne, uslove:

$$y_{ih}^s \geq z_{ijhk}^{sr} \tag{3.9}$$

$$y_{jk}^r \geq z_{ijhk}^{sr} \tag{3.10}$$

$$y_{ih}^s + y_{jk}^r - 1 \leq z_{ijhk}^{sr} \tag{3.11}$$

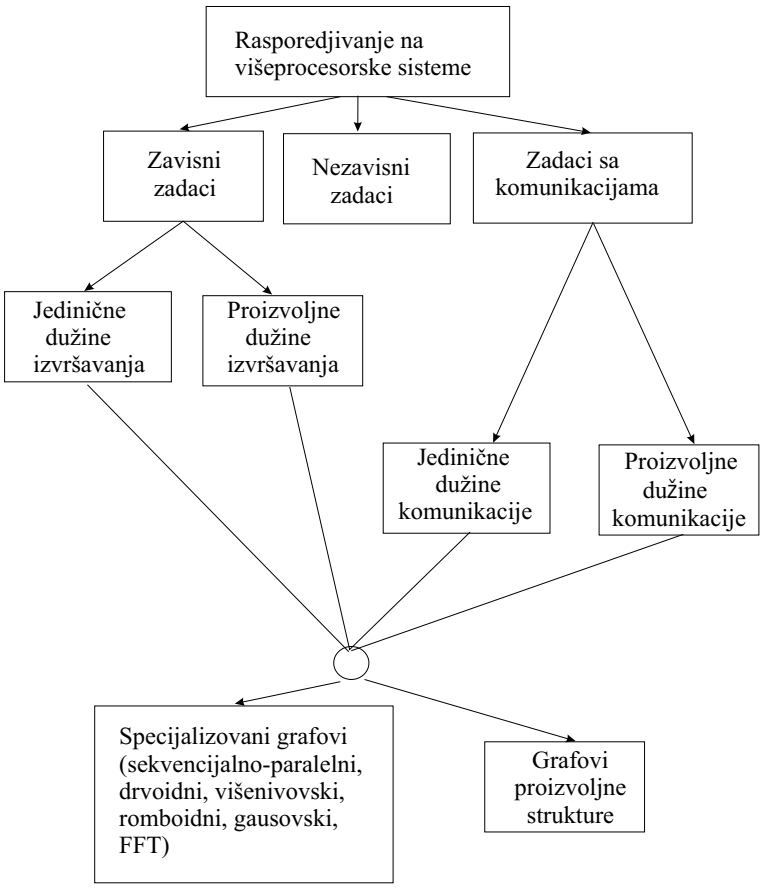
$$z_{ijhk}^{sr} \geq 0 \tag{3.12}$$

Ovako definisane promenljive z_{ijhk}^{sr} se zatim ubacuju u 3.5 umesto odgovarajućeg proizvoda i dobija se linearizovani model. Opisani matematički model, proširen za slučaj heterogenih višeprocesorskih sistema (koji se ovde ne razmatraju) dat je u [47].

Obzirom na veliki broj promenljivih koje se u modelu javljaju ($O(n^6)$), formulacija problema preko matematičkog programiranja ima više teorijski značaj, a može se koristiti i za dobijanje donje granice pri egzaktnom rešavanju problema malih dimenzija. Ova formulacija iskorišćena je u radu [46] za ispitivanje efekata primene ograničenja kojima se smanjuje red sistema (reduction constraints) [91]. Rezultati prikazani u [46] potvrđuju da se red sistema

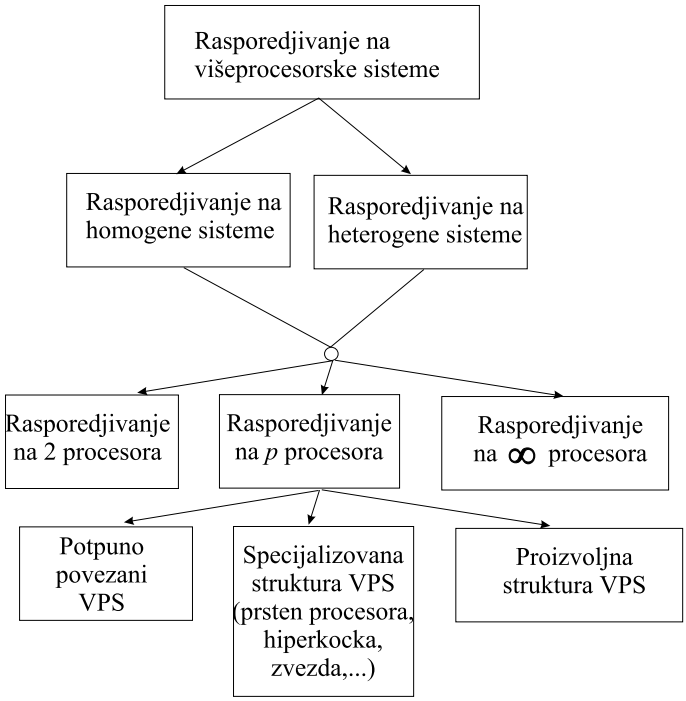
može smanjiti na $O(n^4)$, a samim tim i vreme potrebno za izvršavanje egzaktnog algoritma pri rešavanju problema malih dimenzija.

3.4. Specijalni slučajevi statičkog raspoređivanja zadataka



Sl. 3.3: Klasifikacija problema statičke paralelizacije na osnovu osobina grafa zadataka

Specijalni slučajevi problema raspoređivanja ogledaju se u uvođenju dodatnih pretpostavki na strukturu grafa zadataka i/ili višeprocesorskog sistema. Jedna od mogućih klasifikacija problema statičkog raspoređivanja



Sl. 3.4: Klasifikacija problema statičke paralelizacije na osnovu karakteristika višeprocesorskog sistema

prikazana je na slici 3.3 i odnosi se na klasifikaciju na osnovu karakteristika grafa zadataka koji se raspoređuje. Problem raspoređivanja se može klasifikovati i na osnovu osobina višeprocesorskog sistema na koji se vrši raspoređivanje i to je prikazano na slici 3.4. Svaka od ovih klasifikacija je dovoljno složena, ali se stvarna slika o mogućim varijantama problema raspoređivanja dobija kombinovanjem: "svaki" od slučajeva grafa zadataka raspoređuje se na "svaku" od mogućih arhitektura višeprocesorskog sistema (neke kombinacije su, naravno, besmislene, kao što je raspoređivanje nezavisnih zadataka na neke specijalizovane arhitekture). Detaljan opis mnogih varijanti može se naći u [12, 15]. Pri tome je važno napomenuti da su na slikama 3.3 i 3.4 zanemareni slučajevi raspoređivanja na jednoprocesorske sisteme (Single Machine Scheduling) jer je to posebna oblast istraživanja koja se ne razmatra u ovom radu. Međutim, i taj slučaj je od praktičnog značaja i veoma je zastupljen u literaturi kao što se može zaključiti iz [12, 15, 77].

Najjednostavniji slučaj je kada ne postoje zavisnosti među zadacima koji se raspoređuju. To je raspoređivanje nezavisnih zadataka (poslova), tzv. job–shop raspoređivanje kod kojega se formiraju redovi čekanja na procesore za koje tada nije važno kako, pa čak ni da li su povezani. Međutim i tu se razlikuju podslučajevi: raspoređivanje na dva, više, pa čak i neograničeni broj procesora. Ova varijanta problema sa lako prevodi na problem pakovanja, odsecanja ili grupisanja (Packing, Cutting, Clustering). Među radovima koji se bave raspoređivanjem nezavisnih zadataka mogu se izdvojiti [11, 17, 20, 81, 82, 122, 128]. Već na ovom malom broju radova vidi se koliko raznih varijani ovog specijalnog slučaja problema (i na koliko različitih načina) se u literaturi razmatra. Model i analiza nekih algoritama dati su u [108], a noviji pregled u [53].

Zatim dolazi raspoređivanje fino granulisanih zadataka, tj. zadataka jedinične dužine izvršavanja (Unit Execution Time, UET). Literatura u vezi sa ovim slučajem brojna je, na primer [16, 83, 113, 135, 140]. Pregledom ovih radova vidi se da se i tu mogu izdvojiti podslučajevi: bez komunikacija/sa komunikacijama, način povezivanja (zavisnost) među zadacima (drveta, višenivoovski grafovi, specijalizovani grafovi kao na primer, sekvencijalno-paralelni, romboidni, gausovski, FFT i drugi), struktura višeprocesorskog sistema (2 procesora, potpuno povezani, specijalne arhitekture kao nizovi procesora, prstenovi, hiperkocke, zvezde ili proizvoljne strukture). Raspoređivanje fino granulisanih zadataka, pri čemu je u obzir uzimano i vreme komunikacije (Unit Execution Time/Unit Communication Time, UEC/UET), na višeprocesorski sistem baziran na transpjuterima, razmatrano je u magistarskom radu autora [111]. U tom smislu ovaj rad je logičan nastavak i proširenje ranijih istraživanja, kako u odnosu na problem koji se razmatra, tako i na metode koje se primenjuju.

Iako jedan od jednostavnijih slučajeva problema paralelizacije, raspoređivanje fino granulisanih zadataka ima veliki značaj i primenu na primer, u projektovanju i implementaciji prevodilaca za programske jezike. Čak i u savremenim jednoprocesorskim računarima, postoje paralelne aritmetičko-logičke jedinice, pa je važno da se one što bolje iskoriste generisanjem efikasnih, paralelizovanih mašinskih kodova pri prevođenju programa sa nekog višeg programskog jezika.

Kao treća grupa problema može se izdvojiti raspoređivanje zadataka proizvoljne dužine izvršavanja, pri čemu se opet razlikuju podslučajevi definisani komunikacijama, vezama među zadacima i tipom višeprocesorske arhitekture na koju se vrši raspoređivanje. Sem toga problem zavisi i od kriteri-

juma koji se koriste za utvrđivanje kvaliteta rešenja: uobičajeno je da se koristi minimizacija ukupnog vremena izvršavanja, ali se mogu pojaviti i drugi (maksimalni protok zadataka, minimizacija vremena koje se troši na komunikaciju, ravnomerno opterećenje procesora i sl.). Dakle, u definiciju problema raspoređivanja ulaze tri "parametra": struktura grafa zavisnosti među zadacima, struktura višeprocesorskog sistema i kriterijum optimalnosti. Shodno tome, predložena je klasifikacija [63, 136], koja je korišćena i detaljno opisana u [15]. Svaka varijanta problema označena je trojkom $\alpha|\beta|\gamma$ gde α predstavlja karakterizaciju višeprocesorskog okruženja, β se odnosi na opis poslova koje treba raspoređivati, dok γ definiše kriterijum optimalnosti. Vrednost za α sastoji se od dva polja α_1 i α_2 od kojih prvo definiše tip sistema (homogeni ili heterogeni, opšte namene ili specijalizovani), a drugo broj procesora. Karakteristike modula (poslova, zadataka) opisuju se pomoću 6 elemenata β_1, β_2, ..., β_6 od kojih neki mogu biti i prazni. Prvi opisuje da li je dozvoljeno prekidanje zadataka ili ne, drugi definiše relaciju prethođenja među zadacima, treći se koristi ukoliko je reč o dinamičkom raspoređivanju sa vremenskim ograničenjem. Elemenat β_4 opisuje dužine izvršavanja zadataka i, eventualno, neke dodatne zahteve (na primer, više procesora po zadatku), dok β_5 definiše da li postoji vremensko ograničenje za završetak svakog zadatka. Poslednji element govori da li je možda dozvoljeno kombinovanje izvršavanja kompletnog grafa zadataka sa paketnom (batch) obradom pojedinih grupa modula.

Prema toj klasifikaciji MSPCD može se označiti sa $P^*|prec|C_{max}$ gde P^* označava višeprocesorski sistem koji se sastoji iz p identičnih procesora povezanih na proizvoljan način (ova arhitektura inače nije razmatrana u navedenim referencama, diskusija je ograničena na potpuno povezane višeprocesorske sisteme). Drugim rečima, problem statičkog raspoređivanja se može objasniti na sledeći način: graf zadataka koji se sastoji iz n zadataka proizvoljne dužine, sa proizvoljnom relacijom prethođenja ($prec$) između zadataka i proizvoljnom količinom podataka koja se prenosi između zavisnih zadataka raspoređuje se na višeprocesorski sistem (P^*) koji se sastoji od p identičnih procesora povezanih na proizvoljan način (prsten, zvezda, hiperkocka, mreža, i sl.) tako da se minimizira ukupno vreme potrebno za izvršavanje svih zadataka u grafu (C_{max}). U ovoj formi problem se razmatra u radovima [35, 51, 86, 87, 88, 93, 95, 123, 125].

Neke specijalne slučajeve problema raspoređivanja moguće je rešiti optimalno u polinomnom vremenu. Na primer, u radu [57] predložen je algoritam polinomne složenosti ($O(n^2)$) za optimalno raspoređivanje proizvoljno

povezanih zadataka podjednake dužine na dva procesora. Autori rada [135] razmatrali su problem raspoređivanja drveta, degenerisanih grafova, na neograničeni broj procesora. U radu [85] pokazuje se da za višenivoovske grafove (u kojima veze postoje samo među zadacima na susednim nivoima) postoji algoritam za dobijanje optimalne raspodele u polinomnom vremenu ukoliko je zadovoljen uslov da je maksimalna zahtevana komunikacija u celom grafu manja od minimalnog vremena izvršavanja zadatka, tj. od dužine izvršavanja najkraćeg zadatka u grafu. Ovo su samo neki od mnogobrojnih radova koji razmatraju specijalne slučajeve problema raspoređivanja za koje se može naći optimalno rešenje u polinomnom vremenu. Detaljnije o toj problematici, kao i o raspoređivanju uopšte, može se naći u [15], ili u preglednom članku [12], a spisak relevantne literature dat je u [77].

Najviše radova u literaturi bavi se uvođenjem raznih heuristika za dobijanje zadovoljavajućeg suboptimalnog rešenja. Dakle, polazi se od pretpostavke da je graf zadataka unapred zadat i vrši se pridruživanje zadataka procesorima primenom nekog heurističkog rezonovanja. Na primer, prvo se raspoređuju zadaci koji imaju više sledbenika, jer će se njihovim završavanjem stvoriti uslovi za izvršavanje većeg broja zadataka što ranije. Sledeći primer bio bi forsiranje zadataka koji se duže izvršavaju jer oni zahtevaju najviše vremena. Suprotno bi bilo forsiranje što kraćih zadataka, jer se tada na početku izvršavanja odradi najviše. Zatim, metoda najranijeg starta forsira startovanje izvršavanja zadatka što pre, jer to znači i njegovo ranije završavanje i stvaranje uslova za raniji početak izvršavanja njegovih sledbenika. Heurističke metode raspoređivanja najčešće se sastoje iz dva koraka: 1) određivanje totalnog poretka među zadacima, tj. utvrđivanje redosleda kojim će se zadaci raspoređivati i 2) dodeljivanje zadataka redom procesorima na izvršavanje. To nije opšte pravilo, na primer, PPS metoda [93] u prvom koraku određuje koji će se zadatak izvršavati na kom procesoru, a u drugom kada će početi njegovo izvršavanje.

Do sada je razvijeno mnogo heurističkih algoritama, međutim, svaki od tih algoritama je efikasan samo za određenu klasu problema. Na efikasnost algoritma utiče pre svega struktura grafa zadataka, zatim struktura višeprocesorskog sistema. Pokušaj klasifikacije konstruktivnih heuristika urađen je u [34, 67], a ovde su samo, pregleda radi, navedene neke od najpoznatijih heurističkih metoda predloženih u literaturi. Neke od njih implementirane su u toku istraživanja opisanih u ovom radu, upoređivane i korišćene za realizaciju metaheurističkih metoda sa ciljem da se poprave performanse rešenja koja ove metode daju. Diskusija korišćenih metoda data je u odeljku 4.9.

4. Metaheuristike za raspoređivanje

Na početku ove glave naveden je pregled postojećih rezultata primene izabranih metaheuristika na problem raspoređivanja. Zatim su opisani detalji implementacije metaheuristika (GA, MLS, TS, VNS) koje se primenjuju za rešavanje problema statičke paralelizacije. Implementacija je bila neophodna iz više razloga. Prvo, ne postoji komercijalni softver kojim su obuhvaćene metaheurističke metode. Svi autori uglavnom sami razvijaju programe prilikom implementacije. Samim tim, ti programi su razvijeni korišćenjem različitih programskih jezika, za različite platforme računara (kako hardverske tako i softverske komponente, tj. operativne sisteme). Osim toga, kao što se vidi iz pregleda literature datog u odeljku 4.1. korišćene su različite reprezentacije rešenja. Konačno, svaki od tih programa bio je namenjen rešavanju neke varijante problema koje se često razlikuju međusobno, a svakako su različite od varijante koja se u ovom radu razmatra. Stoga je za potrebe ovog rada razvijen novi softver u okviru kojega je implementirano više metaheurističkih metoda u odnosu na isto okruženje. Sve tri metode implementirane su u odnosu na istu reprezentaciju rešenja za MSPCD kako bi se olakšalo njihovo poređenje. Izbor reprezentacije baziran je na ranijim iskustvima i detaljno je obrazložen u odeljku 4.2.

U devetom odeljku navedeni su rezultati raspoređivanja slučajno generisanih grafova zadataka. Vrednosti parametara izabrane su na osnovu ranijih iskustava. U desetom odeljku opisana je detaljna parametarska analiza vezana kako za strukturu podataka korišćenih za reprezentaciju rešenja, tako i za vrednosti parametara metoda. Obzirom na to da su GA i TS već detaljno ispitane, a VNS je nova metoda i po prvi put primenjena je na problem statičkog raspoređivanja, naglasak parametarske analize je na metodi promenljivih okolina. Analizirane su i definicije okolina u kojima se vrši pretraživanje i kriterijumi zaustavljanja. Implementirani hibridi metoda opisani su u poslednjem odeljku.

4.1. Dosadašnji rezultati

U ovom odeljku dat je pregled radova u kojima su opisane metaheuristike korišćene u rešavanju nekih varijanti problema raspoređivanja.

Na problem raspoređivanja u uslovima ograničenja resursa (Resource-Constrained Scheduling) VNS metoda primenjena je u [56]. Ovaj problem nije u direktnoj vezi sa problemom statičkog raspoređivanja na identične procesore koji se u ovom radu razmatra, ali se neki detalji implementacije mogu primeniti u oba slučaja. Rešenja su predstavljena pomoću permutacija zadataka, pri čemu je potrebno voditi računa o dopustivosti, tj. poštovati relaciju zavisnosti među zadacima. Prednost ovakve reprezentacije je jednostavno manipulisanje, dok je nedostatak što ne postoji $1-1$ korespondencija između rešenja i finalne raspodele. Naime, ista raspodela se može dobiti počev od različitih permutacija zadataka.

LS procedura u [56] definisana je premeštanjem zadatka sa jednog mesta u permutaciji na sva ostala koja čuvaju dopustivost. Dozvoljeni region za premeštanje nekog zadatka je skup svih pozicija između njegovog poslednjeg prethodnika i prvog sledbenika. To znači da se zadatak može pomerati ulevo sve dok se ne nađe neposredno iza poslednjeg od svojih prethodnika i, analogno, udesno sve do pozicije neposredno ispred prvog od svojih sledbenika. Međutim, autori rada [56] su predložili da se taj region proširi tako što će se pri nailasku na nekog prethodnika nastaviti premeštanje, ali sada ne samo datog zadatka nego i njegovog prethodnika. Kad se pojavi novi prethodnik, i on učestvuje u daljem premeštanju, itd. Postupak se nastavlja sve dok se ispred datog zadatka u permutaciji ne nađu samo njegovi prethodnici. Slično se i pri premeštanju udesno, u odnosu na sledbenike dati zadatak premešta sve dok iza njega ne ostanu samo njegovi sledbenici. To se može shvatiti i kao premeštanje zadataka nezavisnih od datog u suprotnom smeru od smera premeštanja datog zadatka. Početno rešenje dobija se heuristikom. Vršeni su eksperimenti sa nekoliko poznatih heuristika, a razmatran je i slučaj kada se početno rešenje dobija nakon primene LS procedure na permutaciju dobijenu heuristikom. Okolina $\mathcal{N}_k$ sadrži sve permutacije dobijene izvršavanjem k dopustivih premeštanja zadataka u okviru permutacije. Razmrdavanje (Shaking) realizovano je slučajnim izborom rešenja u okolini $\mathcal{N}_k$.

VND metoda primenjena na nekoliko varijanti problema raspoređivanja sekvence zadataka na jedan procesor predložena je u [49]. Rešenje je predstavljeno permutacijom zadataka, a LS procedura definisana u odnosu na

tri različite okoline: *transponovanje*, tj. zamena mesta paru susednih zadataka, *razmena*, koja se dobija zamenom mesta proizvoljnom paru zadataka i *umetanje*, realizovano premeštanjem zadataka sa jednog mesta na drugo u permutaciji. Veličine ovih okolina su redom $n-1$, $n(n-1)/2$ i $(n-1)^2$, te su pogodne da se, upravo tim redom, kombinuju u VND metodi. Ideja je da se prvo ostvare poboljšanja brzom pretragom manjih okolina, a da se tek potom ispituju moguće popravke tekućeg rešenja u okolini sa velikim brojem suseda. Primenjene su obe strategije prelaska u bolje rešenje i nakon prvog poboljšanja (FI) i detaljna pretraga i prelazak u najbolje rešenje u celoj okolini (BI).

Na problem raspoređivanja nezavisnih zadataka identičnim procesorima na izvršavanje, TS algoritam primenjen je u radu [80]. Početno rešenje dobijeno je slučajnim izborom zadataka i njihovim dodeljivanjem trenutno najmanje opterećenim procesorima. LS procedura definisana je promenom mesta parovima zadataka raspoređenim na različite procesore. Pri tome se pretpostavlja da na svakom procesoru postoji tzv. *prazan zadatak* (dummy task) čije vreme izvršavanja je 0, tako da se razmenom mesta nekog zadatka sa tim zadatkom omogućava jednostavno premeštanje zadatka sa procesora na procesor. Da bi se smanjila okolina za pretraživanje biraju se oni zadaci čije premeštanje najviše približava raspodelu idealnoj, tj. raspodeli kod koje je ostvarena potpuna ravnoteža opterećenja procesora pa je dužina izvršavanja SL_k na svakom procesoru k, $k = 1, \ldots, p$ jednaka ukupnoj dužini izvršavanja svih zadataka podeljenoj sa brojem procesora p $SL_k = \frac{1}{p}\sum_{i=1}^{n} l_i$. Sem toga, uvek se kandidati za premeštanje traže između zadataka na najopterećenijem procesoru i onih procesora kod kojih je opterećenje manje od idealnog. Kao dodatni kriterijum za smanjenje okoline koristi se eliminacija tzv. *nultih* premeštanja, tj. onih kod kojih je dužina oba zadatka jednaka. Očito je da takva premeštanja ne utiču na promenu vrednosti funkcije cilja, tj. konačne raspodele. U tabu listi TL pre svega, čuvaju se oni parovi procesora koji su učestvovali u prethodnim premeštanjima. Pri tome je bitan redosled procesora, tj. dozvoljeno je premeštanje *na* onaj procesor *sa* koga je ranije premeštano (ukoliko su zadovoljeni ostali uslovi). Drugi podatak koji se čuva u TL je dužina zadatka koji je nedavno premešten *sa* datog procesora. Time se sprečava da se (neki drugi) zadatak iste dužine vrati na taj procesor i tako izazove neka vrsta cikliranja. Dužina tabu liste određuje se dinamički.

Isti problem razmatran je u radu [128]. Rešenja i okoline definisani su na

isti način kao i u [80]. Obzirom na jednostavnost problema koji se rešava, primenjen je trik za smanjivanje okoline, tako da se premeštanje vrši samo sa najopterećenijeg procesora (onog čije vreme zauzetosti definiše vrednost funkcije cilja koja se minimizira) na najslobodniji procesor. Pri tome se pogodnost kandidata za premeštanje ne određuje na osnovu njegovog doprinosa revnomernosti opterećenja (kao što je to bio slučaj u [80]) nego uravnoteženju razlike opterećenja između ta dva procesora. Da bi se sprečilo cikliranje, u TL se čuvaju zadaci koji su premeštani u poslednjih $|TL|$ iteracija.

Na problem raspoređivanja zavisnih zadataka na heterogene višeprocesorske sisteme primenjena je TS metoda u [115]. Uvedene su pretpostavke da je vreme komunikacije zanemarljivo malo i da se višeprocesorski sistem sastoji od p–1 identičnih (sporih) procesora i jednog brzog. Rešenja su predstavljena listama zadataka pridruženih svakom procesoru, pri čemu se početno rešenje dobija heurističkom metodom DES+MFT [97] (koja je u stvari modifikacija CPES algoritma tako da se može primeniti na heterogene sisteme). Okolina rešenja dobija se premeštanjem zadatka sa jednog procesora na sve preostale. Očigledno je da se zadatak na novom procesoru može smestiti samo na onim pozicijama koje su iza poslednjeg prethodnika datog zadatka i ispred njegovog prvog sledbenika. Smanjenje okoline postignuto je uređenjem zadataka u polaznom grafu tako da važi $i \notin Pred(j)$ ako je $i > j$. Zadaci se tada u okviru liste za jedan procesor uređuju u rastućem redosledu indeksa. To znači da u svakoj listi postoji tačno jedno mesto na koje se može ubaciti novi zadatak. U TL se čuvaju parovi (i, k) takvi da je premeštanjem zadatka i sa procesora k pokvarena vrednost funkcije cilja. To znači da se u narednih $|TL|$ koraka (iteracija) zabranjuje vraćanje zadatka i na procesor k kako bi se izbeglo cikliranje.

Nad istom reprezentacijom rešenja implementiran je i GA predložen u [78] i namenjen je raspoređivanju zavisnih zadataka, kod kojih je vreme komunikacije zanemarljivo, na potpuno povezanu višeprocesorsku arhitekturu. Izvršavanje GA bazirano je na definiciji *visine, nivoa* zadatka tj. maksimalnom broju zadataka na putu od tog zadatka do nekog zadatka bez prethodnika. Očigledno je da su zadaci na istoj visini međusobno nezavisni. Inicijalna populacija generiše se na slučajan način: za svaku jedinku se redom po nivoima izvrši slučajna particija zadataka na istoj visini i te particije se raspodele između procesora. Funkcija prilagođenosti zadata je funkcijom cilja, tj. dužinom raspodele definisane odgovarajućim listama zadataka po procesorima.

Prilikom definicije genetskih operatora ukrštanja i mutacije mora se voditi računa o tome da rezultujuće jedinke predstavljaju dopustiva rešenja, tj. da se poštuje relacija zavisnosti među zadacima. To je postignuto sledećim definicijama pomenutih operatora. Ukrštanje je jednopoziciono. Pozicija se bira na slučajan način i vezana je za visine zadataka: da bi ukrštanje proizvelo dopustive jedinke potrebno je da na izabranoj poziciji budu zadaci sa istom visinom. U tom slučaju se razmenom ostataka lista kojima se definiše jedinka (rešenje) neće narušiti relacija prethođenja među zadacima. Definicija mutacije takođe je bazirana na visini zadataka: slučajno izabranom paru zadataka na istoj visini razmenjuju se mesta u tekućoj raspodeli. Selekcija jedinki za novu generaciju vrši se pomoću težinskih ruleta. Svakoj jedinki se dodeljuje prostor (isečak) na ruletu, proporcionalan vrednosti funkcije prilagođenosti. Na slučajan način se bira broj koji predstavlja indeks isečka pridruženog odgovarajućoj jedinki. Težinski rulet obezbeđuje da bolje prilagođene jedinke imaju veću verovatnoću selekcije jer im odgovara veći isečak na ruletu. Izuzetak je napravljen za trenutno najbolju jedinku koja se bezuslovno prenosi u sledeći generaciju.

PSGA, Problem–Spaced GA predložen u [4] koristi populaciju jedinki koje predstavljaju liste prioriteta zadataka. Svaka jedinka je niz od n realnih brojeva koji predstavljaju statički prioritet svakog zadatka. Ovakvom reprezentacijom rešenja postiže se veća efikasnost GA obzirom da ne postoji mogućnost generisanja nedopustivih jedinki. Za dobijanje vrednosti funkcije cilja koristi se neka heuristika bazirana na listama prioriteta zadataka. Inicijalna populacija generiše se na sledeći način: odredi se jedinka definisana primenom CP metoda na izračunavanje prioriteta zadatka. Ostale jedinke dobijaju se slučajnim perturbacijama elemenata prve, CP-bazirane jedinke. Ukrštanje je jednopoziciono, mutacija je ostvarena slučajnom perturbacijom prioriteta nekog, slučajno izabranog zadatka. Funkcija prilagođenosti definisana je slično kao u prethodnom radu, s tim što se koristi njena normalizovana vrednost (vrednost je svedena na interval (0,1)). Nedostatak ovog pristupa je indirektna reprezentacija: lista prioriteta ne određuje jedinstveno redosled zadataka za raspoređivanje, tako da se zahteva dodatno pretraživanje i pravilo za definisanje jednoznačnog izbora zadataka kojima je dodeljen isti prioritet. Osim toga, indirektnost se ogleda i u tome što je potrebno izvršiti raspoređivanje kako bi se dobila vrednost funkcije cilja.

U nastavku ove glave predložena je implementacija četiri metaheurističke metode, koje su ovde detaljno opisane (GA, MLS, TS i VNS). Implementacija svih metoda bazirana je na istoj reprezentaciji rešenja. Korišćene su ideje

predložene u nekim od radova opisanim u ovom odeljku. Na osnovu rezultata poznatih iz literature, usvojena je reprezentacija koja se pokazala najefikasnijom za svaku od metoda. Time je omogućeno da se sve metode izvršavaju pod istim uslovima i jednostavno porede.

4.2. Strukture podataka

Da bi se implementirala neka metaheuristička metoda, potrebno je definisati nekoliko pojmova i kriterijuma. Najpre se definiše prostor rešenja i prostor dopustivih rešenja, bira se način predstavljanja rešenja u memoriji računara efikasan za implementaciju i način ažuriranja vrednosti funkcije cilja. Zatim se razmatraju moguće transformacije rešenja i biraju okoline, parametri metoda, konkretne definicije pojedinih koraka (operatora) svake od metoda i kriterijumi zaustavljanja.

Prostor rešenja i prostor dopustivih rešenja

Za problem raspoređivanja zadataka na višeprocesorske sisteme rešenja mogu biti permutacije n zadataka ili uređene p-torke listi zadataka pri čemu su elementi jedne liste zadaci raspoređeni za izvršavanje istom procesoru, ili, ukoliko se rešava linearna formulacija problema, skup svih 0,1-nizova dužine n^6. Dopustiva rešenja su dopustive permutacije (one koje zadovoljavaju relaciju prethođenja – zavisnosti među zadacima) ili p-torka listi takvih da je ukupni broj elemenata u svim listama n, da se svaki zadatak pojavljuje tačno jednom u tačno jednoj od lista i da redosled zadataka u okviru jedne liste zadovoljava relaciju prethođenja među zadacima, ili 0,1-nizovi koji zadovoljavaju ograničenja 3.2-3.12.

U implementaciji opisanoj u ovom rukopisu za prostor rešenja S izabran je skup svih permutacija zadataka. Naime, zadaci su numerisani tako da se prati uređenje definisano relacijom prethođenja (zavisnosti) koja važi u okviru grafa zadataka, tj. da važi $i \in Pred(j) \Rightarrow i < j$. Tada se skup od n zadataka jednog paralelnog programa može predstaviti permutacijom koju čine indeksi tih zadataka. Skup dopustivih rešenja $X \subseteq S$ čini skup svih dopustivih permutacija tj. onih koje čuvaju relaciju zavisnosti među zadacima. To znači da se u dopustivoj permutaciji ne može desiti da se neki zadatak pojavi pre bilo kog od svojih prethodnika ili nakon bilo kog od svojih sledbenika.

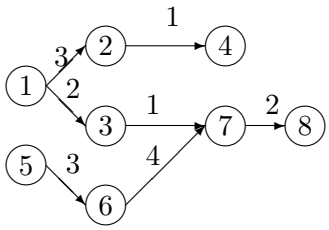
Ovakva reprezentacija rešenja predložena je u [35] imajući u vidu da se konstruktivne heuristike sastoje od dva koraka: prvi je uređenje zadataka u listu kojom se definiše redosled raspoređivanja, a drugi je izbor procesora i određivanje početnog trenutka za svaki od zadataka po redosledu definisanom listom prioriteta. To je najčešća šema po kojoj se izvršavaju heurističke metode, mada nije jedina (kao što se vidi iz pregleda datog u glavi 3.). Problem koji se javlja prilikom formiranja listi prioriteta je što definicije prioriteta uglavnom nisu jednoznačne (dva ili više zadataka mogu imati isti prioritet). Na primer, ako se prioritet definiše kao dužina kritičnog puta (CP), mogu se pojaviti dva ili više zadataka koji imaju istu vrednost tog parametra i onda se zahteva dodatni kriterijum (redni broj zadatka, broj sledbenika, nivo zadatka i sl.) kojim se vrši rangiranje zadataka sa istim prioritetom. To praktično znači da treba proširiti relaciju zavisnosti među zadacima od relacije parcijalnog poretka do relacije totalnog poretka. Predstavljanjem rešenja pomoću dopustive permutacije zadataka ovaj problem se izbegava.

Kada se zada dopustiva permutacija zadataka, jednoznačno je određen redosled kojim će se zadaci raspoređivati procesorima na izvršavanje. Zatim se primenom neke heuristike za svaki od zadataka lako određuje procesor na kome će se taj zadatak izvršavati kao i vremenski interval u kome će se taj zadatak izvršavati, tj. trenutak u kome će početi njegovo izvršavanje. Na primer, ako se usvoji heuristika najranijeg starta (ES), za svaki od zadataka se izračunava trenutak u kome on može početi svoje izvršavanje na svakom od procesora i dodeljuje se onom procesoru kod koga je taj trenutak najraniji mogući [4, 35]. Time se pokazuje da je rešenje problema raspoređivanja moguće predstaviti pomoću dopustive permutacije zadataka. U literaturi se mogu sresti i druge reprezentacije i najrasprostranjenija je pomoću lista zadataka dodeljenih procesorima na izvršavanje [78, 115], ali se u ovom slučaju reprezentacija pomoću permutacija pokazala pogodnijom za implementaciju. Prednost reprezentacije preko dopustivih permutacija u okviru GA pokazali su autori rada [88]. Ovde se na malim primerima analiziraju prednosti i nedostaci ove reprezentacije u odnosu na metaheuristike bazirane na lokalnom pretraživanju.

Osnovne prednosti reprezentacije pomoću dopustivih permutacija su što su jednostavne za manipulaciju, što se mogu kombinovati sa raznim heuristikama za raspoređivanje i što se nad njima može definisati veliki broj okolina na čijem pretraživanju se zasniva većina metaheurističkih metoda.

Međutim, reprezentacija rešenja pomoću dopustivih permutacija zadataka ima i svojih nedostataka. Jedan od nedostataka ove reprezentacije je nejed-

noznačnost – više permutacija može odgovarati istoj raspodeli [35].



i	1	2	3	4	5	6	7	8
L_i	5	6	2	3	6	4	4	2

Sl. 4.1: Primer grafa zadataka iz disertacije [124]

Na primer, graf sa slike 4.1 ima 8 zadataka među kojima zavisnost nije intenzivna. Stoga se od njih može generisati veliki broj dopustivih permutacija (preciznije, od 8! mogućih permutacija, dopustive su 103). Heuristička raspodela prikazana na slici 4.2 dobija se ako se ES heuristika za raspoređivanje primeni na bilo koju od sledećih permutacija:

1 2 3 4 5 6 7 8
1 2 3 5 4 6 7 8
1 2 3 5 6 4 7 8
1 2 3 5 6 7 4 8
1 2 3 5 6 7 8 4
itd.

P_1: 1_5, 2_{11}, 4_{14}
P_2: 5_6, $_7 3_9$, 6_{13}, 7_{17}, 8_{19}

Sl. 4.2: Heuristička raspodela grafa sa Sl. 4.1

Na slici 4.2 indeks ispred zadatka označava vreme početka njegovog izvršavanja, dok indeks koji se navodi iza zadatka označava vreme njegovog završetka. Ukoliko prednji indeks nije naveden podrazumeva se da je ili 1 (početak izvršavanja) ili da je jednak vremenu završetka prethodnog zadatka na istom procesoru.

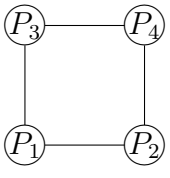
Nejednoznačnost znači da se prilikom pretraživanja (bilo da se koristi GA bilo metaheuristike zasnovane na lokalnom pretraživanju) izvršava određeni broj neproduktivnih transformacija. Međutim, ovaj nedostatak se lako može prevazići pogodnim izborom podskupa dopustivih permutacija koji predstavlja okolinu za pretraživanje kao što se može videti iz odeljka 4.10.

Drugi nedostatak je simetrija rešenja, problem koji izaziva indeksiranje procesora. Lako se vidi da su rešenja prikazana na slikama 4.2 i 4.3 identična. Time se ustvari povećava skup permutacija koje generišu isto rešenje, te je stoga očito da se i ovaj problem može prevazići pogodnim smanjenjem skupa dopustivih permutacija koje se razmatraju prilikom izvršavanja metaheurističkih metoda.

P_1: 5_6, ${}_7 3_9$, 6_{13}, 7_{17}, 8_{19}
P_2: 1_5, 2_{11}, 4_{14}

Sl. 4.3: Raspodela simetrična u odnosu na Sl. 4.2

Problem simetrije ne javlja se u svim slučajevima, on postoji samo kod potpuno povezanih višeprocesorskih arhitektura i kod nekih specijalnih slučajeva kao što je zvezdasta (master–slave) struktura procesora. Na primer, već kod hiperkocke predstavljene na slici 4.4 simetrija ne važi u potpunosti. Može se desiti da su grupe zadataka po procesorima iste i čak sa istim rasporedom zadataka, ali da je došlo do permutovanja rednih brojeva procesora tako da su neka rastojanja među njima promenjena. To za posledicu ima promenu (smanjenje ili povećanje) dužine zahtevane komunikacije, koja zatim utiče na početke izvršavanja nekih zadataka, pa samim tim i na dužinu konačne raspodele.



$$D = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

Sl. 4.4: 2-dimenziona hiperkocka procesora

U radu [35] po prvi put je uočena osobina rešenja prikazanih pomoću do-

pustivih permutacija da se mogu lako kombinovati sa raznim heuristikama za raspoređivanje. Predloženo je da se izlistaju sve dopustive permutacije u leksikografskom poretku, da se izvrši raspoređivanje svake od njih primenom neke heuristike raspoređivanja i da se zatim usvoji najbolja dobijena raspodela. Izvršeno je upoređivanje dve heuristike za raspoređivanje: metode najranijeg starta (Earliest Start, ES) i pregrupisavanje (DeClustering, DC). Jedan od zaključaka rezultata prikazanih u radu [35] je da se primenom ES metode može dobiti raspodela čije izvršavanje na datom višeprocesorskom sistemu može trajati duže nego sekvencijalno izvršavanje (izvršavanje na jednom procesoru) istog skupa zadataka. Ova nepovoljna činjenica posledica je komunikacionog kašnjenja i potrebno je da se o tome vodi računa prilikom raspoređivanja. Primenom DC heuristike ovaj problem može se prevazići, ali je složenost te metode veća.

Drugi važan zaključak izveden u [35] je da je broj dopustivih permutacija nekog grafa zadataka teško predvideti (ali je potpuno jasno da on raste sa povećanjem broja zadataka u grafu), te je stoga nemoguće vršiti raspoređivanje svih dopustivih permutacija za svaki od polaznih grafova zadataka. Rezultati prikazani u radu pokazali su da je za raspoređivanje pojedinih grafova koji sadrže 100 zadataka potrebno i po više od $24h$ rada računara.

Stoga je bilo prirodno da se umesto detaljnog ispitivanja svih dopustivih permutacija pristupi nekom heurističkom izboru podskupa permutacija među kojima će se tražiti "dobra" raspodela polaznog grafa zadataka. To je prirodno dovelo do primene metaheurističkih metoda.

Predstavljanje rešenja

Dopustive permutacije čuvaju se u memoriji računara kao dvostruko povezane liste (doubly linked lists) jer se na taj način svaki od suseda (u bilo kojoj okolini) dobija u konstantnom broju koraka, tj. složenost transformacije kojom se nekoj permutaciji generiše susedna je $O(1)$.

Preciznije, svaka dopustiva permutacija čuva se pomoću dva niza `pred` i `next`. Niz `next` za svaki zadatak i sadrži zadatak j koji se u dopustivoj permutaciji nalazi iza (desno od) njega, tj. važi `next[i]=j`. Slično, niz `pred` sadrži zadatak ispred (levo od) datog, tj. `pred[j]=i`. Ažuriranje permutacije koja se dobija transformisanjem polazne u nekog njenog suseda dobija se manipulacijom nad elementima ova dva niza.

Na primer, neka je transformacija koja definiše okolinu premeštanje za-

datka sa jednog mesta na drugo (u regionu dopustivosti). Neka je, na primer, potrebno zadatak i umetnuti ispred zadatka j. Tada je neophodno izvršiti sledeće operacije nad elementima nizova `pred` i `next`.

```
pred[next[i]] = pred[i]; /* zadatak i se izbacije sa */
next[pred[i]] = next[i]; /* prethodne pozicije */
next[pred[j]] = i;         /* umetanje i ispred j */
next[i] = j;
pred[i] = pred[j];
pred[j] = i;
```

Slično se dobija i ukoliko se umetanje vrši iza zadatka j kao i za ostale okoline koje se koriste u ovom radu, a koje će biti opisane u odeljku 4.3.

Ažuriranje vrednosti za dužinu raspodele

Funkcija cilja koja se minimizira prilikom rešavanja problema raspoređivanja je dužina raspodele SL definisana dopustivom permutacijom (videti odeljak 3.2.).

Već je napomenuto da samo poznavanje dopustive permutacije nije dovoljno za ocenu kvaliteta rešenja koje ona predstavlja. Da bi se odredila vrednost funkcije cilja koju to rešenje (data dopustiva permutacija) predstavlja, potrebno je izvršiti raspodelu zadataka po procesorima primenom neke heurističke metode. Za razliku od male složenosti transformacije kojom se od jedne permutacije dobija njoj susedna, složenost heurističkih metoda raspoređivanja nije zanemarljiva. Uobičajena složenost metoda koje se mogu pronaći u literaturi (kao što se vidi iz preglednih članaka [42, 89]) je $O(n^2p)$. Da bi se smanjilo vreme potrebno za izračunavanje dužine raspodele definisane datom dopustivom permutacijom, uočeno je da nije neophodno vršiti preraspodelu svih zadataka već samo onih koji su promenili mesta u permutaciji. Preciznije, preraspodela se vrši počev od prve promene do kraja permutacije. Tako na primer, ako se počev od permutacije 1-3-2-5-7-4-6-9-8-10, odredi sused 1-3-2-5-7-4-6-8-9-10 koji je dobijen tako što je zadatak $i = 9$ premešten iza zadatka $j = 8$ potrebno je izvršiti preraspoređivanje samo zadataka 8, 9 i 10. Ostalim zadacima se ne menja ni procesor na kome se izvršavaju kao ni vreme početka njihovog izvršavanja.

Ovaj kriterijum ne može se primeniti u slučaju reprezentacije rešenja pomoću lista zadataka, jer kada se neki zadatak ekpslicitno premesti sa jednog procesora na drugi, njegova nova pozicija može da utiče na početak

izvršavanja i zadataka koji su bili ispred njega na procesoru sa koga je on premešten. To svakako povlači i dalje lančane zavisnosti, tako da je neophodno precizno utvrditi tačno mesto prve promene (što može biti veoma složeno) ili vršiti preraspodelu uvek od početka.

Kao ilustracija može poslužiti sledeći primer. Polazeći od raspodele prikazane na slici 4.2, neka je zadatak $i = 6$ premešten na prvi procesor i to na prvu poziciju. Rezultujuća raspodela prikazana je na slici 4.5. Kako zadatak $i = 6$ zavisi od zadatka $j = 5$ to njegovo izvršavanje ne može započeti pre nego što zadatak $j = 5$ završi svoje izvršavanje i prenesu se međurezultati prvom procesoru. S druge strane, kod reprezentacije pomoću lista, redosled izvršavanja zadataka definisan je redosledom zadataka u listi. To znači da zadatak $j = 1$ ne može započeti svoje izvršavanje dok se zadatak $i = 6$ ne završi. Odlaganje izvršenja zadatka $j = 1$ utiče na vreme početka njegovog sledbenika $j' = 3$ na drugom procesoru. Time je pokazano kako premeštanje zadatka sa procesora na procesor utiče na početak izvršavanja zadatka koji je u polaznoj raspodeli bio ispred njega i ukazuje na potrebu kompletnog preraspoređivanja svih zadataka. Naravno, i ovaj problem se može prevazići uvođenjem dodatnih kriterijuma kojima se definiše koji zadaci se mogu premeštati i na koje pozicije. Očito je da svaka reprezentacija ima svoje prednosti i nedostatke o kojima se mora voditi računa prilikom implementacije metaheurističkih metoda.

P_1: $\quad {}_9 6_{13},\ 1_{18},\ 2_{24},\ 4_{27}$
P_2: $5_6,\quad {}_{20} 3_{22},\ 7_{26},\ 8_{28}$

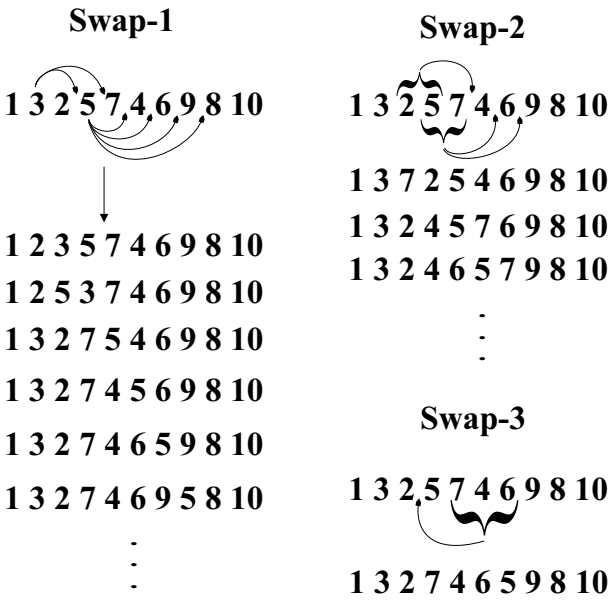
Sl. 4.5: Ilustracija nedostatka reprezentacije pomoću lista

U nastavku ove glave najpre je opisana implementacija metaheurističkih metoda bazirana na reprezentaciji pomoću dopustivih permutacija. Zatim su navedeni rezultati raspoređivanja slučajno generisanih grafova zadataka koji su poslužili kao test primeri za upoređivanje različitih metaheuristika. Za poređenje metaheuristika korišćen je jedinstven kriterijum zaustavljanja i to dopustivo vreme izvršavanja. Na kraju je opisana analiza dobijenih rezultata i date smernice za poboljšanje originalnih implementacija nekih od metoda.

4.3. Definicije okolina

U ovom odeljku opisana je implementacija metaheurističkih metoda baziranih na lokalnom pretraživanju (VNS, TS, MLS). Osnovni pojam koji je potrebno definisati kod upotrebe lokalnog pretraživanja je okolina, tj. podskup dopustivih permutacija koje će se razmatrati u svakoj iteraciji lokalnog pretraživanja.

Predstavljanjem rešenja problema raspoređivanja pomoću dopustivih permutacija moguće je koristiti nekoliko vrsta okolina poznatih u literaturi za rešavanje problema trgovačkog putnika [27, 32, 109]. Značajno je napomenuti da ne važi potpuna analogija sa problemom trgovačkog putnika, jer se kod tog problema razmatraju neorijentisani grafovi i ne postoje zavisnosti među čvorovima grafa, tj. nije bitan redosled kojim će trgovački putnik obilaziti gradove. U slučaju problema raspoređivanja, okoline se mogu definisani na sličan način, uz obavezno vođenje računa o dopustivosti.



Sl. 4.6: Primeri različitih Swap okolina

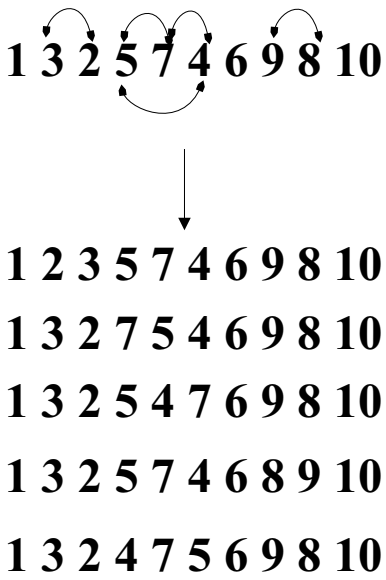
Za implementaciju lokalnog pretraživanja definisane su dve vrste okolina: premeštanje (Swap) i zamena mesta (Interchange). Swap okolina se može pojaviti u nekoliko varijanti. U ovom radu korišćene su sledeće okoline:

Swap-1 Ova okolina dobija se premeštanjem po jednog zadatka sa njegove prvobitne pozicije u permutaciji na sve "dopustive" pozicije. Dakle, dva suseda u Swap-1 okolini razlikuju se u poziciji jednog zadatka. Na primer, Swap-1 susedi permutacije 1-3-2-5-7-4-6-9-8-10 za graf zadataka predstavljen na Sl. 3.1 su 1-2-3-5-7-4-6-9-8-10, 1-2-5-3-7-4-6-9-8-10, i td. (Sl. 4.6).

Swap-2 Ako se vrši promena pozicije paru susednih zadataka dobija se Swap-2 okolina.

Swap-3 U okolini Swap-3 susedi se dobijaju tako što se svaka trojka susednih zadataka premešta na sve dopustive pozicije (Sl. 4.6).

IntCh Interchange (IntCh) okolinu čine sve dopustive permutacije koje se dobijaju zamenom pozicija paru proizvoljnih zadataka (Sl. 4.7). Zamena mesta je "dopustiva" ukoliko između dva zadatka kojima se menja mesto ne postoji ni jedan sledbenik prvog od njih kao ni prethodnik drugog zadatka.



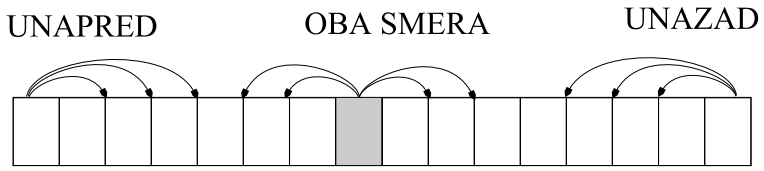
Sl. 4.7: IntCh okolina

Eksplicitan tretman dopustivosti je neophodan da bi se obezbedila korektna primena heuristike za raspoređivanje i dobila konačna raspodela koja će

se korektno izvršavati na datom višeprocesorskom sistemu. Tako, na primer permutacija 1-7-3-2-5-4-6-9-8-10 ne pripada Swap-3 okolini permutacije 1-3-2-5-7-4-6-9-8-10 (iako se od nje dobija tako što se trojka zadataka 3-2-5 premesti iza zadatka 7) jer ne predstavlja dopustivo rešenje za graf zadataka sa Sl. 3.1.

U skladu sa izlaganjem iz prethodnog poglavlja vezanim za metodu promenljivih okolina, gore navedene okoline korišćene su kako samostalno, tako i u kombinaciji. Za primenu u metodi promenljivog spusta (VND) najprirodnija kombinacija bila bi Swap-3;Swap-2;Swap-1 (skraćeno označena sa Swap-321). Redosled je određen veličinom okolina (prvo se pretražuju okoline sa manje elemenata) kao i idejom da se prvo pokuša popravka polaznog rešenja "grubim" transformacijama (koje se dobijaju većim "skokovima" u odnosu na polazno rešenje), a tek ako to ne uspe, ide se na pretraživanje okolina sa više suseda (za čije dobijanje se koriste finije transformacije). Zaista, lako se vidi da je Swap-3 okolina (kao i Intch) najmanja, jer je verovatnoća da među proizvoljna 4 izabrana zadatka postoje zavisni i da je data transformacija nedopustiva veća nego verovatnoća zavisnosti za dva proizvoljno izabrana zadatka.

Da bi se izbeglo dupliranje suseda prilikom transformisanja permutacija korišćeno je pretraživanje samo u jednu stranu. Preciznije, zadaci se unutar permutacije premeštaju samo udesno (pretraživanje unapred, FORWARD) ili samo unazad (BACKWARD), tj. počev od poslednjeg zadatka koji se premešta što je više moguće ulevo Sl. 4.8.



Sl. 4.8: Smer pretraživanja okolina

Očito je da se, ukoliko se ne razmatraju oba smera pretraživanja, neki susedi ne uzimaju u obzir, ali to ubrzava pretraživanje i može da se nadoknadi na drugi način što će biti opisano kasnije (odeljak 4.10.).

Ilustracije radi, polazeći od permutacije 1-2-3-4-5-6-7-8-9-10 za primer sa slike 3.1 susedi koji se dobijaju pretraživanjem Swap-1 okoline u pojedinim smerovima predstavljeni su na Sl. 4.9.

pretraživanje unapred	pretraživanje unazad
1-3-2-4-5-6-7-8-9-10	1-2-3-4-5-6-7-9-8-10
1-2-3-5-4-6-7-8-9-10	1-2-3-4-5-6-8-7-9-10
1-2-3-4-6-5-7-8-9-10	1-2-3-4-5-7-6-8-9-10
1-2-3-4-6-7-5-8-9-10*	1-2-3-4-7-5-6-8-9-10**
1-2-3-4-5-7-6-8-9-10	1-2-3-7-4-5-6-8-9-10**
1-2-3-4-5-6-8-7-9-10	1-2-3-4-6-5-7-8-9-10
1-2-3-4-5-6-7-9-8-10	1-2-3-5-4-6-7-8-9-10
	1-2-5-3-4-6-7-8-9-10**
	1-3-2-4-5-6-7-8-9-10

Sl. 4.9: Razlike u podokolinama dobijenim jednosmernim pretraživanjem

Zvezdicom (*) je obeležen sused koji se nalazi u podokolini dobijenoj pretraživanjem unapred dok su sa dve zvezdice (**) označeni susedi koji se ne pojavljuju u toj podokolini, tj. mogu se dobiti samo pretraživanjem unazad.

Kao što se sa slike 4.9 može videti, prilikom pretraživanja u oba smera veliki je broj suseda koji se ponavljaju. Nasuprot tome, malo njih se preskače prilikom pretraživanja samo u jednu stranu.

Sličan zaključak važi i za ostale Swap okoline. Očigledna posledica toga je da se lokalni minimumi u odnosu na dve podokoline mogu razlikovati. Što se tiče IntCh okoline, obe njene podokoline se podudaraju, te je stoga logično da se uvek pretražuje samo u jednom smeru, kako bi se izbeglo dupliranje.

U svim metaheuristikama polazi se od (najmanje jednog) početnog rešenja x. To rešenje je u ovom slučaju dopustiva permutacija zadataka i može se dobiti kao prva dopustiva permutacija u topološkom uređenju, ili permutacija dobijena primenom neke heurističke metode za uređenje zadataka (na primer, LPT ili nerastući redosled CP prioriteta zadataka) ili kao proizvoljna slučajno izabrana dopustiva permutacija.

Lokalno pretraživanje (LS) u ovom slučaju implementirano je tako što se kreće od početnog rešenja i vrši se premeštanje zadataka redom, pri čemu se vodi računa o dopustivosti rezultujuće permutacije. Parametri lokalnog pretraživanja su:
- smer: da li se pretražuje unapred ili unazad (FOR–BACK);
- stepen poboljšanja: zaustavljanje pri nailasku na prvo bolje rešenje ili de-

taljno pretraživanje u potrazi za najboljim rešenjem u celoj okolini (FI–BI);
- tip okoline: Swap ili IntCh.

Zavisno od vrednosti parametara, LS procedurom pretražuje se najbliža okolina trenutno najboljeg rešenja sve dok u takvoj okolini postoji bolje rešenje. To znači da se postupak završava čim se pronađe lokalni minimum.

Kao okolina za pretraživanje može poslužiti bilo koja od prethodno navedenih okolina (Swap-1, Swap-2, Swap-3, IntCh). Ukoliko se pojedinačna okolina zameni njihovom proizvoljnom kombinacijom, dobija se deterministička varijanta metode promenljivih okolina, tj. metoda promenljivog spusta (VND, videti odeljak 2.3.). Ova metoda se takođe zaustavlja pri nalaženju lokalnog minimuma, ali u odnosu na sve okoline koje učestvuju u pretraživanju.

4.4. Višestartno lokalno pretraživanje

Da bi se LS (odnosno VND) rezultati popravili, kao što je već navedeno u prethodnoj glavi, najjednostavniji način je da se ovako opisana LS (VND) procedura ponavlja iz različitih početnih rešenja, čime se dobija najjednostavnija metaheuristika Višestartno lokalno pretraživanje (MLS, MVND).

MLS procedura izvršava se sa istim parametrima kao i obična LS procedura, a sastoji se u tome da se LS startuje počev od različitih (slučajno generisanih) početnih rešenja, sve dok se ne zadovolji neki zadati kriterijum zaustavljanja. Uobičajeni kriterijumi zaustavljanja su t_{max} - maksimalno dozvoljeno vreme izvršavanja, n_{iter} - maksimalni broj iteracija, $nimp_{iter}$ - maksimalni broj iteracija između dve popravke trenutno najboljeg rešenja.

4.5. Metoda promenljivih okolina

VNS metoda implementirana je u skladu sa opisom datim u odeljku 2.3. i blok-dijagramom ilustrovanim na slici 2.6.

Korak koji se odnosi na lokalno pretraživanje sastoji se od osnovne LS (VND) procedure, nad nekom izabranom okolinom ili kombinacijom okolina i parametrima koji su izabrani na pogodan način.

Drugi osnovni korak VNS metode je razmrdavanje tj. operacija izlaska iz zamke lokalnog minimuma. Osnovna ideja je da se intenzivira pretraživanje u najbližoj okolini dobrog rešenja, a ukoliko to ne dovede do daljih poboljšanja da se pređe u novo rešenje koje je daleko od trenutnog lokalnog minimuma.

To novo rešenje se dobija slučajnim izborom rešenja koje pripada k-toj okolini trenutno najboljeg pronađenog rešenja.

Prilikom implementacije operacije razmrdavanja korišćene su k-Swap i k-IntCh okoline.

Razmrdavanje u k-Swap okolini podrazumeva k slučajnih koraka u okolini Swap-1 i realizovano je na sledeći način:

For $i = 1$ **to** k

1. izabrati slučajan zadatak koji će se premestiti;
2. odrediti region za premeštanje, tj. pronaći poslednjeg prethodnika i prvog sledbenika datog zadatka u tekućem rešenju (prikazanom dopustivom permutacijom);
3. izabrati novu poziciju za dati zadatak, tj. izabrati slučajno broj između dva indeksa izračunata u prethodnom koraku;
4. izvršiti premeštanje izabranog zadatka na novu poziciju određenu u koraku 3.

U slučaju k-IntCh okoline procedura je malo složenija jer nije moguće zameniti mesta proizvoljnom paru zadataka, tj. za svaki slučajno izabrani par zadataka neophodno je proveriti da li se može izvršiti zamena mesta, a samu zamenu je potrebno izvršiti k puta. Stoga bi pseudokod procedure razmrdavanja u k-IntCh okolini imao sledeći oblik:

For $i = 1$ **to** k

1. DONE = FALSE;
2. **while not** DONE
   (a) slučajno izabrati dva zadatka u dopustivoj permutaciji koja predstavlja trenutno najbolje rešenje;
   (b) **if** zamena mesta je moguća, tj. među zadacima koji se nalaze između izabrana dva nema ni sledbenika prvog ni prethodnika drugog od njih;
      i. DONE = TRUE;
      ii. promeniti mesta izabranim zadacima;

Prilikom implementacije VNS metode potrebno je izabrati vrednosti dodatnih parametara (k_{min}, k_{max}, k_{step}, $p_{plateaux}$, videti odeljak 2.3.).

4.6. Tabu pretraživanje

Nad dopustivim permutacijama implementirano je tabu pretraživanje uz pomoć tabu liste (TL) promenljive dužine. U toj listi čuvaju se zadaci koje ne bi trebalo premeštati u narednih nekoliko koraka pretraživanja. Kada se u toku pretraživanja naiđe na lokalni minimum, u TL dodaje se zadatak čijim premeštanjem je taj minimum dobijen. Zatim se kao polazna tačka usvaja prvo sledeće rešenje (po kvalitetu) i od njega se nastavlja pretraživanje. Podatak sačuvan u TL sprečiće povratak u već posećeni lokani minimum. Maksimalna dužina tabu liste ($NTABU$) je parametar koji određuje u koliko narednih iteracija lokalnog pretraživanja neki zadatak neće učestvovati. Pri samoj implementaciji pokazalo se da se bolji rezultati dobijaju kada se ne koristi fiksirana vrednost za dužinu TL nego se ona u svakoj iteraciji određuje slučajnim izborom broja između 1 i $NTABU$. Za pretraživanje je korišćena svaka od ranije navedenih okolina, a parametri pretraživanja definisani su kao i za samostalni LS.

4.7. Genetski algoritam

Na osnovu rezultata publikovanih u [88], razvijena je GA metoda bazirana na reprezentaciji pomoću dopustivih permutacija [48]. Dopustive permutacije su članovi populacije (jedinke, hromozomi). One se kombinuju primenom genetskih operatora proizvodeći nove permutacije koje se zatim raspoređuju primenom neke od heuristika, a najbolja raspodela dobijena u svim generacijama usvaja se za konačno rešenje. Genetski operatori u opštem slučaju ne čuvaju dopustivost rezultujuće permutacije, te su stoga za potrebe ovog rada u njihove definicije ugrađeni zahtevi dopustivosti.

Ukrštanje dve dopustive permutacije x i y definisano je kao jednopoziciono. Izvršava se tako što se na slučajan način odredi m (pozicija za ukrštanje), a zatim se generišu dve nove jedinke x' i y'. Svaka od njih ima početni deo (do izabrane pozicije m) jednak sa polaznim permutacijama, tj. početni deo od x' jednak je početnom delu od x i slično za y'. Ostatak permutacije određuje se tako što se preostali zadaci navode redosledom njihovog pojavljivanja u drugoj permutaciji koja učestvuje u ukrštanju. Na primer, ukoliko se posmatra graf zadataka sa slike 3.1, ukrštanjem jedinki

1 2 3 4 5| 6 7 8 9 10,
1 3 2 4 7| 6 5 9 8 10,

počev od pozicije 5 dobijaju se dve nove jedinke

1 2 3 4 5| 7 6 9 8 10,
1 3 2 4 7| 5 6 8 9 10.

Ukrštanje se vrši sa zadatom verovatnoćom, što znači da se ukrštaju one dve izabrane jedinke za koje je vrednost slučajno izabranog indikatora (u intervalu [0,100]) manja od zadatog parametra p_{xover}.

Operator mutacije zasnovan je na ideji zamene mesta međusobno nezavisnih zadataka. U ovom radu realizovan je na sledeći način: za svaki par susednih međusobno nezavisnih zadataka bira se na slučajan način broj između 0 i 100. Ukoliko je on ispod praga verovatnoće mutacije p_m, vrši se zamena mesta tim zadacima. Tako se, na primer, od jedinke

1 2 3 4 5 6 7 8 9 10,

koja predstavlja dopustivo rešenje za raspoređivanje grafa zadataka sa slike 3.1, primenom operatora mutacije (sa zadatom verovatnoćom), mogu dobiti sledeće jedinke:

1 3 2 4 5 6 7 8 9 10,
1 2 3 5 4 6 7 8 9 10,
1 2 3 4 6 5 7 8 9 10,
1 2 3 4 5 7 6 8 9 10,
1 2 3 4 5 6 7 9 8 10.

Susednost pozicija za nezavisne zadatke nije neophodan uslov, mogu se menjati pozicije i nesusednim nezavisnim zadacima, jedino se pri tome mora voditi računa da dopustivost nije narušena u odnosu na zadatke koji se nalaze između njih. U tom slučaju rezultat mutacije mogla bi da bude i sledeća permutacija

1 2 3 4 7 6 5 8 9 10.

Obzirom da to zahteva dodatna ispitivanja zavisnosti, u ovom radu taj slučaj nije razmatran.

Navedeni operatori primenjuju se na sve jedinke tekuće populacije sa zadatim verovatnoćama. Na taj način dobija se novi skup jedinki koji zajedno sa jedinkama iz tekuće populacije konkurišu za ulazak u novu generaciju. Izbor jedinki za novu generaciju vrši se primenom operatora selekcije i u ovom radu to je klasična rulet selekcija. Svakoj jedinki se, proporcionalno kvalitetu raspodele koju ona određuje, dodeljuje deo intervala [0,100] i slučajnim izborom broja u tom intervalu određuje se koje će jedinke

preći u novu generaciju. Pri tome, mora se voditi računa da se jedinka ne duplira, tj. da se ne pojavi dva (ili više) puta u novoj generaciji. Postoji još jedan detalj o kome treba voditi računa prilikom implementacije operatora selekcije, a to je da li se trenutno najbolja jedinka obavezno prenosi u novu generaciju ili ne. Eksperimenti izvršeni u okviru ovog rada pokazali su da se bolji rezultati dobijaju ako se ne forsira prenošenje najbolje jedinke.

U svakoj generaciji određuje se najbolja raspodela koja se poredi sa trenutno najboljom iz svih prethodnih generacija, a najbolja raspodela kroz sve generacije tokom izvršavanja GA usvaja se kao konačno rešenje. Kao kriterijum zaustavljanja uzet je broj generacija, a kasnije je vreme potrebno da se izvrši određeni broj generacija korišćeno kao vremenski kriterijum zaustavljanja prilikom poređenja svih implementiranih metoda.

4.8. Hibridizacija metoda za raspoređivanje

Osim osnovnih verzija metaheurističkih metoda, implementirane su i neke varijante njihovih hibrida. Na primer, u GA je operator mutacije zamenjen LS procedurom čime je dobijena osnovna varijanta tzv. memetskog algoritma (MEMetic algorithm, MEM [99, 105]).

Kombinacija VND i TS, tj. korišćenje više okolina za lokalno pretraživanje u okviru TS metode do sada nije zabeleženo u literaturi. Neke od jednostavnijih kombinacija MVND (tj. višestartna metoda promenljivog spusta) i VNSVND (upotreba više okolina u LS koraku osnovne VNS metode) pomenute su ranije.

Prvi implementirani hibrid je VND-TS, TS metoda u kojoj je LS procedura zamenjena sa VND. U tabu listu se upisuje zadatak iza koga je premeštena trojka ili dvojka susednih zadataka, tako da je ažuriranje ove liste uniformno, tj. ne zavisi od okoline po kojoj se trenutno pretražuje. Parametri su isti kao i kod originalne TS metode: okoline se pretražuju unapred, dužina TL je promenljiva pri čemu je maksimalna dužina $NTABU = n/2$. MVND predstavlja višestartnu metodu promenljivog spusta, pri čemu se pretraživanje vrši do prve popravke (FI strategija), okoline se pretražuju unapred, a pretraživanje započinje od slučajnog rešenja svaki put kad se nađe minimum u odnosu na sve okoline. Kombinacija GA i LS označena je sa MEM i to predstavlja GA metodu kojoj je operator mutacije zamenjen lokalnim pretraživanjem do prvog poboljšanja okoline Swap-1 unapred. Ideja koja leži u pozadini ove kombinacije je da se popravi efekat operatora

mutacije time što će on uvek generisati kvalitetnije jedinke. Implementirane su dve varijante mutacije, prva koja se izvršava nad jedinkama (kompletnim permutacijama zadataka) i druga koja je kao i originalni operator vezana za gene, tj. pojedine zadatke u okviru neke permutacije. Prva varijanta dala je bolje rezultate. Parametri ove metode modifikovani su zbog složenosti operatora mutacije (tj. LS procedure), oko 30% jedinki popravlja se mutacijom, a ukupan broj jedinki smanjen je na $N_p = 90$. Kriterijum zaustavljanja za sve metode je maksimalno dozvoljeno vreme rada.

4.9. Rezultati primene metaheuristika

U ovom odeljku opisani su rezultati poređenja implementiranih metaheurističkih metoda na slučajno generisanim primerima grafova za raspoređivanje. Najpre su navedene karakteristike generisanih test primera, zatim je diskutovan izbor parametara svake metode i na kraju su opisani dobijeni rezultati.

Odeljak 4.10. posvećen je pokušajima da se poveća efikasnost metaheurističkih metoda i to uvođenjem dodatnih parametara, smanjenjem okolina za pretraživanje i hibridizacijom metoda.

Metaheurističke metode GA, VNS, TS, kao i njihove kombinacije implementirane su u programskom jeziku C na Intel Pentium IV procesoru (1GHz) pod Linux operativnim sistemom.

4.9.1. Slučajno generisani primeri grafova zadataka

Da bi se ilustrovala efikasnost implementiranih metaheurističkih metoda kao i uporedile razne metaheuristike izvršeno je raspoređivanje slučajno generisanih test primera. Uvidom u odgovarajuću literaturu, utvrđeno je da postojeći test primeri [89, 129] nisu adekvatni za varijantu problema raspoređivnja koja se razmatra u ovom radu. Stoga su generisani novi primeri grafova zadataka, opisani u radu [42] i predloženi za zvanične test primere (Benchmarks) prilikom razmatranja problema raspoređivanja zavisnih zadataka uz prisustvo komunikacija na višeprocesorske sisteme proizvoljne arhitekture (MSPCD). Test primeri su ukratko opisani u nastavku teksta, a nalaze se dostupni preko interneta na adresi `http://www.mi.sanu.ac.yu/~tanjad`. Procedure za njihovo generisanje detaljno su opisane u [42] kao i u prilogu A.

Korišćeni test primeri sadrže do $n = 500$ zadataka i raspoređivani su na višeprocesorske sisteme koji se sastoje od $p = 2, 4, 8$ procesora. Razlikuju

se dve grupe test primera, prvu čine slučajno generisani grafovi sa zadatom gustinom veza među zadacima ρ. Parametar ρ biran je u intervalu 0.2 do 0.8. Ovi grafovi raspoređivani su na različite višeprocesorske arhitekture i detaljni rezultati raspodela primenom kako heurističkih tako i metaheurističkih metoda opisani su u [42] i dostupni su preko interneta na već navedenoj adresi. Ovde su dati samo rezultati primene metaheurističkih metoda prilikom raspoređivanja na hiperkocke procesora. Naime, retki grafovi ($\rho = 0.2$) raspoređivani su na 3-dimenzionalnu hiperkocku procesora (Sl. 3.2). Grafovi sa gustinom zavisnosti među zadacima $\rho = 0.4, 0.5$ raspoređivani su na 2-dimenzionalnu hiperkocku (prsten procesora, Sl. 4.4), dok se za guste grafove ($\rho = 0.6, 0.8$) ciljna višeprocesorska arhitektura sastojala od $p = 2$ procesora. Izbor višeprocesorske arhitekture izvršen je na osnovu eksperimentalne analize opisane u [42]. Pokazalo se da kod gustih grafova zadataka uvođenje novih procesora nema smisla jer se mnogo vremena troši na razmenu podataka među zadacima pridruženim različitim procesorima.

Kod grafova zadataka iz prve grupe, parametar kojim se definiše broj zadataka, uzimao je sledeće vrednosti $n = 50, 100, 200, 300, 400, 500$. Za svako n postoji 30 grafova koji se razlikuju po dužini izvršavanja zadataka, količini komunikacija među zavisnim zadacima, gustini veza i drugim relevantnim parametrima čime je osigurano postojanje različitih primera (gusti-retki grafovi, grafovi kod kojih dominira izračunavanje nasuprot onima kod kojih preovlađuju komunikacije, sa uniformnom ili proizvoljnom distribucijom za vreme izvršavanja zadatka i/ili vreme komunikacije, i sl.).

Drugi skup test primera čine grafovi sa poznatom dužinom optimalne raspodele. Postupak generisanja ovih grafova opisan je u [42] i prilogu A. Parametar n varira od 50 do 500 sa korakom 50. Za svako n postoji deset grafova koji se razlikuju po gustini veza među procesorima. Važno je napomenuti da ova grupa test primera ima unapred definisanu i veoma specifičnu strukturu optimalnog rešenja, te stoga ne predstavlja reprezentativan uzorak test primera. Namena im je gruba procena efikasnosti (meta)heurističkih metoda kao i analiza samog procesa pretraživanja prostora dopustivih rešenja kao što je opisano u odeljku 4.10.

4.9.2. Rezultati raspoređivanja

U ovom odeljku prikazani su rezultati raspoređivanja slučajno generisanih test primera primenom metaheurističkih metoda. Poređene su metode opisane u prethodnom odeljku kao i PSGA metoda poznata iz literature [4] koja je

za potrebe ovog rada proširena na MSPCD [45]. Rezultati su prikazani u tabeli 4.1.

U prvoj koloni tabele 4.1 nalazi se broj zadataka u grafovima koji se raspoređuju, prosečna vrednost najbolje dobijene raspodele (za 30 primera iste dimenzije) data je u drugoj koloni. Preostalih sedam kolona sadrže prosečne vrednosti odstupanja dužina raspodela dobijenih (meta)heurističkim metodama od najbolje vrednosti navedene u drugoj koloni tabele. U trećoj koloni prikazani su rezultati primene CPES konstruktivne heuristike. Za svaki od testiranih primera ovo rešenje služilo je kao početno za primenu metaheurističkih metoda. Četvrta kolona sadrži prosečna odstupanja lokalnih minimuma, dok su poboljšanja dobijena primenom metaheuristika navedena u preostalim kolonama tabele 4.1.

Tabela 4.1: Rezultati raspoređivanja slučajno generisanih primera: prosečni rezultati za 30 grafova iste dimenzije

	najbolji	*% odstupanja*						
n	(u proseku)	CP	LS	VNS	TS	MLS	GA	PSGA
50	437.5	37.4	31.6	0.02	0.15	0.09	0.08	5.73
100	983.5	26.7	19.9	0.01	0.07	0.02	0.02	3.92
200	2137.6	17.1	12.2	0.01	0.08	0.02	0.03	2.56
300	3251.74	13.96	8.8	0.01	0.11	0.06	0.04	4.12
400	4454.46	12.4	7.7	0.01	0.12	0.08	0.02	5.17
500	5599.5	11.5	7.2	0.01	0.08	0.03	0.03	4.66

Tabela 4.2: CPU - vreme izvršavanja heurističkih metoda: prosečni rezultati za 30 grafova iste dimenzije

	CPU vreme	*CPU vreme (sekunde)*				
n	(max.)	VNS	TS	MLS	GA	PSGA
50	35.6	16.89	33.16	18.33	25.02	28.96
100	140.0	87.20	116.30	63.85	112.22	129.13
200	616.0	488.30	608.11	532.43	551.41	566.87
300	1400.0	924.19	1288.12	886.73	1227.33	1335.28
400	2440.0	1694.63	2360.70	1866.20	2262.52	1992.70
500	3860.0	2693.31	3032.18	2270.62	3622.98	3620.83

Rezultati prikazani u tabeli 4.1 dobijeni su nakon pažljive analize i eksperimenata izvršenih da bi se odredile vrednosti parametara metaheurističkih metoda. Utvrđene najpogodnije kombinacije parametara su sledeće.

Okolina Swap-1 koja se pretražuje unapred osnovna je za definiciju MLS metode.

Za VNS metodu u LS koraku Swap-1 okolina je pretraživana unapred (FORward). Prilikom pretraživanja primenjena je FI strategija u skladu sa rezultatima prikazanim u [69] i eksperimentima koji su izvršeni na ovim primerima. Vrednosti ostalih parametara su $k_{min} = 1$, $k_{max} = 20$, $k_{step} = 1$, $p_{plateaux} = 0.0$.

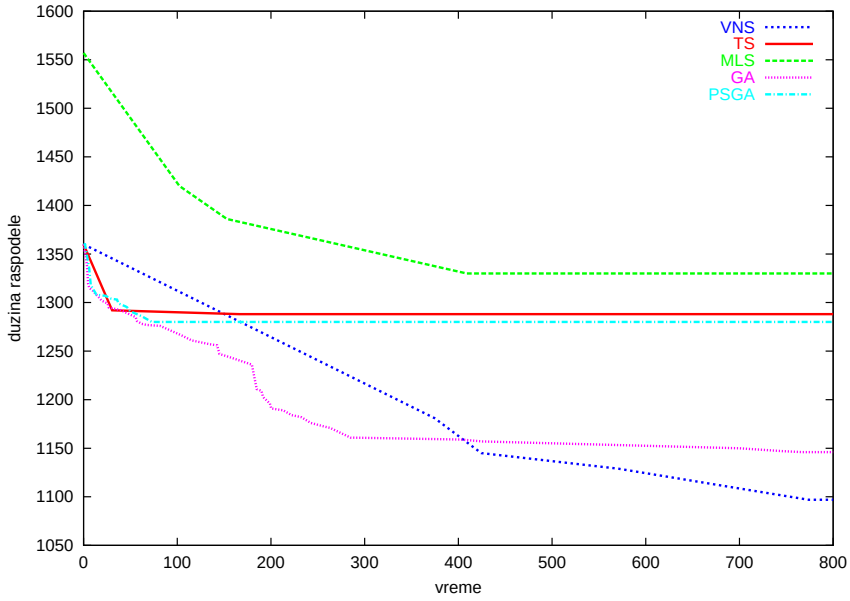
U TS metodi, okolina koja se pretražuje takođe je Swap-1, a smer pretraživanja je unapred. Maksimalna dužina tabu liste $NTABU = n/2$, a stvarna se određuje na slučajan način u svakoj iteraciji. Kod TS, korišćena je BI strategija poboljšavanja.

Primena Swap-1 okoline dala je najbolje rezultate jer se vrši veoma detaljno pretraživanje u okolini dobrih rešenja, dok pretraživanje unapred daje bolje rezultate jer je veći uticaj promene početka dopustive permutacije na novo rešenje te se poboljšanja (ukoliko ih ima) pre otkrivaju. Preskakanje okolina ili nesistematsko usvajanje rešenja koja određuju istu dužinu raspodele (definisani pomoćnim parametrima VNS metode) nisu se pokazali dobro, uglavnom su rezultovali ponašanjem koje se opisuje kao slučajno kretanje (random walk).

Vrednosti parametara GA (PSGA) metode koje su dale najbolje rezultate su $N_p = 120$, $p_{xover} = 60$ $p_m = 0.01$. Broj generacija N_g ovde je bio samo pomoćni parametar koji je poslužio za određivanje jedinstvenog kriterijuma zaustavljanja svih metaheurističkih metoda. Vreme koje je GA metodi bilo potrebno da izvrši $N_g = 600$ generacija pretraživanja usvojeno je za kriterijum zaustavljanja svih metaheurističkih metoda.

Dakle, kriterijum zaustavljanja je vremenski: t_{max} - maksimalno dozvoljeno vreme izvršavanja neke metaheurističke metode. Pretpostavlja se da je to fer kriterijum jer se sve metode izvršavaju na istom računaru, sa rešenjima predstavljenim u istom obliku (što podrazumeva isto vreme pristupa podacima) i vrše slične transformacije nad njima.

U tabeli 4.2 prikazana su prosečna vremena koja je svaka metoda potrošila da pronađe (sopstveno) najbolje rešenje. U prvoj koloni data je dimenzija problema, u drugoj prosečno (za 30 primera) masimalno dozvoljeno vreme, a u preostalih pet kolona prosečno vreme za svaku od metaheuristika, potrošeno za nalaženje rezultujuće raspodele za grafove istih dimenzija.



Sl. 4.10: Poređenje heurističkih metoda za primer iz prve grupe sa n=200

Da bi se ilustrovao proces pretraživanja, na slici 4.10 prikazano je poboljšanje tekuće raspodele kao funkcija vremena. Kao primer uzet je jedan od grafova sa $n = 200$ zadataka za koji je beleženo svako poboljšanje detektovano u intervalu $[0,\ t_{max}]$ za svaku od metoda.

Kao što se može videti iz tabele 4.1 i sa slike 4.10 (u ostalim slučajevima zaključci su vrlo slični), odstupanja konačnih rešenja različitih metoda nisu velika. Najbolja raspodela najčešće je dobijena primenom VNS metode, mada su ponekad MLS i GA generisale bolja rešenja. Odstupanja svih metoda (sem PSGA) od najboljeg pronađenog rešenja veoma su mala (ispod 1%), što govori da su metode korektno implementirane i parametri adekvatno izabrani. U svakom slučaju, najbolja moguća (optimalna) rešenja za ovu grupu test primera nisu poznata, te je stoga nemoguće govoriti o kvalitetu dobijenih heurističkih rešenja. Može se samo zaključiti da je odstupanje rešenja dobijenih svakom od heuristika relativno malo u odnosu na najbolja dobijena heuristička rešenja.

Stoga su korišćeni test primeri iz drugog skupa i metaheuristikama postavljen teži zadatak: pronaći poznato (unapred zadato) optimalno rešenje.

Od skupa koji sadrži 700 primera opisanog u [42] izabrano je $m = 100$ primera generisanih za 2-dimenzionalnu hiperkocku identičnih procesora, preciznije, uzeto je $p = 4$.

Tabela 4.3: Rezultati raspoređivanja slučajno generisanih primera sa poznatom dužinom optimalne raspodele

		% odstupanja						
n	SL_{opt}	CP	LS	VNS	TS	MLS	GA	PSGA
50	600	48.20	22.30	2.42	8.30	5.32	15.70	12.43
100	800	57.54	36.22	10.14	20.79	14.35	25.46	35.37
150	1000	74.70	38.96	11.47	24.91	19.50	44.33	42.50
200	1200	86.97	59.78	24.50	38.37	21.18	49.47	35.95
250	1400	78.48	61.93	31.05	55.10	37.03	65.69	46.53
300	1600	82.82	66.17	53.43	64.23	51.33	72.92	78.99
350	1800	82.24	61.15	35.51	55.59	36.04	63.52	72.88
400	2000	83.66	80.96	47.21	69.88	57.03	26.37	77.99
450	2200	85.36	58.85	31.76	55.60	45.59	30.14	56.69
500	2400	84.87	38.59	37.45	37.41	52.98	76.96	69.76
Prosek	1500.00	76.49	52.49	28.50	43.02	34.03	47.06	52.91

Za svako n postoji deset grafova sa različitom gustinom zavisnosti među zadacima [42]. Rezultati raspoređivanja prikazani su u tabeli 4.3. Svaka vrsta ove tabele sadrži prosečna (za 10 primera iste dimenzije) odstupanja heurističkih rešenja od poznatog optimalnog. Odgovarajuća vremena izvršavanja metoda data su u tabeli 4.4, zajedno sa maksimalno dozvoljenim vremenom (koje je i u ovom slučaju dobijeno od vremena potrebnog GA metodi da izvrši 600 generacija).

Kao i u prethodnom slučaju, izdvojen je primer grafa zadataka ($n = 200$, $\rho = 50\%$) i prikazano je poboljšanje rešenja u funkciji vremena za svaku od metoda (Sl. 4.11). Sa ove slike uočava se da metode pokazuju drugačije ponašanje (krive nisu tako glatke kao u prethodnom slučaju, popravke su retke, ali drastične). Rezultati su takođe drugačiji. Pre svega, MLS ovde daje mnogo oblje rezultate (sa slučaj sa slike 4.11 je najbolja) dok je GA značajno podbacila. Međutim, zaključak koji je od mnogo većeg značaja je da je odstupanje svih, pa i najboljeg heurističkog rešenja od poznatog optimalnog izuzetno veliko (28.5% u proseku, pa i preko 50% za $n = 300$). Ovaj zaključak predmet je detaljne analize koja je izvršena i opisana u sledećem odeljku.

Na kraju ovog odeljka data je napomena u vezi PSGA metode [4]. Na

Tabela 4.4: CPU vreme za raspoređivanje primera sa poznatim optimalnim rešenjem

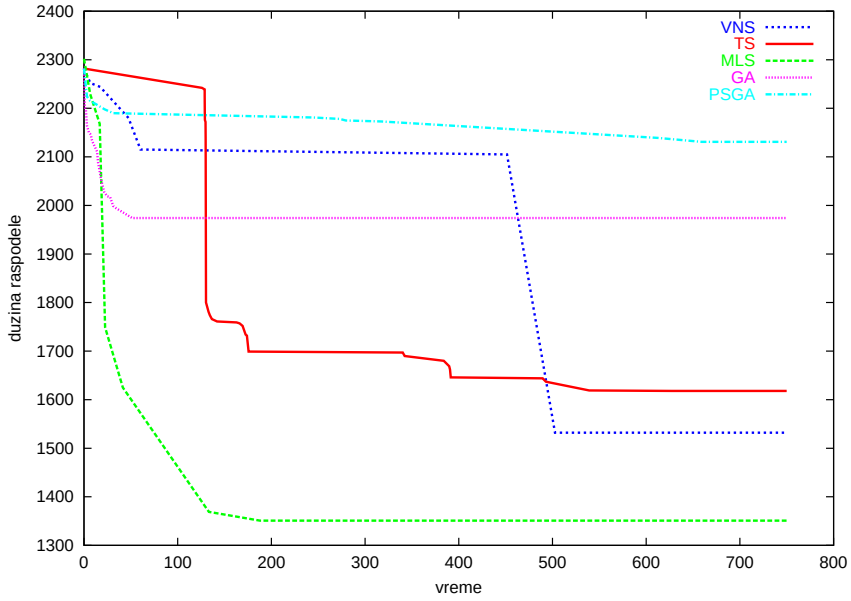
	CPU	*vreme (sekunde)*				
n	(max.)	VNS	TS	MLS	GA	PSGA
50	41.0	9.98	6.72	11.13	6.81	12.42
100	180.0	88.40	43.58	76.62	55.9	89.15
150	430.0	231.9	315.1	193.27	162.28	443.67
200	750.0	461.58	539.15	297.02	353.1	777.99
250	1200.0	864.12	945.21	702.68	556.44	1059.78
300	1600.0	1106.97	1042.15	852.52	837.58	744.00
350	2300.0	1709.71	1549.62	1425.41	865.38	1180.29
400	3000.0	2352.4	722.51	1833.1	1500.27	2069.75
450	3500.0	2956.76	1260.39	1527.29	1871.9	1597.36
500	4500.0	1992.32	3334.33	3404.45	2649.23	2861.92
Prosek	1750.1	1177.41	975.88	1032.35	885.89	1083.63

osnovu tabela 4.1 i 4.3 vidi se da je ona davala najlošije rezultate. Mišljenje je autora ovoga rada da je to posledica pre svega drugačije reprezentacije rešenja. Obzirom na to da prioritet zadataka nije dovoljan da se utvrdi jedinstven redosled kojim će se zadaci dodeljivati procesorima, neophodna su dodatna ispitivanja koja zahtevaju i dodatno vreme. Da bi se to potvrdilo, izvršeni su testovi raspoređivanja ovom metodom, sa kriterijumom zaustavljanja vezanim za broj generacija, tj. stavljeno je $N_g = 600$. Zaključak je da PSGA zahteva mnogo duže vreme za izvršavanje u odnosu na prethodno utvrđeno vreme t_{max}, a pri tome su rezultati još uvek bili lošiji od ostalih metoda.

U narednom odeljku analizirani su rezultati dobijeni raspoređivanjem test primera sa poznatim optimalnim rešenjem. Cilj ove analize je utvrđivanje razloga za velika odstupanja heuristički dobijenih raspodela od poznatih optimalnih kao i pokušaj da se dobijeni rezultati poprave.

Analize su usmerene na VNS metodu iz nekoliko razloga. Prvi je to što je ova metoda skoro nastala i u literaturi nije još dovoljno ispitana. Drugi razlog je što se izvedeni zaključci lako mogu primeniti i na druge metode bazirane na lokalnom pretraživanju. Evolutivni algoritmi bazirani su na drugačijoj tehnici, te je realno očekivati da i analize rezultata treba da idu drugim tokom.

Ovde se navode neke smernice za poboljšavanje metaheurističkih metoda.



Sl. 4.11: Poređenje heurističkih metoda na grafu zadataka sa poznatim optimalnim rešenjem za n=200 i $\rho = 50\%$

Na primer, kod TS metode može se koristiti više tabu lista (više memorija različitog tipa), mogu se uvoditi razni kriterijumi za diversifikaciju pretraživanja i slično [61]. Pri implementaciji GA, moguće je promeniti definicije genetskih operatora (višepoziciono ukrštanje, turnirska selekcija), a posebno bi bilo korisno primeniti neke novije tehnike, kao što su keširanje [84] i fino gradirana turnirska selekcija [54].

4.10. Podešavanje parametara VNS metode

U ovom odeljku opisane su modifikacije VNS metode u cilju poboljšavanja rezultata dobijenih za test primere sa poznatim optimalnim rešenjem. Analizirana je reprezentacija rešenja, eksperimentisano je sa izborom početnog rešenja i heuristike raspoređivanja, sa izborom parametara same metode (k_{max}, k_{step}, $plateaux$), sa definicijom operatora razmrdavanja i sa promenom kriterijuma zaustavljanja, ali je zaključak da izbor okolina za pretraživanje ima najveći uticaj na kvalitet konačnog rešenja.

4.10.1. Priroda rešenja i osobine reprezentacije

U tabelama 4.1 i 4.3 dati su rezultati dobijeni izvršavanjem osnovnih varijanti implementiranih metoda. Podešavanje parametera izvršeno je tako da se dobiju najbolji rezultati za neki polazni skup test primera. Jednostavne varijacije vrednosti parametara nisu dovele do značajnih poboljšanja dobijenih rezultata.

Tabela 4.5: Detaljni rezultati primene VNS metode na slučajno generisane grafove sa $n = 200$ zadataka i poznatim optimalnim rešenjem $SL_{opt} = 1200$.

n	ρ	$SL(CP)$	$SL(LS)$	$SL(VNS)$	t_{min}	t_{max}
200	0	1203	1200	1200	34.43	750.00
200	10	1955	1850	1801	672.01	750.00
200	20	2191	2111	2045	180.50	750.00
200	30	2266	2169	1574	660.43	750.00
200	40	2246	2192	1621	707.49	750.00
200	50	2282	2252	1532	502.69	750.00
200	60	2287	1248	1215	215.51	750.00
200	70	2305	1894	1210	669.56	750.00
200	80	2307	2284	1248	478.36	750.00
200	90	2354	1256	1200	32.33	750.00
prosečno	45.0	2243.67	1917.33	1494.00	461.58	750.00
% odstupanja		86.97	59.78	24.5	-	-

U tabelama 4.1 i 4.3 dati su zbirni (prosečni) rezultati za 10 primera sa istim brojem zadataka. Ti primeri imaju isti skup zadataka (u odnosu na dužine njihovog izvršavanja), istu optimalnu raspodelu (na zadatu višeprocesorsku arhitekturu koja se sastoji od $p = 4$ procesora povezanih u prsten, sl. 4.4), a razlikuju se po gustini zavisnosti među zadacima. Da bi se utvrdio kvalitet raspodele svakog pojedinog primera, izdvojeni su pojedinačni rezultati raspoređivnja grafova sa $n = 200$ zadataka i prikazani u tabeli 4.5. Za ovu grupu grafova, dužina optimalne raspodele iznosti $SL_{opt} = 1200$ ciklusa izračunavanja i dobija se raspoređivanjem dopustive permutacije 1,2,...,200 (prve permutacije u topološkom uređenju) primenom ES heuristike.

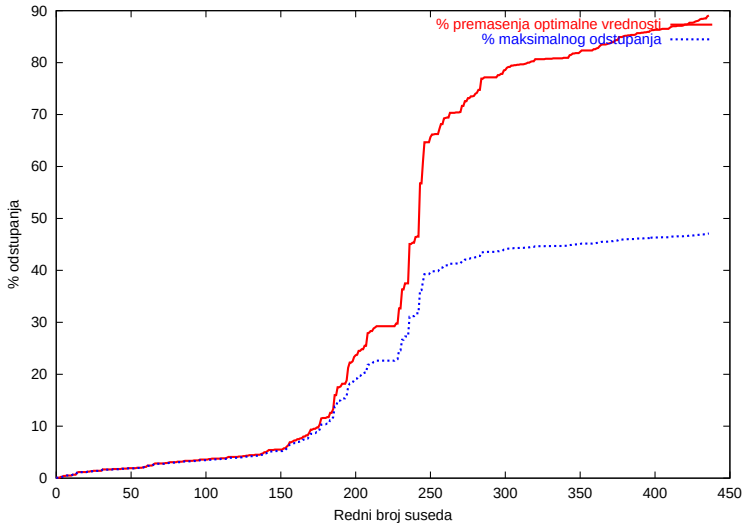
Na osnovu rezultata prikazanih u tabeli 4.5, može se zaključiti da su odstupanja mala prilikom raspoređivanja nezavisnih zadataka ($\rho = 0$) i prilikom raspoređivanja gustih grafova ($\rho > 50$). Grafovi srednje gustine zavisnosti među zadacima predstavljaju teške primere za raspoređivanje i

rezultuju velikim odstupanjima heurističke raspodele od poznate optimalne. Objašnjenje ovoga zaključka je veoma jednostavno. Nezavisne zadatke je lako "lepo upakovati" jer ne postoje intervali čekanja na međurezultate sa drugih procesora. To znači da je svaka od heurističkih raspodela prilično dobra i da ih je veoma lako popraviti jer i najmanje popravke prilično doprinose poboljšanju kvaliteta dobijenog rešenja. Sa druge strane, kod gustih grafova početna (heuristička) rešenja nisu kvalitetna, ali je zbog zavisnosti među zadacima prostor za pretraživanje (definisan dopustivim permutacijama) manji. Stoga procedura lokalnog pretraživanja brže uspeva da pronađe bolja rešenja i tako u istom vremenu dovede do značajnijih popravki i konačnog rešenja koje veoma malo odstupa od optimalnog.

Prilikom raspoređivanja grafova srednje gustine, polazno rešenje je loše, prostor za pretragu veliki, te je stoga izuzetno teško u zadatom vremenu ostvariti značajno poboljšanje rezultata raspoređivanja.

Grafovi zadataka sa poznatim optimalnim rešenjem generisani su tako da je ostvareno potpuno ravnomerno opterećenje (load balancing) među procesorima: svi procesori završavaju u isto vreme definisano zadatom dužinom optimalne raspodele SL_{opt}, nema intervala čekanja na međurezultate, tj. količina komunikacije je određena kao maksimalna moguća koja ne izaziva zastoje u procesu izračunavanja (videti [42], kao i prilog A za opis procesa slučajnog generisanja test primera). Primer optimalne raspodele ilustrovan je na slici A3. Proces generisanja obezbeđuje da je permutacija 1,2,...,n sigurno dopustiva i da uvek vodi (raspoređivanjem primenom ES heuristike) do optimalnog rešenja. Kao što je već napomenuto, problemi kombinatorne optimizacije najčešće nemaju jedinstveno optimalno rešenje. To je slučaj i kod problema raspoređivanja: postoji više permutacija koje primenom određenog pravila raspoređivanja (ES u ovom slučaju) dovode do optimalne raspodele.

Da bi se ilustrovala ova činjenica, pretražena je detaljno unapred (FORward smer) Swap-1 okolina prve dopustive permutacije (1,2,...,n) za koju je poznato da vodi optimalnom rešenju na primeru grafa sa $n = 200$ zadataka i gustinom veza među zadacima $\rho = 30\%$ (ovaj primer spada u kategoriju retkih grafova koji su se pokazali "teškima" za raspoređivanje). Cilj pretrage je ustanoviti koliko "optimalnih" susreda ima u okolini poznatog optimalnog rešenja. U navedenoj okolini, pronađeno je 679 dopustivih permutacija koje su rezultovale optimalnom raspodelom i 437 permutacija koje su dovele do lošijih rezultata. Ove brojke pre svega ukazuju na veličinu okoline za pretraživanje (679+437=1116$\approx 5.5n$). Dakle u svakom koraku LS procedure potrebno je generisati veliki broj suseda (dopustivih permutacija) i izvršiti



Sl. 4.12: % odstupanja od optimalnog rešenja u Swap-1 okolini.

preraspodelu promenjenog dela permutacije.

Na slici 4.12 prikazana su odstupanja svih neoptimalnih raspodela od optimalne, sortirana u neopadajućem poretku. Kriva predstavljena punom linijom prikazuje za koliko je premašena vrednost optimane dužine raspodele, dok tačkasta linija predstavlja normalizovanu vrednost prekoračenja. Kao što se sa slike može videti, permutacija koje dovode do iste neoptimalne raspodele takođe može biti nekoliko, što je ilustrovano delovima grafika paralelnim x-osi. Pri tome treba imati na umu da same raspodele ne moraju biti identične, postoje i simetrična rešenja u odnosu na indeksiranje procesora, o čemu je već bilo reči u ovom poglavlju.

Najupečatljiviji zaključak koji se može izvesti na osnovu grafika prikazanog na slici 4.12 je da najlošija rešenja odstupaju i do 90% od optimalnog. Obzirom da su trenutna razmatranja ograničena na najbližu okolinu optimalnog rešenja, uočljiva je jedna važna osobina MSPCD varijante problema raspoređivanja: male promene u strukturi rešenja mogu dovesti do velikih promena u vrednosti za dužinu raspodele. To je veoma atipično za probleme kombinatorne optimizacije, međutim karakteristično je za MSPCD i posledica je komunikacionih kašnjenja i strukture veza među procesorima. To se lako može pokazati na primeru reprezentacije rešenja pomoću listi zadataka

pridruženih procesorima. Na primer, CP-bazirana raspodela grafa zadataka ilustrovanog na Sl. 3.1 prikazana je na Sl. 4.13. Dužna ove raspodele iznosi $SL_{CP} = 56$ ciklusa izračunavanja.

P_1: 1, 2, 4, 6, 9, 10
P_2: 3, 7, 5, 8

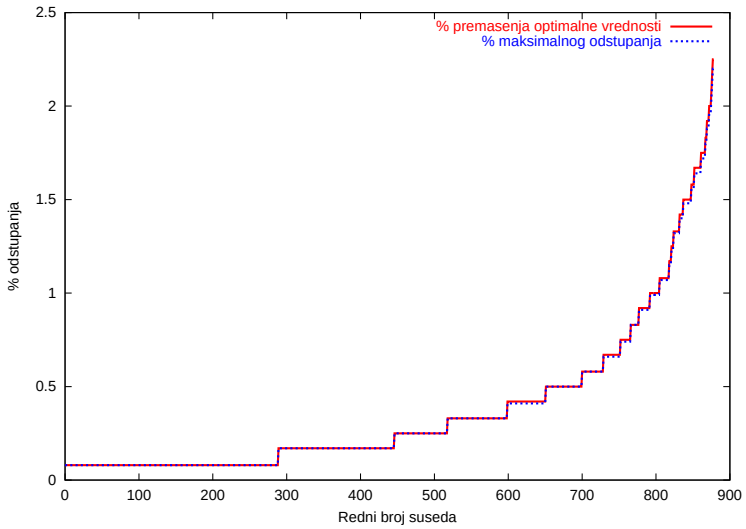
Sl. 4.13: CP-bazirana raspodela za graf sa slike 3.1

Kada se koristi reprezentacija pomoću lista zadataka dodeljinih procesorima na izvršavanje, Swap-1 okolina se definiše premeštanjem svakog od zadataka sa procesora kome je pridružen na sve preostale procesore u višeprocesorskom sistemu i to na sve moguće pozicije, tj. tako da se ne naruši zavisnost među zadacima. Kod raspodele prikazane na slici 4.13 ovako definisana okolina sadrži 26 suseda sa odstupanjem dobijenih raspodela od polazne, CP-bazirane u intervalu 0.02% do 41.5%. Ovo je jednostavan primer i nema indirektnih veza među procesorima, tako da je realno očekivati i mnogo veće razlike među rešenjima, kada se komunikaciona kašnjenja c_{ij} pojavljuju sa različitim koeficijentima d_{kl}, prilikom izračunavanja vremena potrebnog za transfer podataka između dva procesora (videti formulu (3.0) o odeljku 3.3.).

Da bi se uporedile dve reprezentacije, posmatrana je prva (u topološkom uređenju) dopustiva permutacija za primer sa slike 3.1. To je permutacija 1,2,3,...,10. Pretraživanjem Swap-1 okoline u FORWARD smeru dobija se 7 suseda (dopustivih permutacija) kod kojih raspodele odstupaju od 1.88 do 3.77%. U BACKWARD smeru ima 9 suseda i maksimum odstupanja je 9.43%. U odnosu na CP-baziranu permutaciju FORWARD pretraživanje dovodi do 6 suseda i maksimalnog odstupanja 7.55%, dok je u BACKWARD smeru, maksimalno odstupanje za 9 dobijenih suseda 3.77%. Ova analiza na malom primeru pokazuje prednosti reprezentacije pomoću dopustivih permutacija: čak i ako se zanemare preklapanja prilikom pretraživanja okolina u oba smera, okoline su manje kao i granice odstupanja rezultujućih raspodela od polazne.

Da bi se potvrdila osobina MSPCD varijante problema, da velike promene u vrednosti funkcije cilja (dužine raspodele) nakon malih promena u permutacijama potiču od komunikacionih kašnjenja i strukture veza među procesorima, analiziran je primer sa poznatim optimalnim rešenjem razmatran prilikom generisanja slike 4.12. Rezultati analize prikazani su na slikama 4.14 i 4.15. Slika 4.14 sadrži podatke analogne onima prikazanim na Sl. 4.12 u

slučaju da se prilikom raspoređivanja zanemaruje vreme potrebno za transfer podataka. Ovaj slučaj lako se modelira uvođenjem parametra za brzinu komunikacije *ccr* iz izraza (3.0). Kada se njegova vrednost postavi na nulu vreme komunikacije se anulira.



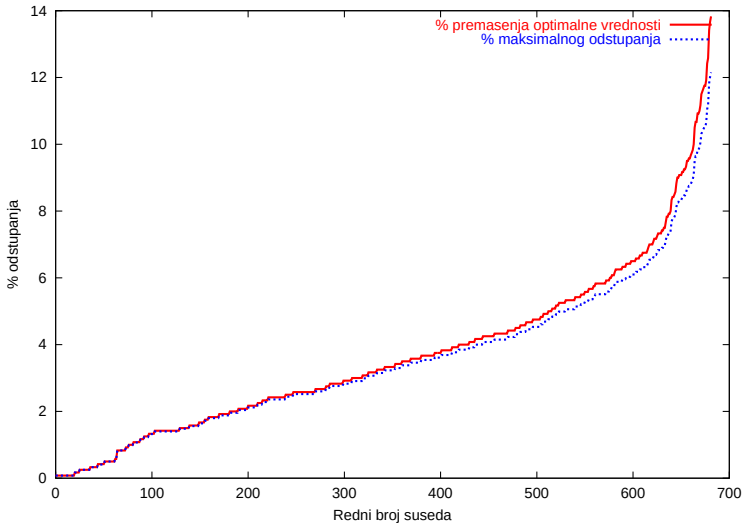
Sl. 4.14: Odstupanje raspodela u slučaju zanemarivanja komunikacije

Pretraživanjem Swap-1 okoline u FORWARD smeru u ovom slučaju dobija se 237 optimalnih raspodela i 879 neoptimalnih, a devijacije neoptimanih rešenja prikazane su na slici 4.14. Dakle, prvi zaključak je da ima manje optimanih rešenja u odnosu na slučaj sa komunikacijama. To se može objasniti time što se više ne vodi računa o izbegavanju komunikacionih kašnjenja već se zadaci raspoređuju samo na osnovu vremena izvršavanja. Međutim, odstupanja neoptimalnih rešenja su u ovom slučaju izrazito manja. U nedostatku komunikacionih kašnjenja, proizvedeni intervali čekanja posledice su isključivo zavisnosti podataka i samim tim značajno manji.

Drugi specijalan slučaj koji se može analizirati je kada vreme komunikacije nije zanemarljivo, ali je višeprocesorska arhitektura potpuno povezana, tj. postoje direktne veze između svaka dva procesora. Ovakva arhitektura modelira se matricom rastojanja $D_{p\times p}$ sledećeg oblika

$$d_{kl} = \begin{cases} 0, & k = l, \\ 1, & \text{u suprotnom} \end{cases}$$

Rezultati pretraživanja Swap-1 okoline u FORWARD smeru za ovaj slučaj prikazani su na slici 4.15. Moglo bi se reći da su ovi rezultati na neki način "između" prethodna dva slučaja: odnos broja optimalnih i neoptimalnih permutacija je 434 optimalnih prema 682 neoptimalne, dok je maksimalno odstupanje od optimalne raspodele 13.83%.



Sl. 4.15: Odstupanje raspodele za potpuno povezane višeprocesorske sisteme

U svim navedenim primerima Swap-1 okolina je pretraživna unapred (FORWARD smer). Za pretraživanje unazad (BACKWARD) numerički podaci su slični: $609 + 430 = 1039$ dopustivih permutacija (veći broj optimalnih) sa maksimalnim odstupanjem 89% u opštem slučaju (raspoređivanje na 2-dimenzionu hiperkocku uz prisustvo komunikacionih kašnjenja). Kada se zanemari komunikacija, dobiju se 266 optimalnih i 773 neoptimalne permutacije, kod kojih maksimano odstupanje iznosi 1.667%, a raspoređivanje na potpuno povezanu arhitekturu procesora rezultuje sa 429 optimalnih i 610 neoptimalnih permutacija i maksimalnim odstupanjem 6.08%.

Time je pokazano da se problem raspoređivanja na proizvoljne višeprocesorske sisteme, uz prisustvo komunikacionih kašnjenja, razlikuje od klasičnih problema kombinatorne optimizacije, što dodatno otežava njegovo rešavanje primenom standardnih metoda. Stoga je uloga metaheuristika još značajnija.

Međutim, jedan zaključak prethodne analize prividno je u kontradikciji

sa prethodnim konstatacijama: broj optimalnih rešenja najveći je u najsloženijoj varijanti problema. Broj optimalnih permutacija je 1.5 put veći od neoptimalnih. Pa kako to da ih je tako teško pronaći?

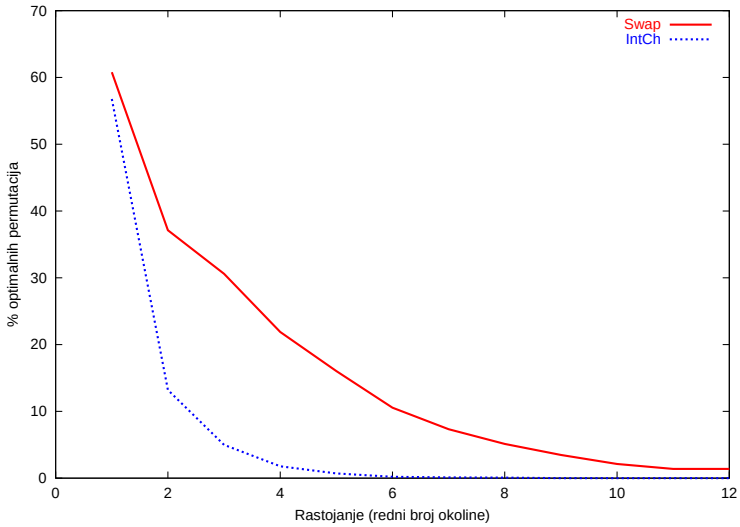
Naravno, ovaj zaključak odnosi se na najbližu okolinu jednog izabranog poznatog optimalnog rešanja. Dakle, pitanje bi bilo na mestu pod uslovom da je proces pretraživanja sigurno koncentrisan u blizini takvih rešenja. Međutim, prostor pretrage je ogroman i veliko je pitanje koliko su polazna rešenja i svako poboljšanje udaljeni od datih optimalnih rešenja. Dodatno pitanje je koliko su optimalna rešenja međusobno udaljena, tj. da li postoji jedinstven region u kome su koncentrisana sva optimalna rešenja ili postoji više, međusobno udaljenih oblasti koje sve sadrže veći broj optimalnih rešenja. Odgovori na ova pitanja nisu nimalo jednostavni i uvode potrebu za preciznim definisanjem rastojanja među permutacijama, što takođe predstavlja veliki problem.

Kao pokušaj odgovora na barem neka od ovih pitanja, ispitana je okolina 2 datog optimalnog rešenja. To znači da je za svakog neposrednog suseda polaznog optimalnog rešenja detaljno ispitana njegova najbliža okolina. Dobijeni rezultati sumirani su u tabeli 4.6. U tabeli su prikazani procenti optimalnih permutacija za svaki od slučajeva opisanih prilikom pretraživanja najbliže okoline. Dakle, u najopštijem slučaju pretraživanja najbliže okoline, oko 60% ukupnog broja suseda bile su optimalne permutacije, dok je neoptimalnih bilo oko 40%. Kod okoline 2, broj optimalnih permutacija spao je na oko 35% (samim tim neoptimalnih oko 65%), ali je još uvek značajno veliki. Sličan zaključak važi i prilikom razmatranja specijalnih slučajeva (potpuno povezane višeprocesorske arhitekture i zanemarivanja komunikacija).

Tabela 4.6: Procenat optimalnih permutacija u okolini 2.

	2-hiperkocka sa komun.	2-hiperkocka bez komun.	poptuna veza sa komun.
FORWARD	37.13%	4.94%	17.60%
BACKWARD	34.47%	7.34%	17.11%

Daljim pretraživanjem, okoline 3, 4, ..., dobijaju se podaci na osnovu kojih je generisan grafik predstavljen punom linijom na slici 4.16. Kriva predstavlja procenat optimalnih permutacija kada se raspoređivanje vrši na 2-hiperkocku i komunikacija je značajna, a okolina se pretražuje unapred.



Sl. 4.16: Procenat optimalnih permutacija u raznim okolinama

Očito je da se udaljenije okoline ne mogu pretraživati detaljno, te je stoga uzet dovoljno veliki slučajni uzorak rešenja u okolini k i detaljno ispitana najbliža okolina svakog od tih rešenja, čime je dobijena procena odnosa optimalnih i neoptimalnih permutacija u okolini $k + 1$.

Kao što se vidi sa slike 4.16, broj optimalnih rešenja opada sa rastojanjem od zadate optimalne permutacije, što navodi na zaključak da su su optimalne permutacije koncentrisane oko zadate permutacije ili po regionima. Istražena je i IntCh okolina na sličan način i rezultujuća kriva predstavljena tačkasto na slici 4.16. Kao što se sa slike 4.16 vidi, IntCh okoline su manje, pa je i broj optimalnih permutacija manji i brže opada. Važno je napomenuti da se kod IntCh okoline susedi međusobno više razlikuju (jer se vrši zamena mesta paru zadataka) pa je i to jedan od razloga zbog kojeg broj optimalnih permutacija brže opada.

Ova analiza pokazuje kako se menja broj permutacija u okolini nekog poznatog optimalnog rešenja, ono što je i dalje nepoznato je udaljenost početnog rešenja kao i proizvoljnog heurističkog rešenja od datog optimalnog. Da bi se to odredilo, potrebno je najpre definisati rastojanje između proizvoljne dve permutacije, a zatim za veliki broj (1000) izvršavanja bilo koje heurističke metode (VNS, na primer) izračunati rastojanja heurističkog rešenja od poz-

natog optimalnog.

Uobičajena mera rastojanja među permutacijama je pokoordinatno ili Hamingovo rastojanje (d_H). Međutim, u ovom slučaju ono nije adekvatno. Na primer, kao što se vidi sa slike 4.9, dopustive permutacije 1 2 3 4 5 6 7 8 9 10 i 1 2 3 4 7 5 6 8 9 10 su najbliži Swap-1 (BACKward) susedi za primer sa slike 3.1. Njihovo Hamingovo rastojanje je $d_H = 3$, tj. razlikuju se koordinate zadataka 5, 6 i 7. Iz toga bi mogao da se izvede zaključak: svaki put kad se premesti neki zadatak, menjaju se koordinate tri zadatka te se stoga Hamingovo rastojanje uvećava za 3. Ovaj zaključak, na žalost, nije ispravan. Ako se zadatak 7 pomeri još jedno mesto unapred, dobija se permutacija 1 2 3 7 4 5 6 8 9 10, čije Hamingovo rastojanje od polazne je $d_H = 4$.

Stoga se rastojanje između dve dopustive permutacije mora definisati drugačije, tj. zavisno od broja transformacija potrebnih da se od jedne permutacije dobije druga. U literaturi je ovo poznato kao *edit*–rastojanje i koristi se prilikom procesiranja tekstova. Dole navedena definicija nije egzaktna, zavisi od okoline, ali može poslužiti da se orijentaciono utvrdi udaljenost heurističkih rešenja od poznatog optimalnog.

Definicija 1. *Za zadatu okolinu $\mathcal{N}$,* rastojanje *između dve dopustive permutacije, definiše se kao minimalni broj transformacija koje tu okolinu opisuju potrebnih da se iz jedne permutacije pređe u drugu.*

Na primer permutacije 1 2 3 4 5 6 7 8 9 10 i 1 2 3 4 7 5 6 8 9 10 su na rastojanju $d_{Swap-1} = 1$. U odnosu na Swap-1 FORward njihovo rastojanje je $d_{Swap-1,FOR} = 2$ jer je potrebno premestiti prvo zadatak $i = 5$ iza zadatka $j = 7$, a u sledećem koraku zadatak $i = 6$ iza $j = 5$ da bi se od prve permutacije dobila druga. Dakle u odnosu na svaku okolinu (ili podokolinu) uzima se minimalni broj potrebnih transformacija.

Za Swap-1 okolinu, rastojanje između dve permutacije može se izračunati na sledeći način: prateći poziciju po poziciju u drugoj permutaciji, elementi prve se premeštaju na odgovarajuća mesta i određuje broj potrebnih premeštanja. Na primer, rastojanje između permutacija 1 2 3 5 6 7 4 8 i 1 2 3 4 5 6 7 8, koje predstavljaju dopustiva rešenja za primer sa slike 4.1, je 1 jer je dovoljno zadatak 4 premestiti iza zadatka 3 da se od prve permutacije dobije druga. Između permutacija 5 6 1 3 2 7 8 4 i 1 2 3 4 5 6 7 8 rastojanje je 4, jer je neophodno pomeriti zadatke 1, 2, 3 i 4 ulevo na odgovarajuća mesta. Problem koji se može javiti pri ovakvoj definiciji je da redosled transformacija narušava dopustivost permutacija koje se generišu kao međurezultati, tj. da li se na taj način ne napušta region dopustivosti permutacija. To se može

desiti ukoliko se tim premeštanjem preskače neki prethodnik zadatka koji se premešta. Rešenje daje sledeća lema.

Lema. *Neka su* $P_1 : p_1^1, p_2^1, \ldots, p_n^1$ *i* $P_2 : p_1^2, p_2^2, \ldots, p_n^2$ *dve dopustive permutacije zadataka. Tada je svaka od permutacija koja se dobija od* P_1 *premeštanjem unapred zadataka po redosledu definisanom redosledom zadataka u* P_2 *dopustiva.*

Dokaz. Neka je permutacija P_1' dobijena od P_1 premeštanjem zadatka p_i^1 na prvo mesto, tj. neka je $p_i^1 = p_1^2$. To znači da p_i^1 nema prethodnika (jer je P_2 dopustiva permutacija), te je rezultujuća permutacija P_1' sigurno dopustiva.

Za bilo koji drugi i-ti zadatak, koji treba dovesti na j-to mesto važi $p_i^1 = p_j^2$, $i > j$. Treba pokazati da ni za koje k, $j < k < i$ ne važi $p_k^1 \in Pred(p_i^1)$. Pretpostavka da to važi dovodi do kontradikcije, jer kad se zadatak p_i^1 premesti na j-tu poziciju, naredne transformacije ne menjaju pozicije $1, \ldots, j$ te se stoga zadatak p_k^1 ne može naći ispred $p_i^1 = p_j^2$ u permutaciji P_2. To bi značilo da P_2 nije dopustiva, što je suprotno polaznoj pretpostavci. ■

Logično je da se nameće pitanje povezanosti skupa dopustivih permutacija i za druge okoline, Swap-3, na primer ili IntCh. Pojam povezanosti skupa dopustivih permutacija u odnosu na neku izabranu okolinu (tj. transormaciju) uvodi se sledećom definicijom.

Definicija 2. *Za skup dopustivih permutacija zadataka kaže se da je* povezan *u odnosu na neku zadatu transformaciju ukoliko za svake dve dopustive permutacije iz tog skupa postoji konačan broj transformacija koje jednu permutaciju prevode u drugu, tako da su sve permutacije koje se tom prilikom generišu takodje u datom skupu.*

Drugim rečima, pitanje je da li se od neke dopustive permutacije može doći do neke druge, zadate primenom izabrane transformacije, a da se pri tome ne napušta region dopustivosti, tj. da su sve permutacije koje se usput generišu takođe dopustive. Odgovor na to pitanje daje sledeća teorema.

Teorema. *Skup* $X \subseteq S$ *svih dopustivih permutacija zadataka nekog grafa je povezan u odnosu na transformacije definisane premeštanjem pojedinačnih zadataka ili zamenom mesta paru zadataka, ali nije povezan u odnosu na promene mesta paru (ili trojci) susednih zadataka.*

Dokaz. Preciznije treba dokazati da je skup dopustivih permutacija povezan u odnosu na okoline Swap-1 i IntCh, ali nije povezan u odnosu na Swap-2 i Swap-3 okolinu.

Što se tiče okoline Swap-1, dokaz se dobija automatski na osnovu prethodne leme. U njoj je definisan postupak kako se od jedne dopustive permutacije sigurno generiše bilo koja druga zadata dopustiva permutacija primenom isključivo transformacija vezanih za Swap-1 okolinu.

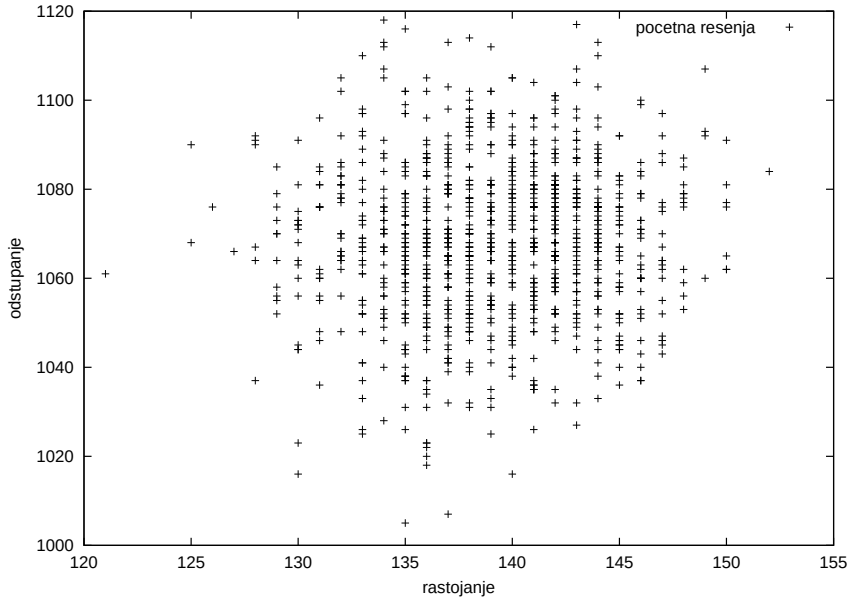
Vezano za okolinu IntCh, postupak je sledeći: izvršavaju se prvo dopustive zamene maksimalnog opsega, koje dovode zadatke na prava mesta (ili u njihovu blizinu), a zatim se zamenama mesta susednim zadacima, tj. finijim podešavanjima, rasporede zadaci u rezultujuću dopustivu permutaciju.

Za drugi deo teoreme (o nepovezanosti u odnosu na okoline Swap-2 i Swap-3) potrebno je naći kontraprimer, tj. par permutacija koje se ne mogu transformisati jedna u drugu primenom isključivo transformacija kojima se definiše jedna od navedenih okolina. Traženi par permutacija je 5 6 1 3 2 7 8 4 i 1 2 3 4 5 6 7 8 za primer sa slike 4.1. ■

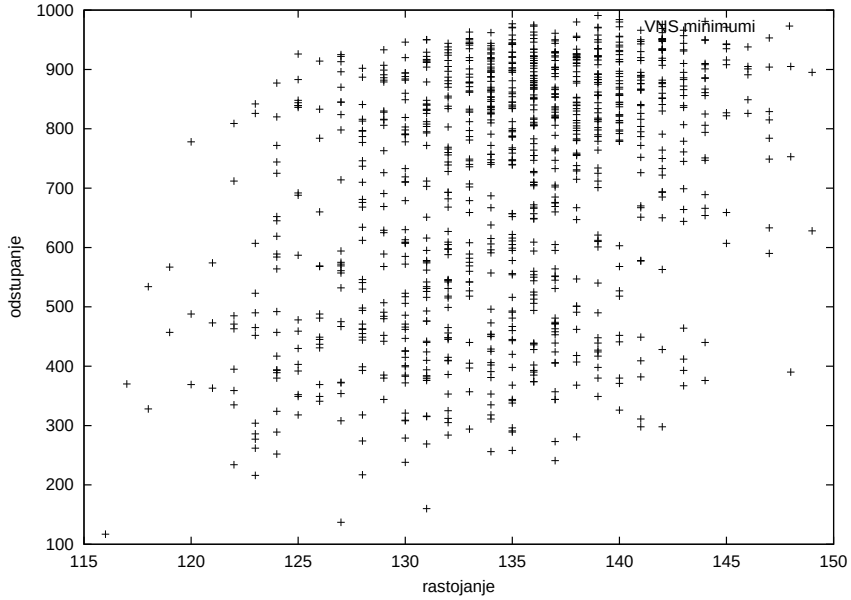
Zaključak koji iz ove teoreme sledi je da se Swap-1, odnosno IntCh okoline moraju obavezno uključiti u bilo koju kombinaciju okolina, da bi se obezbedilo korektno pretraživanje po skupu dopustivih permutacija.

Da bi se ispitala udaljenost heurističkih rešenja od optimalnog, izvršen je sledeći eksperiment: za primer sa $n = 200$ zadataka i $\rho = 30\%$ izvršavana je VNS metoda 1000 puta, svaki put od drugog slučajno izabranog početnog rešenja u zadatom vremenu $t_{max} = 750.00sec$. Za polazne (slučajno izabrane) i rezultujuće (heuristički dobijene) dopustive permutacije izračunavano je Swap-1 rastojanje u odnosu na prvu permutaciju u topološkom uređenju za koju se zna da je optimalna. Grafička reprezentacija tih rastojanja prikazana je na slikama 4.17 i 4.18.

Na apscisama grafika prikazanih na slikama 4.17 i 4.18 data su Swap-1 rastojanja između dopustivih permutacija izračunavana na način opisan u prethodnoj lemi, dok ordinate sadrže (apsolutno) odstupanje heuristički dobijene dužine raspodele od poznate optimalne. Svaka tačka na graficima odgovara jednom od 1000 rešenja (bilo početnih, za sliku 4.17 ili rezultujućih, na slici 4.18) predstavljenih dvema koordinatama: rastojanjem od poznatog optimalnog rešenja i odstupanjem u vrednosti za dužinu raspodele. Kao što se na osnovu ovih slika može zaključiti, bez obzira na značajno smanjenje u dužini raspodele (odstupanje je sa prosečnih 90% za početno rešenje svedeno na prosečnih 60%), heuristička rešenja su izuzetno daleko od zadatog optimalnog (rastojanja su uglavnom ostala u istom intervalu (125–150)).



Sl. 4.17: Rastojanja početnih rešenja od zadatog optimalnog za $n = 200$.



Sl. 4.18: Rastojanja heurističkih rešenja od zadatog optimalnog za $n = 200$.

Obzirom na to da su optimalna rešenja međusobno bliska, a prostor rešenja ogroman, očito je (kao što to slika 4.18 pokazuje) da VNS metoda pretražuje udaljene regione zbog čega se javljaju teškoće u pronalaženju optimalnog rešenja. Stoga je neophodno obezbediti diversifikaciju pretraživanja kako bi se posetio što veći deo prostora rešenja.

U nastavku ovog odeljka opisani su pokušaji poboljšanja efikasnosti VNS metode. Ispitan je uticaj različitih parametara na kvalitet dobijenog rešenja. Rezultati su prikazani za primere srednje dimenzije, tj. raspoređivani su grafovi sa $n = 200$ zadataka.

4.10.2. Početno rešenje i pravilo raspoređivanja

Postoje različiti parametri koji utiču na rezultat raspoređivanja primenom neke heurističke metode. Na primer, smer pretraživanja može se razmatrati kao parametar i ispitati uticaj promene smera (FORWARD-BACKWARD) kao i kombinacije na kvalitet dobijenog rešenja. Zaključak eksperimentalne analize je da u većini slučajeva pretraživanje unapred daje bolje rezultate.

Još jedan važan parametar je stepen poboljšanja (First Improvement – Best Improvement). U toku eksperimenata potvrđen je rezultat objavljen u [69], koji pokazuje da se FI izvršava brže i daje kvalitetnije konačno rešenje polazeći od slučajnih rešenja, tj. u svakom koraku VNS metode sastavljenom od izbora slučajnog rešenja u nekoj okolini i LS procedure počev od tog rešenja. FI strategija pretraživanja ubrzava proceduru lokalnog pretraživanja, jer se ne ispituje detaljno cela okolina.

Najbolje početno rešenje ne vodi obavezno ka najboljem konačnom rešenju dobijenom heurističkom metodom. Stoga su korišćene razne konstruktivne heuristike za dobijanje početnog rešenja i upoređeni rezultati dobijeni izvršavanjem VNS metode, počev od svakog tog početnog rešenja. Heuristike koje su primenjene su: (a) uređenje zadataka na osnovu neopadajuće vrednosti prioriteta baziranih na kritičnom putu (CP), (b) forsiranje najdužih zadatka, Largest Processing Time (LPT), (c) forsiranje zadataka sa najvećim brojem neposrednih sledbenika (SUCC), (d) uređenje zadataka na osnovu količine zahtevane komunikacije, kako sa prethodnicima tako i sa sledbenicima, (COMM) i (e) generisanje rešenja na slučajan način (RND). Na primeru MSPCD pokazalo se da CP heurističko rešenje koje je najkvalitetnije vodi najboljem konačnom rešenju. To još uvek nije dokaz da je to najbolje izabrano početno rešenje, mada veličina prostora rešenja ukazuje na to da se novim eksperimentima ne dobija mnogo: u tako ogromnom prostoru jedno

rešenje ne znači mnogo, a razlike među konačnim rešenjima nisu drastične.

Već je napomenuto da je jedna od prednosti reprezentacije rešenja pomoću dopustivih permutacija mogućnost kombinovanja sa heurističkim pravilima raspoređivanja. Veliki broj heurističkih pravila, koja se mogu pronaći u literaturi, pruža razne mogućnosti za kombinovanje. U ovom radu izabrano je pravilo najranijeg starta, koje se pokazalo kao najbolje među sledećim ispitanima: minimizacija intervala čekanja (Idleness Minimization – IM), izbor najboljeg puta (Preferred Path Selection – PPS [93]) i klasterovanje (DeClustering – DC [35]).

4.10.3. Varijacije parametara k_{max}, k_{step}, *plateaux*

Na osnovu opisa VNS metode realno je očekivati da povećanje vrednosti parametra k_{max} predstavlja osnovni način diversifikacije pretraživanja. Ranije analize pokazale su da su optimalna rešenja međusobno blizu i na toj činjenici bazira se intenzifikacija pretraživanja u okolini trenutno najboljeg rešenja kao osnovni princip VNS metode. Međutim, kako je rastojanje trenutno najboljeg rešenja u odnosu na poznato optimalno veliko (Sl. 4.18), potrebno je ispitati i udaljene regione. Pri tome treba naći pravi balans između intenzifikacije i diversifikacije, da bi se sprečio neželjeni efekat pretvaranja metode u prosto slučajno pretraživanje. Parametar k_{max} definiše maksimalnu udaljenost (u odnosu na trenutno najbolje rešenje) na kojoj će se vršiti pretraživanje. Razmatrane su kako konstantne vrednosti ovog parametra (10, 20), tako i one koje zavise od dimenzije problema ($n/4$, $n/2$ i $3n/4$).

Na osnovu dobijenih rezultata vidi se da parametar k_{max} ne utiče mnogo na rezultat pretraživanja. Detaljnim uvidom u rezultate raspoređivanja koji pokazuju koliko puta i u kojoj okolini je došlo do popravke trenutno najboljeg rešenja, broj suseda koji je tom prilikom razmatran, kao i koliko puta se pretraživanje obavilo počev od najudaljenijeg slučajnog rešenja, utvrđuje se razlog ovog zaključka. Posledica FI filozofije VNS heuristike je da se pretraživanje obavlja uglavnom u najbližoj okolini trenutno najboljeg rešenja: čim se tekuće rešenje popravi, pretraga se usmerava na njegovu najbližu okolinu $\mathcal{N}_1$. Tako se dobija da je vrednost k_{max} dostignuta više od 5 puta u slučaju da je njegova vrednost 10, a jednom ili nijednom ako uzima vrednosti $n/4$, $n/2$ ili $3n/4$.

To ne znači da apriori treba zanemariti uticaj ovoga parametra. Jedna mogućnost je da se uvedu dodatni parametri i ispitaju razne kombinacije njihovih vrednosti. Na primer, uvođenje parametra k_{step} omogućilo bi pres-

kakanje nekih okolina i samim tim povećalo verovatnoću da se više puta ispitaju udaljeni regioni. Naravno, uvek postoji opasnost da se tom prilikom sistematski preskoče dobra rešenja iz okolina koje nisu posećene. Još jedan od parametara koji utiče na preusmeravanje pretrage je *plateaux*, koji predstavlja verovatnoću da se neko rešenje koje vodi do dužine raspodele jednake trenutno najboljoj, usvoji kao novo tekuće najbolje rešenje. Već je pokazano da može postojati više dopustivih permutacija koje generišu iste raspodele zadataka ili raspodele koje nisu potpuno iste samo imaju istu dužinu. Prelazak u novo rešenje (sa nekom verovatnoćom koja može biti jednaka 1, što znači uvek) usmerava pretragu na nove regione i time može uticati na konačni rezultat VNS metode. Eksperimenti izvršeni za razne kombinacije vrednosti parametara prikazani su u tabeli 4.7.

Tabela 4.7: Rezultati raspoređivanja za različite kombinacije parametara k_{max}, k_{step} i *plateaux*

	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
10*	24.75	27.31	21.52	28.69	27.76	24.07	25.91	30.36	28.92
20	24.5	30.69	28.89	29.24	35.39	26.24	23.46	30.43	26.08
50	24.0	23.95	20.34	29.32	27.31	23.11	24.07	18.55	18.48
100	24.43	23.29	22.78	29.32	21.27	22.66	24.29	21.92	24.06
150	24.43	19.67	22.59	29.32	19.28	23.27	23.83	22.96	22.13

* za $k_{max} = 10$, vrednosti za k_{step} su 1, $k_{max}/5$ i $k_{max}/2$.

Rezultati ne pokazuju sistematično ponašanje VNS metode, ali može se izvući zaključak da je bolje ako se neke okoline preskoče i pretraga preusmeri na nove regione. To je i suština heurističkih metoda, sistematski izbor rešenja koja će se preskočiti prilikom pretraživanja, kako bi se povećala verovatnoća posećivanja "dobrih" rešenja.

Sa druge strane, tu je i izbor rešenja koja će se posetiti, tj. u slučaju VNS metode izbor onog suseda u k-toj okolini, čija će se najbliža okolina detaljno ispitati. Svaki put kada se operator razmrdavanja primeni u nekoj okolini k, bira se jedno rešenje kao polazno za dalju pretragu. Eksperimeni sa načinom izbora toga rešenja, tj. definicijom okoline i pravila razmrdavanja, opisani su u narednom odeljku.

4.10.4. Operacija razmrdavanja

U svim eksperimentima opisanim ranije, operator razmrdavanja koristio je k-Swap okolinu za generisanje slučajnog rešenja u k-toj okolini trenutno najbolje dopustive permutacije. Logično se nameće mogućnost upotrebe i drugih navedenih okolina.

Tabela 4.8: Rezultati dobijeni primenom različitih procedura razmrdavanja

Shake IntCh									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	24.21	25.53	23.33	26.73	29.29	22.67	27.07	24.04	20.74
100	26.12	19.12	31.35	26.73	22.42	23.55	27.06	24.21	22.2
150	26.12	21.44	24.78	26.73	18.68	25.67	27.06	16.66	24.62
Shake Hl									
100	23.02	23.84	26.08	26.53	23.70	25.57	27.90	25.68	26.64
150	23.02	31.27	29.14	26.53	31.78	30.72	27.90	31.87	24.32
Shake Hr									
100	22.48	26.52	25.51	30.27	25.78	25.4	31.78	26.54	22.80
150	22.48	27.35	25.07	30.27	27.52	25.24	31.78	24.17	25.24
Shake with $b = 5$									
50	33.10	30.54	22.07	24.16	28.76	35.07	34.72	27.34	27.88
100	33.01	25.49	25.49	24.16	19.54	25.99	34.83	18.12	25.69
100	33.01	21.97	28.48	24.16	18.73	26.74	34.83	19.22	24.75
Shake with $b = 10$									
50	37.63	27.51	31.68	36.12	29.86	28.72	38.18	32.52	26.12
100	37.68	27.3	19.88	36.12	29.47	20.37	38.18	25.27	24.62
150	37.68	23.76	24.92	36.12	27.89	24.37	38.18	25.44	21.28
Shake with $b = 20$									
50	27.45	35.52	32.37	29.13	24.46	27.27	27.32	25.00	37.53
100	27.72	22.24	32.93	29.19	28.43	23.67	27.32	21.74	26.18
150	27.86	24.67	23.02	29.65	22.47	23.84	27.17	26.22	19.35

U ranijim radovima [45, 43, 44], implementirane su sledeće procedure razmrdavanja: 1) k-Swap – k puta se izabere slučajno zadatak i premesti na novu poziciju u dopustivoj permutaciji; 2) k-IntCh – k puta se zameni mesto paru nesusednih zadataka uz očuvanje dopustivosti; 3) H_l ili H_r pravilo [56]; 4) uvođenje parametra b [73, 74]. Rezultati eksperimenata sa različitim pravilima kao i različitim vrednostima parametra b prikazani su u tabeli 4.8.

Korišćenje IntCh okolina može da utiče na preusmeravanje procesa pretraživanja, jer su promene na rešenjima u tim okolinama veće. Takođe, isprobana je ideja predložena u [56]: slučajno izabrani zadatak pomeri se zajedno sa svojim prethodnicima maksimalno ulevo, to znači da se svi zadaci koji od njega ne zavise nalaze desno od njega, a sleva su mu samo prethodnici. Ovo pravilo označeno je sa H_l. Ukoliko se slučajno izabrani zadatak premešta udesno zajedno sa svojim sledbenicima, tako da mu sa leve strane ostaju prethodnici i svi zadaci koji od njega ne zavise, dobijeno pravilo se označava sa H_r. Za k-tu okolinu, čitava procedura se ponavlja k puta. Ovakva definicija operatora razmrdavanja ne obezbeđuje potpunu slučajnost prilikom formiranja novih rešenja, ali su promene na rešenjima veće nego kod uobičajene implementacije k-Swap razmrdavanja. Poslednja varijanta operatora razmrdavanja sastoji se u izbroru b slučajnih rešenja u k-toj okolini i usvajanju najboljeg među njima za novu polaznu tačku LS koraka

Rezultati pokazuju da su odstupanja od poznatih optimalnih raspodela i dalje velika i da varijacije parametara nisu dovoljne za poboljšanje performansi VNS metode.

4.10.5. Modifikacije okolina

Sledeći korak koji se nameće je smanjenje okolina za pretraživanje u svakoj iteraciji lokalnog pretraživanja. Problem predstavlja ne toliko broj suseda koje treba generisati, već činjenica da za svakoga od njih treba izvršiti preraspodelu zadataka po procesorima kako bi se dobila nova vrednost za dužinu raspodele.

Obzirom na činjenicu da se neki susedi već preskaču pretraživanjem okolina samo u jednom smeru (unapred ili unazad), prva ideja je bila da se koristi kombinacija okolina. Na prvi pogled moglo bi se zaključiti da se na taj način okoline proširuju, međutim ne mora biti tako. Korišćenjem Swap-321 okoline za korak lokalnog pretraživanja (dakle kombinacija VNS i VND metode) uz FI stepen poboljšanja, očekuju se pozitivni efekti na kvalitet heurističkih rešenja. Ideja je da se prvo pretraže udaljenije okoline (koje su i manje) i da se krupnijim transformacijama dobiju značajnije popravke polaznog rešenja. FI i VNS algoritam obezbeđuju da se pretraživanje vrši u najudaljenijoj okolini (samim tim i najmanjoj) sve dok ima poboljšanja. Zatim se prelazi na srednju okolinu, ali samo do prvog sledećeg poboljšanja – algoritam tada vraća pretragu na prvu okolinu u nizu. Dakle, najveća okolina se detaljno pretražuje samo u poslednjoj iteraciji lokalnog pretraživanja, kada tekuće

rešenje nije moguće dalje popraviti.

U tabeli 4.9 dati su rezultati primene VNS-VND kombinacije, kao i još složenije kombinacije u kojoj je VNS-VND pretrazi dodata in IntCh okolina i pri tome pretraga naizmenično vršena unapred i unazad. Različite kombinacije VNS parametara (k_{max}, k_{step}, $plateaux$) i ovde su uzete u obzir.

Tabela 4.9: Rezultati raspoređivanja dobijeni kombinovanjem okolina

VNS-VND									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
100	28.47	23.82	22.45	25.85	20.77	21.92	28.05	21.07	25.21
150	28.47	21.16	21.46	25.85	16.69	23.37	28.05	17.31	19.75
VNS-VND-IntCh-FB									
100	41.00	19.07	21.22	34.56	23.08	23.66	37.57	25.5	24.39
150	38.87	25.43	28.45	34.72	24.91	21.73	36.6	30.38	30.28
VNS-VND-IntCh-FB, BI stepen poboljšanja									
100	34.16	27.60	24.65	31.72	35.10	24.42	32.73	24.64	31.20
150	33.95	26.72	23.12	31.53	27.67	30.45	32.72	29.42	29.57

Kao što se vidi iz tabele 4.9, odstupanja su i dalje uglavnom iznad 20%. Razlog ovako loših rezultata je što FI strategija u velikim okolinama pokazuje tendenciju pukog "slučajnog izbora". Sa druge strane, BI strategija u svakoj iteraciji završava u lošijem rešenju, pa je stoga i konačno rešenje nekvalitetno (videti poslednji deo tabele 4.9).

Sve prethodne analize ukazuju na to da je neophodno smanjiti okoline, te su stoga u nastavku opisane strategije koje su korišćene da bi se prilikom smanjivanja zadržala rešenja koja potencijalno vode boljim finalnim raspodelama. Prilikom odlučivanja koja rešenja treba da čine smanjenu okolinu, treba pronaći pravila izbora rešenja za koja se pretpostavlja da se njihovim modifikacijama može značajno popraviti trenutna raspodela. Ovde je razmatrano nekoliko strategija, sve su opisane i prikazani su rezultati raspodela primera sa poznatim optimalnim rešenjima za svaki od slučajeva.

Prva ideja za redukciju broja dopustivih permutacija koje čine okolinu je izbor zadataka sa najopterećenijeg procesora. Preciznije, okolina se dobija na sledeći način: primeni se operator razmrdavanja u datoj okolini na dopustivu permutaciju koja predstavlja trenutno najbolje rešenje; odredi se odgovarajuća raspodela primenom heurističkog pravila ES; odredi se indeks pro-

cesora koji definiše dužinu raspodele, tj. onaj koji poslednji završava rad; u permutaciji se za premeštanje obeleže zadaci koji su dodeljeni na izvršavanje tom procesoru.

Kada procedura lokalnog pretraživanja otpočne svoje izvršavanje, ona preskače (tj. ne pomera) zadatke koji nisu obeleženi u prethodnom postupku. Time se okolina smanjuje približno n/p puta (u slučaju idealne raspodele i ravnomernog opterećenja procesora, svaki od p procesora bi izvršavao n/p zadataka, naravno u realnom slučaju to ne mora da važi). Ova strategija bazirana je na pretpostavci da će premeštanje zadataka dodeljenih najopterećenijem procesoru na nova mesta u dopustivoj permutaciji za posledicu imati njihovo dodeljivanje drugom procesoru, pa samim tim i rasterećenje procesora koji je u prethodnom slučaju najduže radio. Postupak se pokazao efikasnim prilikom raspoređivnja nezavisnih zadataka, dok u slučaju MSPCD, komunikacije mogu izazvati dodatne probleme. Međutim, kako komunikacije utiču na ukupnu dužinu rada procesora, može se očekivati da neko od premeštanja zadatka dovede i do smanjenja zahtevanog vremena komunikacije.

Opisana strategija nazvana je redukovana procesorski bazirana i u oznaci nosi prefiks RP. Primenjena je na Swap-1 okolinu (RPVNS) i na Swap-321 (RPVND), a rezultati su prikazani u prvih 6 vrsta tabele 4.10. Upoređivanjem rezultata utvrđuje se da je jednostavnija verzija (RPVNS) efikasnija i smanjuje odstupanje na 16-20%.

Druga ideja eksplicitno uzima u obzir smanjenje komunikacija, zato je nazvana komunikaciono bazirana i nosi prefiks RC. Suština je u sledećem: u tekućoj raspodeli odredi se zadatak koji zahteva najviše prenosa podataka između procesora (saberu se sve komunikacije kako sa prethodnicima tako i sa sledbenicima pomnožene rastojanjem procesora kojima su dodeljeni na izvršavanje od procesora na kome se izvršava dati zadatak). Za premeštanje se obeleže svi prethodnici takvog zadatka i startuje LS procedura koja pretražuje samo po tim prethodnicima.

Premeštanje prethodnika zadatka koji zahteva najviše komunikacija na nova mesta u permutaciji može dovesti do njihovog dodeljivanja istom procesoru (ili barem nekom koji je na manjem rastojanju) i tako za posledicu imati smanjenje zahtevane komunikacije. Sledbenici nisu razmatrani eksplicitno jer će premeštanje prethodnika imati uticaja i na raspodelu sledbenika. Preciznije premeštanje nekog zadatka u permutaciji ima uticaja na raspodelu svih zadataka koji se nalaze desno od njega u permutaciji, a logično je da "prethodnici" zauzimaju mesta sa manjim indeksima i samim tim imaju uticaja na veći deo permutacije.

Tabela 4.10: Rezultati raspoređivanja pretraživanjem redukovanih okolina

RPVNS									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	15.23	19.85	21.45	17.83	18.82	16.20	16.32	16.93	17.15
100	16.75	19.23	24.87	17.12	17.12	20.53	16.07	16.48	16.06
150	16.87	16.61	19.53	18.20	17.63	21.99	16.95	16.52	21.56
RPVND									
	$plateaux$=0.0			$plateaux$=0.5			$plataux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	18.57	23.08	26.57	20.54	23.13	23.12	22.92	24.42	24.41
100	20.46	21.19	26.70	18.92	20.54	24.30	26.47	18.34	23.44
150	25.88	21.39	23.81	20.02	27.10	22.88	27.41	22.03	31.80
RCVNS									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	18.61	17.02	16.63	16.73	16.41	18.79	18.49	16.61	18.65
100	18.47	13.94	17.17	16.84	14.97	16.83	18.11	13.89	16.30
150	18.59	16.43	15.64	16.96	16.96	15.12	18.24	16.84	15.27
RPDVNS									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	22.20	16.53	16.62	21.56	20.17	23.97	20.55	21.91	21.92
100	21.12	14.02	21.55	22.37	19.58	20.74	23.59	20.63	18.84
150	25.56	24.59	23.49	19.46	22.57	22.89	18.47	24.19	22.32
RTDVNS									
	$plateaux$=0.0			$plateaux$=0.5			$plateaux$=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	20.09	19.69	33.17	19.57	20.73	23.50	23.67	23.50	23.86
100	21.36	20.15	21.64	22.53	20.66	19.38	15.66	21.91	18.66
150	23.54	18.41	17.12	21.32	20.27	25.34	19.67	21.02	26.46

Rezultati su prikazani u tabeli 4.10, kao treća grupa vrednosti (RCVNS) i pokazuju nova poboljšanja konačnog rešenja. Važno je napomenuti da su u slučaju pretraživanja redukovanih okolina premeštanja zadataka vršena u oba smera (vrednost parametra FORWARD-BACKWARD nije uzimana u obzir). Kako se premešta samo mali broj zadataka to ni preklapanja, tj. razmatranja istih suseda više puta nisu velika.

Implementirane su još dve strategije za redukciju okoline bazirane na ideji

dekompozicije. Pravu dekompoziciju nije bilo moguće izvršiti zato što ne postoji mogućnost da se bilo šta fiksira u rešenju. Zavisnosti postoje na raznim nivoima i uslovljene su povezanošću zadataka u grafu kao i reprezentacijom preko dopustivih permutacija.

Prva strategija nazvana RPDVNS (Restricted Processor-based Decomposition-like VNS), sastoji se u slučajnom izboru j=1,2,3,...,p procesora za svaki korak LS procedure i obeležavanju za premeštanje samo zadataka dodeljenih tim procesorima. Dakle, u prvoj iteraciji slučajno se izabere jedan procesor i premeštaju zadaci dodeljeni tom procesoru. U drugoj iteraciji u igri su dva slučajno izabrana procesora i broj zadataka za premeštanje se povećava. Nakon p iteracija, pretražuje se cela okolina, a u iteraciji $p + 1$ ponovo se bira samo jedan procesor. Obzirom na slučajnost izbora procesora i progres u popravljanju tekućeg rešenja, očito je da je ovakav postupak korektan.

Druga varijanta zasnovana na ideji dekompozicije bazirana je na slučajnom izboru zadataka koji će se premeštati. U svakoj iteraciji LS procedure izabere se slučajno i=1,2,3,...,n zadataka i oni se obeleže za premeštanje. Očito je da se u ovom slučaju u svakoj n-toj iteraciji pretražuje cela okolina, a zatim se u narednoj okolina maksimalno smanjuje izborom samo jednog zadatka za premeštanje. Ova varijanta obeležena je sa RTDVNS (Restricted Task-based Decomposition-like VNS), a rezultati prikazani u poslednja dva dela tabele 4.10 pokazuju da varijante bazirane na dekompoziciji poboljšavaju rezultate osnovne VNS metode, mada najbolja varijanta ostaje RCVNS.

4.10.6. Kriterijum zaustavljanja

U svim prethodnim eksperimentima kriterijum zaustavljanja bio je dozvoljeno vreme rada. Već je napomenuto da postoje i drugi kriterijumi, ali da je vremenski izabran zbog obezbeđivanja ravnopravnih uslova za poređenje metaheurističkih metoda.

Da bi se ispitalo ponašanje VNS metode, startovana je osnovna varijanta (VNS, Swap-1, FI, FOR, $k_{max} = n/2$, $k_{step} = k_{max}/10$) sa kriterijumom zaustavljanja 5 iteracija bez poboljšanja. Ovaj kriterijum podrazumeva da se sve okoline (od 1 do k_{max}) ispitaju 5 puta bez poboljšanja trenutno najboljeg rešenja. Prosečno vreme rada za 10 primera sa $n = 200$ zadataka je 14038.64 *sec* (pri čemu je oko polovina bila neproduktivna, baš zbog novog kriterijuma zaustavljanja) dok je prosečno odstupanje svedeno na 9.9%. Poređenja radi, vremena izvršavanja VNS metode sa novim kriteri-

jumom zaustavljanja korišćena su kod ostalih metoda i rezultati su: za MLS prosečno odstupanje je 13.96%, za GA 33.73%, a za TS 34.67%. Zaključak je da produženje dozvoljenog vremena izvršavanja dovodi do poboljšanja konačnog rešenja, što svedoči o složenosti problema.

Kod najbolje redukovane varijante (RCVNS) postupak je brži jer se okolina značajno smanjila. Posledica toga je da je vreme potrebno za 5 iteracija bez poboljšanja kraće od prethodnog vremenskog kriterijuma. Stoga je novi zadati kriterijum definisan kao 50 iteracija bez poboljšanja. Dobijeno odstupanje je 12.28% ostvareno za 1150.71 *sec*, dok je ukupno vreme izvršavanja (uključujući i fazu bez poboljšanja) 4192.62 *sec*. Vremenski kriterijum zasnovan na 5 iteracija bez poboljšanja za osnovnu VNS metodu (14038.64 *sec*), primenjen na RCVNS rezultuje odstupanjem od 10.45% u proseku, dobijenih za prosečno vreme od 6513.88 *sec*.

Parametarska analiza opisana u ovom odeljku izvršena je za skup primera sa poznatim optimalnim rešenjam. Osnovni zaključak je da se redukcijom veličine okoline za pretraživanje u LS proceduri mogu popraviti rezultati raspoređivanja. Međutim, ovako modifikovane metode ne daju dobre rezultate za skup slučajno generisanih primera, kod kojih optimalno rešenje nije poznato, jer su strukture primera veoma različite. Na osnovu toga može se pretpostaviti da postoji mogućnost za dodatne eksperimente u potrazi za efikasnijim modifikacijama metaheurističkih metoda baziranih na reprezentaciji rešenja pomoću dopustivih permutacija.

4.10.7. Rezultati primene hibrida

U ovom odeljku dati su rezultati primene tri hibridizovane metode na problem raspoređivanja. Deo ovih rezultata objavjjen je nedavno u radu [33]. U tabeli 4.11 dati su rezultati primene najbolje varijante VNS metode (RVNS) i složenih metoda (dobijenih kombinovanjem VNS, TS i GA) na raspoređivanje primera sa $n = 200$ zadataka i poznatim dužinama optimalnih raspodela $SL_{OPT} = 1200$. U poslednjem redu tabele ponovljeni su rezultati odgovarajuće vrste tabele 4.3, da bi se uporedile osnovne verzije metoda sa hibridima.

Dobijeni rezultati ukazuju na to da se hibridizacijom metoda mogu popraviti performanse svake od metoda. Međutim, rezultati dobijeni hibridizacijom osnovnih varijanti metoda još uvek daju lošije rezultate od onih koji se dobijaju strategijama za smanjivanje okolina i podešavanje parametara. Na osnovu toga može se zaključiti da je hibridizacija jedna od plodnih tema za

Tabela 4.11: Rezultati primene hibrida metaheurističkih metoda.

n	ρ	$SL(RVNS)$	$SL(MVND)$	$SL(VND-TS)$	$SL(MEM)$	t_{max}
200	0	1200	1200	1200	1200	750.00
200	10	1483	1820	1830	1941	750.00
200	20	1513	1857	2047	2033	750.00
200	30	1713	1844	2157	1518	750.00
200	40	1357	1320	2145	1683	750.00
200	50	1200	1304	1618	1572	750.00
200	60	1200	1225	1287	1265	750.00
200	70	1200	1200	1209	1244	750.00
200	80	1200	1214	1200	1264	750.00
200	90	1200	1200	1200	1200	750.00
prosečno	45.0	1326.60	1418.40	1589.30	1492.00	750.00
% odstupanja		13.94	18.20	32.44	24.33	750.00
% odstupanja osnovne metode		24.50	21.18	38.37	49.47	750.00

dalja istraživanja u ovoj oblasti.

5. Paralelizacija metode promenljivih okolina

Iako su na mnogim primerima primene pokazale svoju efikasnost, metaheurističke metode često imaju teškoće prilikom rešavanja nekih složenijih NP-teških problema (kao što je problem koji se razmatra u ovome radu) ili problema velikih dimenzija (na primer, problem p-medijane sa preko 3000 lokacija, tj. dimenzije $n > 3000$). Jedan od načina da se ove teškoće prevaziđu je da se originalne metaheurističke metode paralelizuju i da se njihovim distribuiranim izvršavanjem na više procesora ostvari bar jedan od mogućih ciljeva: ubrzavanje neophodnih izračunavanja (tj. dobijanje rešenja za kraće vreme izvršavanja metode) ili popravljanje kvaliteta rešenja (tj. dobijanje boljeg rešenja u istom ili kraćem vremenu izvršavanja).

U ovoj glavi daje se detaljan pregled strategija za paralelizaciju metaheurističkih metoda koje se koriste za rešavanje problema raspoređivanja. Najpre se opisuju generalni trendovi u razvoju paralelnih metaheuritičkih metoda [26], zatim konkretne paralelne metode predložene u literaturi za rešavanje problema raspoređivanja [88, 114, 116]. Na kraju poglavlja opisane su strategije paralelizacije metode promenljivih okolina (VNS) predložene u ovom radu i detalji vezani za paralelnu implementaciju VNS metode. Rezultati raspoređivanja slučajno generisanih grafova zadataka opisani su na kraju ove glave.

5.1. Paralelizam u metaheuristikama

U radu [26] data je generalna slika o stanju u oblasti paralelnih metaheuristika. Ciljevi toga rada bili su:

1. prikazati pregled trenutnog stanja (state-of-the-art) u razvoju paralelnih metaheuristika i dobijenim rezultatima;

2. diskusija uopštenih principa za dizajn i implementaciju koji se mogu primeniti na većinu metaheuristika;

3. opis tih principa na primerima GA, TS i SA;

4. pregled trendova i pravaca u istraživanjima koji najviše obećavaju.

Osnovni cilj paralelnog izvršavanja je da ubrza izračunavanja deleći ih između više procesora. Sa stanovišta dizajniranja algoritama, jednostavne strategije paralelizacije koriste parcijalni poredak među koracima algoritma tj. postojanje skupa operacija koje se mogu izvršiti paralelno (istovremeno, nezavisno) bez uticaja na metodu rešavanja i na dobijeno rešenje. One koriste "prirodni" paralelizam koji već postoji u algoritmu. Parcijalni poredak operacija u algoritmu omogućava dva osnovna vida paralelizma: paralelizam podataka (data, d) i funkcionalni paralelizam (functional, f).

Paralelizam podataka podrazumeva da se iste operacije istovremeno izvršavaju nad raznim podacima (SIMD model). Funkcionalni paralelizam je kada se nad raznim podacima istovremeno izvršavaju razni skupovi operacija (MIMD model).

Algoritmi koji se izvršavaju nad neregularnim strukturama podataka, kao što su grafovi, ili nad podacima sa jakim zavisnostima među različitim operacijama teški su za paralelizaciju samo jednim od ova dva vida paralelizma. Metaheuristike generalno spadaju u klasu algoritama koji se teško paralelizuju, ali ima interesantnih rezultata iz te oblasti i novih izazova.

Tipovi paralelizma kod metaheuristika

Metaheuristike nije jednostavno posmatrati sa stanovišta izdvajanja (d) ili (f) paralelizma. LS petlje su međusobno zavisne, kao i definisanje populacije u svakoj generaciji GA. Međutim unutar LS ili unutar jedne generacije ima prostora za paralelizaciju. Sem toga, kada se vrši slučajno ponavljanje izvršavanja, to može biti izvor funkcionalnog paralelizma jer su iteracije iz raznih polaznih rešenja nezavisne.

Metaheuristike kao algoritmi imaju ograničene mogućnosti za primenu paralelizma, ali kao metode za rešavanje problema pružaju razne mogućnosti za paralelno izračunavanje (za kreiranje novih paralelnih metoda koje samo "liče" na originalne, sekvencijalne metaheuristike).

Na primer, metoda grananja i ograničavanja (Branch and Bound, B&B): na raznim procesorima izvršavaju se razne tehnike grananja, distribucija pod-

drveta na razne procesore (to nije ni (d) ni (f) paralelizam, ali je veoma uobičajen, razlika je u tome što sekvencijalni i paralelni program ne izvršavaju isti skup instrukcija, ne izvršavaju isti algoritam, međutim oba nužno nalaze optimalno rešenje).

Slično važi i za metaheuristike. Paralelne metode koje ne ispoljavaju karakteristike ni (d) ni (f), lako je konstruisati, na primer, start iz različitih polaznih rešenja na različitim procesorima. Ono što je ovde novo je da sekvencijalni i paralelni metaheuristički algoritam ne moraju dati isto rešenje, te stoga treba analizirati ne samo ubrzanje nego i kvalitet dobijenog rešenja.

U [26] strategije paralelizacije metaheuristika klasifikovane su u tri tipa zavisno od izvora paralelizma koji se koristi.

Tip 1: paralelizacija izračunavanja u okviru iteracije. To je paralelizacija niskog nivoa i cilj je isključivo ubrzanje, a ne i podizanje kvaliteta dobijenog rešenja.

Tip 2: distribucija domena pretraživanja. Različiti podskupovi rešenja se razmatraju na različitim procesorima i stoga se serijski i paralelni algoritam razlikuju. Taj pristup je korišćen u [88] i za paralelizaciju VNS-a za problem p-medijane [24].

Tip 3: distribucija pravila za odlučivanje. ceo prostor rešenja razmatra se na svim procesorima, ali različitim metodama, na primer, kooperacija metoda koje su startovane za različite vrednosti parametara [28].

Paralelizam tipa 1

Ovaj vid paralelizma omogućava isključivo ubrzanje izračunavanja. Paralelizovani program radi isto što i sekvencijalni, samo se trudi da u okviru jedne iteracije uradi što je moguće više izračunavanja istovremeno (konkurentno).

U nekim slučajevima mogu da se pojave razlike u izvršavanju sekvencijalne i paralelne verzije programa, na primer ako paralelni program koristi više polaznih (inicijalnih) rešenja, paralelno, ali kako je i tu suština ista (postizanje ubrzanja u izračunavanjima) i ovaj slučaj se kategorizuje kao tip 1.

Paralelizam tipa 2

Pod ovaj slučaj podpadaju dekompozicije raznih vrsta i najčešće se implementira u nadređeni–podređeni procesor (master–slave) okruženju:

- Nadređeni procesor izvodi dekompoziciju i menja je tokom izvršavanja (ako je potrebno). Način promene dekompozicije i vreme kada će se ona izvoditi mogu se definisati unapred ili tokom samog izvršavanja.

- Podređeni procesori istovremeno obrađuju svoju particiju, međusobno nezavisno, bilo da tuđi deo smatraju fiksnim i bez uticaja na sopstvena izračunavanja, bilo da vide ceo skup promenljivih i dozvoljeno im je da i na tuđem delu vrše promene.

- U slučaju da podređeni procesori vide ceo prostor promenljivih, nadređeni ima više posla oko kombinovanja prikupljenih rezultata i sklapanja tekućeg rešenja.

Dekompozicija može da dovede do toga da veliki deo prostora rešenja ostane neistražen pa se zato izvode ponavljanja sa različitim dekompozicijama (čime se donekle anulira taj problem).

Paralelizam tipa 3

Oba prethodna tipa paralelizma orijentisana su na jedinstveno pretraživanje. Paralelizam tipa 3 sastoji se iz više konkurentnih pretraživanja raznim metodama, po celom ili dekomponovanom prostoru. Različitost se ogleda u: raznim metodama, raznim početnim rešenjima, raznim pravilima za komunikaciju tokom izvršavanja (a može se i samo na kraju izabrati najbolje rešenje).

To znači da paralelni program u opštem slučaju daje različita rešenja od sekvencijalnog (i, po nekim studijama, bolja čak i kad ima manje dozvoljenog vremena za izvršavanje). Takvi su na primer, višestruko povezani (multi–thread) paralelni programi.

Posledica toga je da mera performansi takvih programa ne može biti klasična jer je teško izbeći preklapanja u radu raznih procesora. Drugo, to su obično asinhrone metode, a one su vremenski zavisne (za isti ulaz, pod istim uslovima izvršavanja, mogu dati različite izlaze[1]).

Ova klasifikacija je dovoljno opšta da se može primeniti na bilo koju metaheurističku metodu. U literaturi postoje i druge klasifikacije, uglavnom orijentisane na načine za paralelizaciju jedne izabrane metaheuristike, ali se većina njih može svesti na neki od tri navedena tipa paralelizma.

[1]Zavisno od redosleda komunikacija u svakom izvršavanju.

Pregled postojećih strategija paralelizacije VNS metode

Obzirom da je metoda relativno nova, nema mnogo rezultata u literaturi vezanih za paralelizaciju ove metode. Za sada su se pojavila dva rada u kojima se predlažu paralelne verzije metode promenljivih okolina za rešavanje problema p-medijane. Strategije za paralelizaciju VNS metode i oba rada opisani su u preglednom članku [104]. Ne ulazeći u suštinu problema koji se rešava, ovde su izložene samo opšte ideje paralelizacije same metode, predložene u tim radovima.

U radu [58] predložene su i upoređene tri varijante paralelne VNS procedure. Prva je paralelizacija LS procedure (paralelizam tipa 1), tj. paralelno pretraživanje okoline u cilju ubrzanja ovog najzahtevnijeg dela VNS metode. Dobijeni rezultati su se podudarali sa sekvencijalnim izvršavanjem, cilj je bio jedino da se pretraga ubrza. Druga varijanta je nezavisno izvršavanje više VNS procedura i izbor najboljeg rešenja na kraju. Ova varijanta predstavlja višestartni VNS, pri čemu se za razna početna rešenja VNS procedure startuju istovremeno. Treća varijanta je paralelizam nižeg nivoa od prethodnog. Sastoji se u tome da se kombinacija koja se sastoji od razmrdavanja u okolini k (za neko k) i odgovarajuće LS procedure izvršava istovremeno na svim procesorima, ali za različite vrednosti parametra k. Tako ova varijanta u stvari predstavlja diversifikaciju u smislu boljeg pretraživanja u odnosu na datu okolinu k. Kriterijum zaustavljanja za VNS metodu implementiranu u ovom radu je maksimalni broj okolina, tj. procedura se zaustavlja kad je $k = k_{max}$. Stoga su kao mera efikasnosti korišćena vremena rada paralelne implementacije i broj iteracija. Sve implementacije su sinhrone, mada su iz literature [26] mane sinhronog izvršavanja dobro poznate.

Pristup paralelizaciji metode promenljivih okolina predložen u radu [24] potpuno je drugačiji. Autori su pokušali da maksimalno sačuvaju originalnu filozofiju metode i uz pomoć jednog nadređenog i q podređenih procesora realizovali "paralelno, asinhrono i kooperativno" pretraživanje više okolina. Naime, uloga nadređenog procesora je da vodi računa o kriterijumu zaustavljanja, o ažuriranju trenutno najboljeg rešenja i njegovom distribuiranju podređenim procesorima, kao i o donošenju odluke o usmeravanju pretraživanja ili završetku izvršavanja. Zadatak svakog podređenog procesora sastoji se u izvršavanju VNS procedure počev od operacije razmrdavanja u zadatoj okolini. U unapred zadatim stadijumima pretraživanja, dobijeni minimumi šalju se nadređenom procesoru i očekuju od njega dalje instrukcije.

U nastavku ovog odeljka dat je pregled radova u kojima se problem ras-

poređivanja rešava primenom paralelnih metaheurističkih metoda. Obzirom da su GA i TS već paralelizovane za primenu na rešavanje ovog problema, akcenat ovog rada dat je paralelizaciji VNS metode. Tipovi i varijante paralelizacije kao i detalji implementacije opisani su u narednih nekoliko odeljaka, a rezultati raspoređivanja slučajno generisanih grafova zadataka i potencijalne popravke osnovnih varijanti, dati su u poslednjem odeljku ove glave.

Pregled paralelnih metaheuristika za raspoređivanje

Paralelne metaheurističke metode već su primenjivane i na problem statičkog raspoređivanja. Uvidom u postojeću literaturu mogu se izdvojiti sledeći relevantni radovi.

U radu [88] autori su predložili paralelni GA za rešavanje MSPCD. U njihovoj implementaciji, članovi populacije su dopustive permutacije zadataka. Inicijalna populacija dobija se od CP-bazirane permutacije slučajnim perturbacijama. Operator ukrštanja definisan je tako da čuva dopustivost kod rezultujućih permutacija, tj. na isti način kao što je opisano u odeljku 4.7. Mutacija je realizovana zamenom mesta paru susednih nezavisnih zadataka.

Populacija je podeljenja na q delova i svaki je dodeljen jednom procesoru na obrađivanje. Dakle, svaki procesor izvršava operacije ukrštanja i mutacije (sa zadatim verovatnoćama) nad jedinkama svoga dela populacije kao da je to cela populacija. Najbolja jedinka (tekuće najbolje rešenje) razmenjuje se među procesorima svakih $T = N_g/2^n$ iteracija (tj. generacija GA) $n = 1, 2, \ldots$ Verovatnoće mutacije i ukrštanja su adaptivne, tj. vrednosti su im različite na različitim procesorima, a mogu se i menjati u toku izvršavanja paralelnog GA.

Iako deluje vrlo efikasno, ova paralelna metoda ima problem odstupanja na zvaničnim test primerima predloženim od strane samih autora [89] za koje je u [42, 39] pokazano da nisu reprezentativni i da se njihova optimalna rešenja dobijaju efikasnom implementacijom CPES konstruktivne heurističke metode. Sem toga, pokazano je da je sekvencijalna varijanta GA bazirana na istoj reprezentaciji rešenja i na istoj definiciji operatora mutacije i ukrštanja opisana u odeljku 4.7. mnogo efikasnija od paralelne predložene u [88].

Paralelni TS za raspoređivanje zavisnih zadataka na heterogene višeprocesorske sisteme opisan je u [114], dok je njegova detaljna analiza performansi data u [116]. Sa stanovišta problematike koja se razmatra u ovome radu, razlike u radu [114] su drugačija reprezentacija rešenja, heterogeni višeprocesorski sistem (koji se kako je utvrđeno prepiskom sa jednim od

autora sastoji od jednog brzog i $p-1$ sporijeg procesora, odnos brzina definiše se parametrom) i zanemarivanje komunikacionih kašnjenja (što je takođe ustanovljeno u direktnoj prepisci, dok se u radu ne spominje). U [114] predložena je paralelizacija tipa 1, koja se sastoji u tome da se okolina za pretraživanje u okviru LS procedure podeli izmedju paralelnih procesora.

Navedeni radovi prilično su stari, međutim, novija potraga za referencama nije dala zadovoljavajuće rezultate. Postoje radovi koji se bave drugim varijantama problema raspoređivanja, na primer u [7] razmatra se raspoređivanje na mrežu radnih stanica i predlaže paralelizacija kombinacije GA i LS za rešavanje toga problema. Paralelni TS primenjuje se na raspoređivanje sa maksimizacijom toka podataka u radu [14]. U skorašnjoj literaturi se MSPCD uglavnom ne razmatra. Pretpostavlja se da je uzrok tome velika opštost i složenost ove varijante problema raspoređivanja. Rezultati opisani u ovom radu, a publikovani u međunarodnim časopisima i saopšteni na naučnim skupovima, mogli bi da iniciraju nova istraživanja u vezi sa MSPCD.

Nove strategije za paralelizaciju metode promenljivih okolina

Iako relativno nova, VNS metoda je već široko rasprostranjena, a postoje i radovi u kojima se opisuju neke od mogućih strategija za njenu paralelizaciju (videti pregledni članak [104]). U ovom odeljku opisano je nekoliko originalnih strategija za paralelizaciju ove metode.

Osnovni cilj bila je implementacija paralelne verzije VNS metode koja ne narušava (drastično) njenu osnovnu filozofiju: usvajanje prvog poboljšanja (First Improvement, FI). Pri tome cilj je da se proces pretraživanja maksimalno ubrza i zahtevana komunikacija minimizira. FI strategija u VNS metodi [74] podrazumeva da se prekida tok pretraživanja čim se nađe novo tekuće najbolje rešenje. Pretraga se nastavlja u najbližoj okolini tog novog rešenja. Ova strategija bazira se na osobini da su kod većine problema kombinatorne optimizacije (pa tako i kod problema raspoređivanja, kao što se vidi iz odeljka 4.10.) lokalni minimumi blizu jedan drugome.

U ovom radu predloženo je više strategija za paralelizaciju metode promenljivih okolina. Na prvom mestu, paralelizacija LS metode koja se javlja u svakom koraku VNS metode, a zahteva izuzetno mnogo izračunavanja. Zatim, kooperacija različitih varijanti sekvencijalnih VNS metoda, pri čemu su isprobane razne strategije za implementaciju komunikacije i razmene podataka tokom njihovog izvršavanja. Poslednja je paralelna varijanta VNS metode, slične onoj koja je predložena za problem p-medijane u radu [24].

Osnovni cilj paralelizacije bio je da se poboljšaju rezultati dobijeni primenom sekvencijalne VNS metode na MSPCD. Dakle, polazi se od sekvencijalne implementacije VNS metode bazirane na reprezentaciji rešenja pomoću dopustivih permutacija i predlaže se nekoliko strategija za njenu paralelizaciju. Predložene strategije paralelizacije su univerzalne i mogu se primeniti na rešavanje drugih optimizacionih problema, pri čemu se, naravno, mora voditi računa o specifičnostima problema i reprezentacije njegovih rešenja. Uz to, pomenute strategije mogu se primeniti i na druge metaheurističke metode bazirane na lokalnom pretraživanju.

Zadatak paralelizacije je da ubrza izračunavanja, potrebna za rešavanje nekog problema, deleći ih između više procesora. Pri tome se "dobitak" može definisati na dva načina: 1) skraćivanje vremena potrebnog da se dobije neko zadovoljavajuće rešenje ili 2) popravljanje kvaliteta dobijenog rešenja kada se dozvoli da svaki od procesora radi isto vreme kao i sekvencijalna varijanta. Kada se radi o paralelizaciji metaheuristika (koje u sebi uključuju dosta stohastičkih procedura), navedene definicije mogu se kombinovati, što kao krajnji rezultat može imati dvostruku dobit: rezultat boljeg kvaliteta dobija se za kraće vreme.

Analizom VNS metode može se zaključiti da ona pruža dosta prostora za paralelizaciju. Kao prvo, osnovni korak VNS metode, koji se neprekidno ponavlja tokom pretraživanja, je LS procedura. LS procedura je veoma složena, kao što je već opisano u prethodnoj glavi, jer zahteva višestruko izračunavanje vrednosti funkcije cilja (koje je u ovom slučaju polinomne složenosti) u svakoj iteraciji. Stoga se kao najprirodnije mesto za paralelizaciju nameće paralelizacija same LS procedure.

Sledeće očito mesto gde je vrlo jednostavno izvršiti paralelizaciju je kombinacija razmrdavanja u zadatoj okolini i LS koji se izvršava počev od rešenja dobijenog razmrdavanjem. Ovu strategiju uočili su autori rada [58], ali je nisu sproveli do kraja. Ove dve strategije paralelizacije bazirane su na finoj granulaciji, dok bi strategija zasnovana na gruboj granulaciji bila, na primer, simultano izvršavanje više sekvencijalnih VNS procedura za različite vrednosti parametara koji ih definišu. Pri tome treba razlikovati nezavisna izvršavanja (kod kojih se komunikacija odvija samo na kraju, razmenom najboljeg dobijenog rešenja, kao u [58]) od kooperacije procesora (kada se relevantne informacije razmenjuju tokom pretraživanja i utiču na usmeravanje pretrage svake pojedine varijante metode koja učestvuje u kooperaciji).

Prilikom razmatranja problema paralelizacije potrebno je voditi računa o dva osnovna aspekta: izračunavanjima i komunikaciji/sinhronizaciji. Meta-

heuristike spadaju u algoritme koji zahtevaju mnogo izračunavanja i očekuje se da njihova paralelizacija ima pozitivne efekte. Međutim, ukoliko se ne vodi računa o komunikacionom aspektu, može se dogoditi da se pozitivni efekti paralelizacije ponište jer procesori mnogo vremena provode u razmeni međurezultata, a pre svega u čekanju na njih. Za efikasno razmatranje komunikacionog aspekta važni su odgovori na sledeća pitanja [130]: 1) *šta* se prenosi, 2) *kada* se prenosi i 3) *gde* se prenosi.

U ovom radu razmatraju se i sinhrona i asinhrona komunikacija. Na osnovu ranijih iskustava [130], realno je očekivati da se pri sinhronoj implementaciji komunikacije, dobiju paralelne verzije VNS metode, koje su sličnije sekvencijalnoj verziji u smislu redosleda koraka pretraživanja. U slučajevima striktne sinhronizacije razmene podataka među procesorima može se javiti veoma veliki problem. Naime, ukoliko nije postignuto ravnomerno opterećenje među procesorima prilikom podele izračunavanja (a to je u većini slučajeva nemoguće obezbediti), vreme koje procesori provode u čekanju na ostale da završe svoja izračunavanja i uključe se u komunikaciju postaje veliko. Čak i ukoliko se balansira opterećenje procesora, čekanje se može pojaviti kao posledica toga da svi pokušavaju slanje i prijem podataka u isto vreme, a to se ipak mora izvršavati organizovano.

Asinhrona organizacija komunikacije omogućava minimizaciju sačekivanja među procesorima, ali je cena toga nedeterminističko izvršavanje algoritma pretraživanja. To za posledicu ima paralelnu implementaciju koja se veoma razlikuje od originalne, sekvencijalne, varijante metode pretraživanja. Asinhrona komunikacija je ta koja treba da omogući postizanje dvostrukog cilja paralelizacije: poboljšanja kvaliteta rešenja dobijenog za kraće vreme izvršavanja [26].

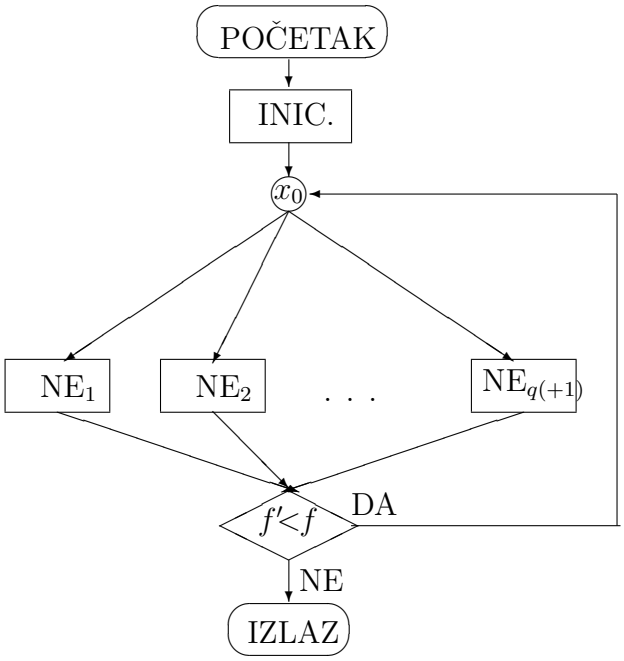
5.2. Paralelizacija LS procedure

Paralelizacija LS procedure može se posmatrati kao problem za sebe i u radu [137] se o tome govori. Ovde će biti reči samo o paralelizaciji lokalnog pretraživanja kao dela nekog šireg optimizacionog algoritma (konkretno VNS metode) ne ulazeći u detalje oko nalaženja najpogodnije strategije za paralelizaciju same LS procedure.

Kao što je prikazano na blok-dijagramu metode promenljivih okolina (videti Sliku 2.6), na početku izvršavanja potrebno je generisati polazno rešenje. Sa aspekta komunikacije, logično je da svaki procesor generiše po-

lazno rešenje za sebe, jer bi inače morao da čeka ne samo da se ono odredi na nekom od procesora nego i da se u organizovanoj komunikaciji prenese do svakog procesora. Sledeći korak VNS metode je popravljanje tog polaznog rešenja, primenom početnog lokalnog pretraživanja. To je prvo mesto gde se LS procedura pojavljuje, a zatim se ponavlja u osnovnoj petlji VNS metode. Stoga je posebna pažnja u ovom odeljku posvećena paralelizaciji LS procedure.

Paralelizam koji obezbeđuje procedura lokalnog pretraživanja predstavlja unutrašnji paralelizam kojim VNS metoda poboljšava svoje karakteristike, bez izmena u sopstvenom algoritmu. Ono o čemu najviše treba voditi računa je da je to paralelizam niskog nivoa koji, po pravilu, zahteva intenzivnu komunikaciju među procesorima. To bi moglo da umanji efekte paralelizacije ukoliko se komunikacije ne planiraju pažljivo.



Sl. 5.1: Paralelizacija LS procedure, PNE varijanta

Najjednostavniji blok-dijagram paralelne LS procedure prikazan je na slici 5.1. Osnovna ideja je da se prostor pretrage (tj. zadata okolina tekućeg rešenja) podeli na nekoliko delova i da se ti delovi distribuiraju procesorima višeprocesorskog sistema. To bi značilo da svaki procesor pretražuje samo

deo koji mu je dodeljen. U opštem slučaju, ova strategija paralelizacije ne menja originalni, sekvencijalni algoritam i namenjena je ostvarenju isključivo prvog cilja paralelizacije – ubrzanju procesa pretraživanja. Strategija je nazvana *paralelno pretraživanje okoline, parallel neighborhood exploration* (PNE) i sastoji se u tome da se u svakoj iteraciji LS procedure tekuća okolina pretražuje paralelno, tako što će različiti procesori istovremeno izvršavati proces pretraživanja u različitim delovima okoline. Svi procesori polaze od istog polaznog rešenja, ali transformišu samo jedan njegov deo (onaj koji se odnosi na deo okoline pridružen tom procesoru). Sa stanovišta izračunavanja, ovde treba očekivati linearno ubrzanje, jer su sva ta izračunavanja nezavisna. Sa stanovišta komunikacija, očekuje se da su one intenzivne na kraju svake iteracije, jer procesori treba da razmene najbolja dobijena rešenja i utvrde koje od njih postaje polazna tačka za sledeću iteraciju LS procedure. Zavisno od broja procesora i brzine veza između njih, razmena rešenja na kraju iteracije mogla bi da predstavlja usko grlo i umanji efekat paralelizacije.

Ovako realizovana PNE strategija spada u kategoriju sinhronih metoda paralelizacije, jer se komunikacije obavljaju u strogo definisanim fazama izvršavanja paralelnog programa i vremenskim trenucima jednakim za sve procesore. Ona se potpuno uklapa u blok-dijagram prikazan na slici 5.1. Međutim, veoma je zanimljivo razmatrati asinhrone implementacije PNE strategije. U najjednostavnijem slučaju to bi značilo da se procesori ne sinhronizuju na "taktove" izračunavanja i komunikacije, već da svaki od njih u trenutku koji njemu najviše odgovara "pošalje" svoje rezultate ostalima, preuzme rezultate koji su do tog trenutka raspoloživi i odluči o daljem izvršavanju na osnovu tih raspoloživih rezultata. Ti rezultati mogu biti kompletni, ali najčešće su delimični.

Motivacija za implementaciju asinhrone varijante potiče od sledećeg. Kako više od jednog procesora izvršava deo procedure lokalnog pretraživanja, na kraju svake iteracije dobija se veći broj parcijalnih lokalnih minimuma, tj. najboljih rešenja u delovima okolina pridruženim različitim procesorima. Usvajanje uvek samo najboljeg rešenja, pokazalo se kao loša strategija [26], pa bi zaista bilo zanimljivo da se pokušava popravljanje trenutno najboljeg rešenja na osnovu više od jednog rešenja iz "populacije" parcijalnih lokalnih minimuma. Najjednostavniji način za to je upravo, gore opisana, implementacija asinhrone varijante PNE (APNE). Na kraju svog izračunavanja vezanog za datu iteraciju, svaki procesor uzima najbolje do tada pronađeno rešenje i nastavlja pretragu u zadatom delu okoline počev od tog najboljeg rešenja. To znači da će u svakom trenutku u igri biti više "dobrih"

rešenja kojima je omogućeno da generišu poboljšanje za tekući (globalni) minimum. Konkretna implementacija ove strategije zavisi od konkretne arhitekture višeprocesorskog sistema na kome će se APNE izvršavati i biće opisana kasnije. Ono što je ovde važno napomenuti je da se kod APN gubi kontrola nad iteracijama lokalnog pretraživanja. Postupak je ipak deterministički i završava se kada ni jedan od procesora nije u mogućnosti da popravi trenutni minimum. Izvršavanje APNE ilustrovano je na slici 5.2.

Sl. 5.2: Asinhrono paralelno pretraživanje okoline (LCPLS)

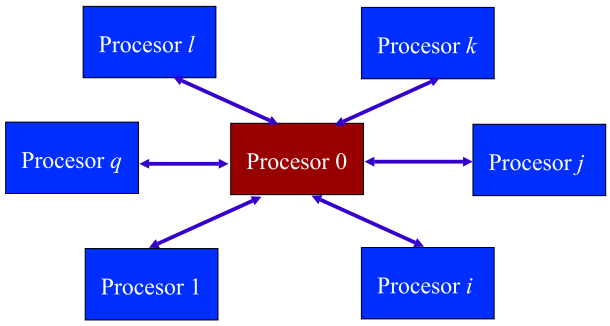
Prethodno navedena strategija asocira na još jedan od mogućih načina paralelizacije LS procedure: sinhrono i asinhrono delimično (parcijalno) lokalno pretraživanje koje je nazvano *ograničeno kooperativno paralelno lokalno pretraživanje, limited cooperative parallel LS* (LCPLS). Ideja ove varijante je da svaki procesor vrši pretraživanje svoga dela okoline sve dok uspeva da popravi sopstveno tekuće rešenje. To znači da se ne zaustavlja na kraju svake iteracije, nego da nastavlja popravljanje svoga novoga rešenja (iz iteracije u iteraciju) dokle god je to moguće. Preciznije, svaki procesor izvršava kompletnu LS proceduru pretpostavljajući da deo okoline koji je njemu dodeljen na obrađivanje predstavlja celu okolinu za pretraživanje. Tek kada se tako definisano delimično lokalno pretraživanje završi, inicira se komunikacija. U sinhronoj implementaciji prikupljaju se svi tako dobijeni (parcijalni) lokalni minimumi i najbolji od njih proglašava se za konačno rešenje. Umesto da se dobijeno rešenje smatra konačnim, moguće je nastaviti paralelno pretraživanje od tog rešenja. Kako je to rešenje lokalni minimum samo u

odnosu na neku podokolinu, može se desiti da, počev od njega, pretraživanje u nekoj drugoj podokolini dovede do poboljšanja.

Važno je primetiti da će u svakoj iteraciji ovako definisanog lokalnog pretraživanja uvek postojati barem jedan procesor koji nije u mogućnosti da popravi tekući delimični lokalni minimum: onaj koji ga je pronašao. Kako se kompletna LS procedura izvršava pre komunikacija, a podela okoline je fiksna, procesor koji je poslao prethodno najbolje rešenje, neće biti u mogućnosti da ga popravi u narednoj iteraciji. Međutim, već u sledećoj, može se pojaviti novo, za koje vredi pokušati pretraživanje u datom delu okoline. Osim toga, postoji nekoliko načina da se uposli procesor koji u datoj iteraciji ne može da popravi tekuće rešenje, ali o tome više reči nešto kasnije.

Asinhrona implementacija za LCPLS slična je asinhronoj implementaciji za PNE, te stoga ovde neće biti detaljno opisivana. Na osnovu ranijih iskustava [26], najbolji rezultati očekuju se upravo kod asinhronih verzija paralelnog lokalnog pretraživanja, tako da je akcenat implementacije na njima.

Implementacija paralelnih verzija izvršena je na homogenoj višeprocesorskoj arhitekturi koja se sastoji od SUN mikroprocesora. Preciznije, pretpostavlja se da se višeprocesorski sistem sastoji od $q+1$ identičnih procesora organizovanih u arhitekturu koju čine jedan nadređeni procesor (master) i q podređenih (slave) procesora (slika 5.3). Za implementaciju komunikacija između procesora, koja se odvija razmenom poruka, korišćena je C biblioteka Message Passing Interface (MPI) komunikacionog protokola [65, 66].



Sl. 5.3: Višeprocesorski sistem za paralelno izvršavanje VNS metode

Procesor 0 namenjen je uglavnom za izvršavanje ulazno/izlaznih instrukcija, komunikaciju sa podređenim procesorima i koordinaciju njihovog rada, kao i za čuvanje i obnavljanje baze svih podataka relevantnih za korektno

i efikasno paralelno izvršavanje VNS metode. U pojedinim slučajevima, moguće je ovom procesoru dodeliti i neka konkretna izračunavanja. Podređeni procesori namenjeni su za intenzivna izračunavanja (lokalno pretraživanje uglavnom), koja su sastavni deo uspešnog izvršavanja VNS algoritma. Veoma je važno da se izračunavanja pravilno rasporede između procesora, ali je takođe neophodno voditi računa da se minimizira vreme čekanja uzrokovano razmenom podataka.

Kao što se vidi sa slike 5.3 i prethodnog opisa, komunikacija među procesorima je indirektna, tj. svaki podređeni procesor komunicira samo sa nadređenim procesorom (procesorom 0), i ne razmenjuju poruke između sebe. Takav vid komunikacije efikasniji je nego da svaki od procesora šalje svoje rezultate svim preostalim procesorima, prima od njih relevantne podatke, vrši poređenje i donošenje odluke na osnovu toga. Minimalna količina podataka šalje se nadređenom procesoru, on vrši analizu dobijenih rezultata, traži dodatne informacije od odgovarajućeg procesora i zatim ih prosleđuje svima preostalima. Kako se to konkretno realizuje i šta su relevantne informacije zavisi od konkretne varijante paralelne VNS metode koja se implementira.

Obzirom na poznatu činjenicu da je komunikacija razmenom poruka izuzetno spora, veoma je važno da se minimizira količina podataka koja se prenosi između procesora. To se može postići kako pravilnim izborom tipa podataka, tako i pažljivim odabirom podataka koji se šalju. Na primer, ne treba slati celu permutaciju koja predstavlja rešenje MSPCD, ukoliko nije došlo do popravke. Pa čak i ako je neki procesor ostvario poboljšanje, dovoljno je da pošalje novu vrednost funkcije cilja, a samo rešenje (permutaciju) tek po zahtevu (ukoliko je njegovo rešenje zaista novo najbolje).

Za situacije u kojima se forsira asinhrona komunikacija, pogodniji su višeprocesorski sistemi sa deljenom memorijom, kada svaki procesor može da pristupi svakoj memorijskoj lokaciji. Međutim, na osnovu uputstva za korišćenje komunikacionih rutina MPI biblioteke baziranih na razmeni poruka [65, 66], zaključuje se da je asinhrona komunikacija lako izvodljiva upotrebom neblokirajućih funkcija iz biblioteke.

Obzirom na to da je minimizacija vremena komunikacije veoma važna za efikasno paralelno izvršavanje VNS metode (kao i svake druge), neophodno je odrediti koliko se vremena tokom izvršavanja troši na komunikaciju. Na osnovu toga, lako je zaključiti koliko često i koju količinu podataka treba razmenjivati. Najprirodniji način da se izmeri vreme komunikacije je da svaki procesor meri vreme za sebe. Meri se ukupno vreme izvršavanja (t_{tot}), tj. vreme koje protekne između prijema dve ključne poruke: početnog rešenja

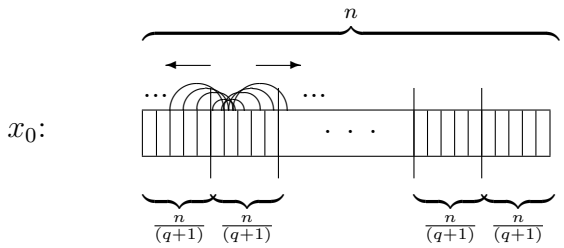
i STOP komande za zaustavljanje izvršavanja. Osim toga, mere se i vremenski intervali između dveju komunikacija. Preciznije, svaki put pre nego inicira komunikaciju, procesor zaustavi merenje vremena i nastavi ga nakon što je dobio željene podatke. Dakle, merenje vremena zaustavlja se za vreme izvršavanja MPI_Send i MPI_Recv komandi [65, 66], a tako dobijena vremena izračunavanja se sabiraju. Ova suma predstavlja vreme izračunavanja (t_{CPU}). Vreme komunikacije dobija se iz razlike $t_{tot} - t_{CPU}$. Ovo vreme uključuje, kako fizički prenos podataka između procesora, tako i vreme čekanja (sinhronizacije) na slanje/prijem poruka.

Imajući sve to u vidu, izvršene su implementacije dva opisana tipa paralelne LS procedure: sinhrona varijanta PNE i asinhroni LCPLS. Na početku svakog procesa raspoređivanja, potrebno je generisati početno rešenje x_0. U sekvencijalnoj implementaciji, početno rešenje (početna dopustiva permutacija) određivano je primenom CPES konstruktivne heuristike. Ono se koristi i u paralelnoj implementaciji, pri čemu ga svaki procesor određuje za sebe jer se time smanjuje vreme čekanja i količina komunikacije.

Ne smanjujući opštost, objašnjenja se mogu bazirati na Swap-1 okolini (za ostale tipove okolina, implementacija je analogna). Osnovna ideja je da se dopustiva permutacija n zadataka podeli na delove (slika 5.4) i da se svi delovi obrađuju simultano od strane različitih procesora. Podela nije fizička, tj. svim procesorima je uvek na raspolaganju cela permutacija kako bi mogli da proveravaju dopustivost transformisanih rešenja. Podela označava samo regione u okviru kojih će svaki od procesora uzimati zadatke za premeštanje.

Na koliko regiona će se podeliti tekuća permutacija zavisi od broja procesora koji su uključeni u paralelno izvršavanje LS procedure. Pri tome se pretpostavlja da je broj procesora (znatno) manji od broja zadataka u grafu, tj. $q + 1 < n$. Na osnovu ranije izloženog, očito je da postoje dva moguća slučaja: 1) svi procesori izvršavaju po deo paralelne LS procedure i 2) nadređeni procesor nije uključen u izvršavanje lokalnog pretraživanja, tj. izvršava samo ulazno/izlazne operacije, komunikacione rutine i ažuriranje relevantnih parametara.

Najjednostavnija podela permutacije za paralelno pretraživanje je podela na jednake delove (kao što je to prikazano na slici 5.4). Za izvršavanje procedure lokalnog pretraživanja paralelno na $q+1$ procesora, okolina Swap-1 deli se između procesora na sledeći način: svaki procesor detaljno pretražuje svoj deo okoline premeštajući $[n/(q+1)]$ susednih zadataka koji se nalaze u delu permutacije pridruženom tom procesoru. Deo permutacije koji pretražuje procesor r ($r = 0, 1, \ldots, q$), zavisno od strategije pretraživanja (FI ili BI),



Sl. 5.4: Paralelna implementacija PNE za Swap-1 okolinu

predstavljaju svi zadaci u jednom od sledećih intervala

$$\left[r * \left\lceil \frac{n}{q+1} \right\rceil + 1, (r+1) * \left\lceil \frac{n}{q+1} \right\rceil\right],$$

$$\left[(q-r) * \left\lceil \frac{n}{q+1} \right\rceil + 1, (q-r+1) * \left\lceil \frac{n}{q+1} \right\rceil\right],$$

Ovde $\lceil x \rceil$ označava najmanji ceo broj veći ili jednak od x.

Razlike u definiciji intervala za različite strategije pretraživanja posledica su iskustva stečenog eksperimentisanjem sa sekvencijalnom implementacijom opisanom u glavi 4. Kombinovanjem FI-BI sa FOR-BACK strategijama utvrđeno je da se najveći broj popravki tekućeg rešenja dobija u početnom delu permutacije. Obzirom na to da nadređeni procesor izvršava i LS proceduru i komunikaciju sa ostalima i sa korisnikom i upravljanje tokom izvršavanja čitave paralelne procedure, bitno je da on što pre završi proceduru pretraživanja i bude spreman da prihvati i obradi rezultate rada ostalih procesora. Stoga mu se pri FI pretraživanju dodeljuje početni deo permutacije. S druge strane, ako se primenjuje BI strategija, svaki procesor treba detaljno da pretraži svoj deo okoline (bez obzira na poboljšanja) i stoga se nadređenom procesoru dodeljuje poslednji deo koji je po pravilu najmanji jer je u većini slučajeva $n\%(q+1) \neq 0$, tj. $q+1$ ne deli n. Osim toga, ažuriranje vrednosti funkcije cilja je u tom delu jednostavno, jer je promenjen samo mali deo permutacije te stoga treba preraspodeliti samo nekoliko zadataka (videti odeljak 4.3.).

U slučaju kada procesor 0 (nadređeni procesor) ne učestvuje u paralelnom izvršavanju LS procedure (što je izuzetno značajno za implementaciju

LCPLS), deo intervala koji obrađuje procesor r ($r = 1, 2, \ldots, q$) određuje se na sličan način, s tim što je ukupni broj delova samo q.

Procedura pretraživanja koju izvršava svaki od procesora sastoji se u premeštanju zadataka odgovarajućeg dela Swap-1 okoline na sve dopustive pozicije u permutaciji. Rezultujuće pozicije nisu ograničene samo na deo okoline pridružen datom procesoru, već mogu pripadati bilo kom delu ostatka permutacije (kao što je prikazano na slici 5.4). Parametri pretrazivanja (FI-BI, FOR-BACK) definišu se analoglno kao u sekvencijalnom slučaju, ali za svaki interval pretraživanja pojedinačno. Moguće je dozvoliti i preklapanja dobijena pretraživanjem svih suseda, obzirom na to da je izvršavanje simultano. Potencijalne prednosti i nedostaci oba pristupa moraju se proveriti eksperimentalno.

Što se tiče strategije poboljšanja, BI LS u slučaju paralelnog izvršavanja isto je što i u sekvencijalnom, jer se u oba slučaja obilaze svi susedi i usvaja se najbolje rešenje (označava se sa BOB, Best Of Bests). FI LS u paralelnom izvršavanju može da generiše različit lokalni minimum, u odnosu na sekvencijalnu varijantu, jer se dobija $q(+1)$ FI rešenja koja nadređeni procesor upoređuje i od njih bira ono koje će se usvojiti za polaznu tačku nastavka pretraživanja. Pravilo izbora može biti najbolje od prvih - "best of firsts" (BOF) ili prvo od prvih - "first of firsts" (FOF). Implementirana su i testirana oba, a treba napomenuti da je FOF nederminističko pravilo, jer se rešenja dobijena višestrukim paralelnim izvršavanjem mogu međusobno razlikovati (zavisno od implementacije komunikacionih rutina).

Sinhrono izvršavanje PNE sastoji se u paralelnom izvršavanju LS procedure sa zadatim vrednostima parametara (FI-BI, FOR-BACK), iteraciju po iteraciju dokle god postoji poboljšanje. Svaki procesor pretražuje deo okoline koji mu je pridružen, a zatim inicira komunikaciju radi razmene međurezultata i donošenja odluke o nastavku izvršavanja. Podređeni procesori šalju nadređenom svoje rezultate i očekuju dalje instrukcije. Šalje se samo vrednost funkcije cilja koja odgovara minimumu u odgovarajućem delu okoline. Uloga nadređenog procesora je da (nakon svoga dela pretrage) prihvati sve vrednosti od podređenih procesora, uporedi ih i odredi najbolju (na osnovu jednog od pravila: BOB, BOF, FOF) i uporedi tu vrednost sa trenutnim minimumom. Ako je to novi minimum, traži od odgovarajućeg procesora rešenje (permutaciju) koje odgovara tom minimumu i prosleđuje ga ostalim procesorima. Ukoliko nije došlo do poboljšanja tekućeg najboljeg rešenja, nadređeni procesor šalje poruku zaustavljanja (STOP message) ostalima i izveštava korisnika o pronađenom lokalnom minimumu.

Kao što se iz prethodnog pasusa vidi, vodilo se računa o minimizaciji komunikacije među procesorima. Rešenja koja odgovaraju parcijalnim lokalnim minimumima (permutacije n zadataka) šalju se samo ukoliko su neophodna, tj. u slučaju poboljšanja tekućeg lokalnog minimuma i to samo jedno, a ne svih q.

U slučaju kada nadređeni procesor izvršava i deo paralelne LS procedure, sinhrona implementacija PNE pokazuje se efikasnijom. Ako nadređeni procesor obavlja samo komunikacioni i upravljački deo posla, paralelni PNE može se implementirati i sinhrono i asinhrono. Sinhrona implementacija je identična kao u prethodnom slučaju samo se okolina deli na manji broj većih intervala. Sem toga, očekivano ubrzanje je manje jer je manji broj procesora koji se koriste za lokalno pretraživanje. Stoga ova varijanta nije ni testirana.

Kao što je na osnovu ranijih iskustava sugerisano, implementirana je asinhrona varijanta LCPLS. Osnovna ideja je u sledećem: podređeni procesori izvršavaju pretraživanje svoga dela okoline sve dok su u mogućnosti da poprave tekuće najbolje rešenje. Dakle, ne prijavljuje se svako poboljšanje, nego samo ono konačno, tj. lokalni minimum u odnosu na datu podokolinu. To u suštini znači da svaki procesor izvršava ne samo pretraživanje svoga dela okoline, nego kompletnu proceduru lokalnog pretraživanja u tom delu okoline (smatrajući da je taj deo cela okolina za pretraživanje). Time se znatno smanjuje broj obraćanja nadređenom procesoru, samim tim i vreme čekanja procesora, povećava efikasnost pretraživanja jer se veliki broj potencijalno dobrih rešenja uzima u obzir za pretraživanje i popravljanje.

Kada se novo rešenje pošalje nadređenom procesoru mogu se desiti tri slučaja. Ukoliko je to novo najbolje rešenje, odgovarajućem procesoru šalje se poruka da pređe u stanje čekanja (HOLD), a procesorima koji su već u stanju čekanja (ukoliko ih ima) prosleđuje se to novo rešenje. Drugi slučaj je da nadređeni procesor već poseduje bolje rešenje (pronađeno od strane nekog drugog procesora) koje će biti prosleđeno datom procesoru da od njega započne lokalno pretraživanje u svome delu okoline. Treći slučaj nastupa kada je dobijeno rešenje isto kao postojeće najbolje. To znači da nije došlo do poboljšanja i da treba proveriti koliko se procesora nalazi u stanju čekanja. Ukoliko postoje procesori koji još rade, dati procesor se šalje u stanje čekanja, uvećava se brojač neuspešnih pokušaja i potraga za boljim rešenjem se nastavlja (sa smanjenim brojem procesora). Ukoliko je dati procesor poslednji koji izveštava o svom neuspehu, lokalni minimum je pronađen, podređenim procesorima se šalje STOP poruka, korisnik se izveštava o konačnom rezultatu i završava se procedura paralelnog lokalnog pretraživanja.

Kod implementacije LCPLS veoma je važno naglasiti da se gubi smisao iteracija lokalnog pretraživanja. Svaki procesor može izvršiti različit broj iteracija, pre nego što pronađe parcijalni lokalnio minimum i inicira komunikaciju sa nadređenim procesorom. Druga važna napomena vezana je za intervale čekanja. Obzirom na organizaciju pretraživanja, intervali čekanja su drugačije raspoređeni i veoma je značajno utvrditi da li je vreme čekanja smanjeno asinhronom implementacijom.

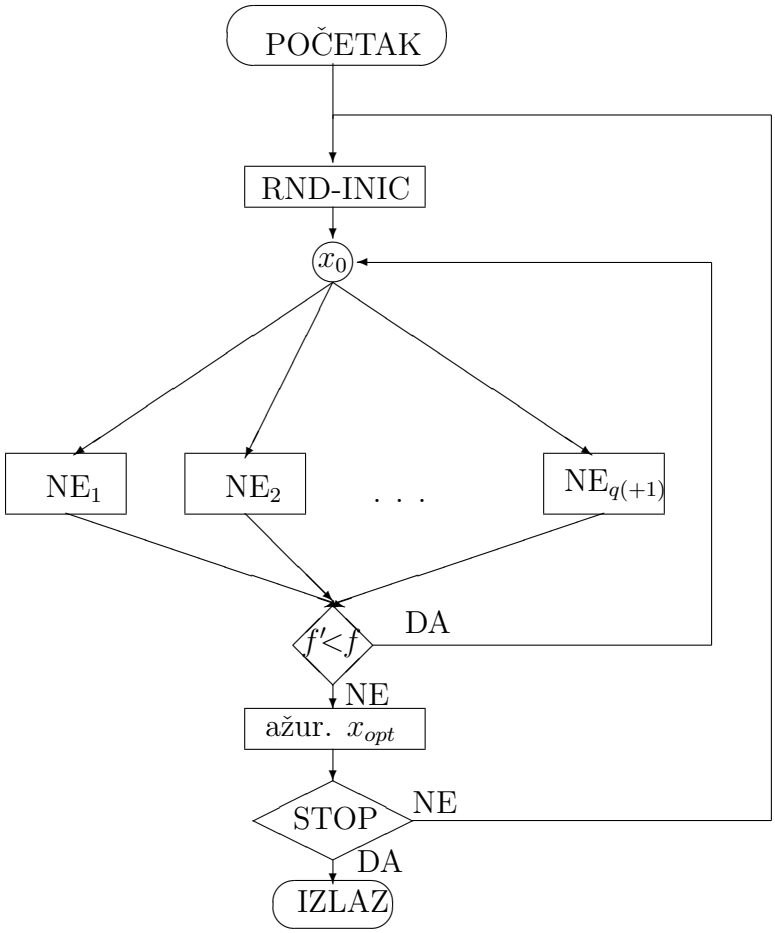
Asinhrona implementacija može se realizovati sa još više nedeterminizma u izvršavanju, tj. slobode dopuštene podređenim procesorima ukoliko se u komunikaciji koriste neblokirajuće rutine. Preciznije, sa stanovišta podređenih procesora, nadređeni procesor posmatra se kao globalna memorija. Svaku promenu koja mu je saopštena on notira, a podređeni procesori je koriste onako kako njima najviše odgovara. To znači sledeće: nakon slanja svoga rezultata nadređenom procesoru, podređeni ne čeka da njegova poruka bude obrađena, nego odmah uzima poslednju poruku koja bi trebalo da odslikava trenutno stanje izvršavanja (tj. stanje globalne memorije bazirano na do tada obrađenim porukama) i na osnovu toga sam donosi odluku o daljem toku svog izvršavanja. Dakle, poređenje rešenja vrši se i na nivou podređenih procesora. Razlika između ove implementacije i sistematske je da se neće uvek koristiti samo najbolje rešenje, već će se uzimati u obzir i neka rešenja koja predstavljaju poboljšanja, ali ne najbolja moguća. Ovo je ključna dobit prilikom paralelnog izvršavanja, jer takva pretraživanja mogu da dovedu do kvalitetnijih konačnih rezultata. Sva "najbolja" rešenja će sigurno biti upotrebljena, ali će se pored njih, još nekim "dobrim" rešenjima pružiti šansa, što svakako povećava verovatnoću popravljanja konačnog rešenja.

5.3. Paralelno višestartno lokalno pretraživanje

Kako se MLS pokazala prilično efikasnom u sekvencijalnoj implementaciji, veoma je zanimljivo videti da li paralelizacija očuvava dobre osobine višestartnog pretraživanja. Osim toga, PMLS je dobra baza za poređenje sa paralelnom metodom promenljivih okolina, koja je glavna tema u ovom delu disertacije.

Nakon što se završi paralelna LS procedura, dobijeno najbolje rešenje deklariše se kao globalni minimum x_{opt} i postaje početno rešenje za paralelne metaheurističke procedure. Svaka od opisanih paralelnih LS procedura može se koristiti prilikom paralelizacije metaheurističkih metoda baziranih

na lokalnom pretraživanju. Na primer, PNE može predstavljati bazu za implementaciju paralelne procedure višestartnog lokalnog pretraživanja (Parallel Multistart Local Search, PMLS). Blok dijagram ove procedure dat je na slici 5.5. Ukoliko se koristi LCPLS, blok dijagram više ne važi, ali je implementacija analogna.



Sl. 5.5: Paralelizacija MLS procedure

Realizacija PMLS je izuzetno jednostavna, potrebno je samo obezbediti petlju koja počinje generisanjem slučajnog početnog rešenja, nastavlja se paralelnim izvršavanjem LS procedure, nakon čega sledi ažuriranje globalno najboljeg rešenja i provera kriterijuma zaustavljanja. Početno rešenje generiše se na svim procesorima, kao i kod paralelne LS procedure opisane u

prethodnom odeljku, s tim što se ne koristi deterministička CPES metoda, već slučajni generator dopustivih permutacija. Na svim procesorima koristi se ista baza generatora slučajnih brojeva, čime se obezbeđuje dobijanje istog početnog rešenja.

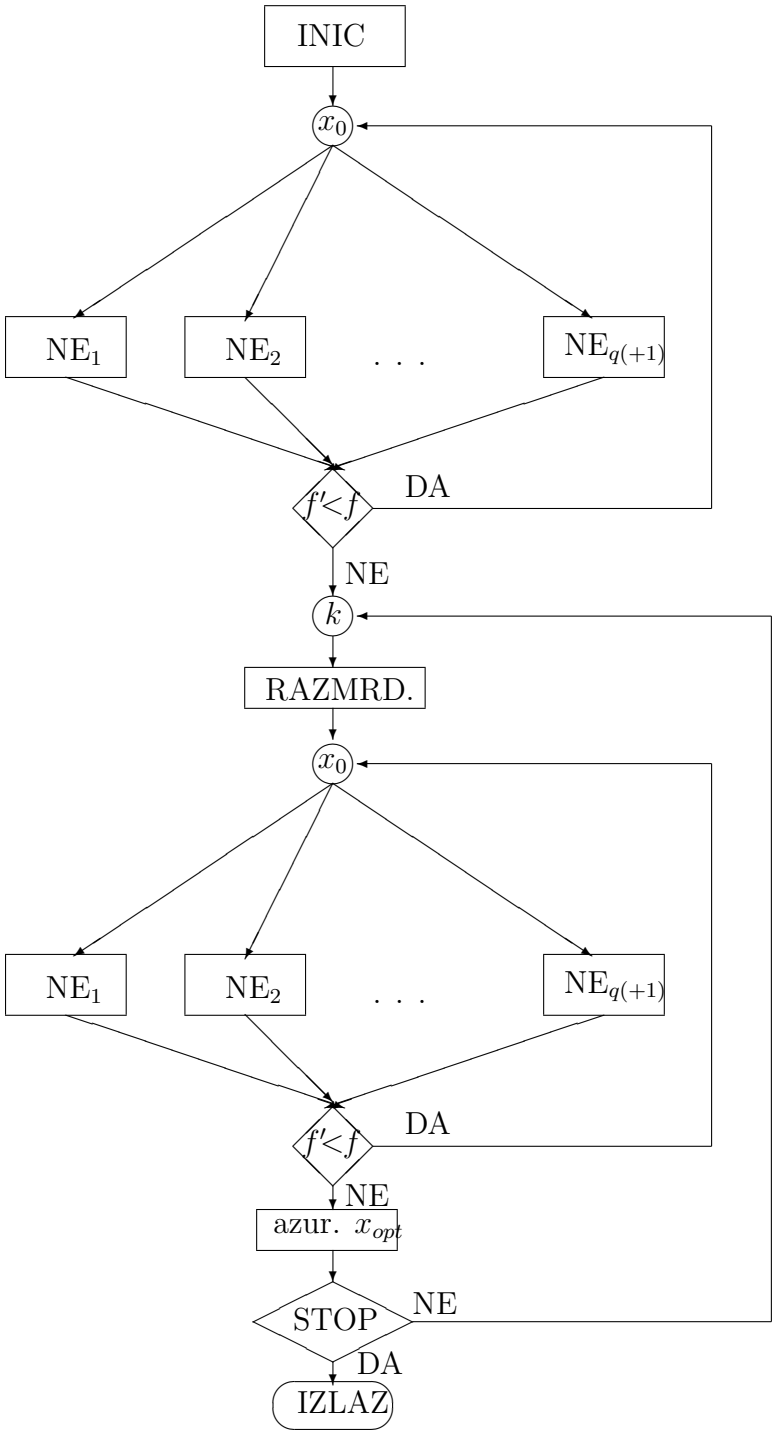
Osim tekućeg lokalno najboljeg rešenja, nadređeni procesor treba da čuva i ažurira i globalno najbolje rešenje, tj. najbolje od svih (paralelnom LS procedurom dobijenih) lokalnih minimuma. Dakle, na kraju lokalnog pretraživanja (ma koja paralelna varijanta se izvršavala), nadređeni procesor treba da uporedi dobijeni lokalni minimum sa tekućim globalno najboljim rešenjem i da sačuva novo najbolje. Tek nakon toga počinje novo izvršavanje paralelne LS procedure na svim procesorima.

Ovde je važno uočiti osnovnu razliku između PLS i PMLS izvršavanja. Umesto STOP poruke, pri paralelnom izvršavanju MLS procedure na kraju lokalnog pretraživanja šalje se END poruka (koja označava kraj LS procedure u odnosu na dato početno rešenje). Poruka STOP, u ovom slučaju, šalje se u trenutku kada je zadovoljen kriterijum zaustavljanja MLS metode, a to je uglavnom dozvoljeno vreme izvršavanja (max CPU time). Zavisno od implementacije, STOP poruka može se slati isključivo nakon END poruke ili u bilo kom trenutku izvršavanja. Obzirom da je nepraktično često proveravati zadovoljivost kriterijuma zaustavljanja, najčešće se on proverava na kraju lokalnog pretraživanja (što odgovara END-STOP redosledu poruka) ili u nekim precizno određenim situacijama (kao što bi bio kraj iteracije lokalnog pretraživanja, tamo gde se o njemu vodi računa ili novo poboljšanje lokalnog minimuma koje za sobom povlači komunikaciju sa svim procesorima).

5.4. VNS sa paralelnom LS procedurom

Najjednostavniji način paralelizacije VNS metode je da se u okviru VNS algoritma LS procedura izvršava paralelno. Na primer, slika 5.6 ilustruje blok šemu VNS metode kod koje je lokalno pretraživanje zamenjeno sa PNE. U odnosu na paralelizaciju lokalnog pretraživanja koja je opisana u odeljku 5.2., potrebno je dodati jednu petlju koja sadrži uvećanje brojača okoline, operaciju razmrdavanja u tekućoj okolini, paralelni LS, ažuriranje globalnog minimuma i proveru kriterijuma zaustavljanja. Implementacija je vrlo slična implementaciji paralelne MLS procedure.

Operacija razmrdavanja može se izvršavati bilo sekvencijalno, bilo paralelno. Prilikom implementacije treba izabrati bolju strategiju zavisno od brzine



Sl. 5.6: VNS sa paralelnim lokalnim pretraživanjem

komunikacije i zahtevane sinhronizacije među procesorima. Ažuriranje globalnog minimuma vrši se na isti način kao i kod same paralelne LS procedure, samo je u pitanju zaista novo, tekuće najbolje rešenje u odnosu na koje se izvršava operacija razmrdavanja. Kriterijum zaustavljanja predstavlja vreme izvršavanja paralelne metode i ono se takođe može proveravati lokalno (na svakom procesoru) ili na globalnom nivou (gde se izdvaja jedan procesor zadužen za komunikaciju sa korisnikom i čuvanje i ažuriranje svih globalnih informacija).

U skladu sa blok-dijagramom VNS procedure prikazanim na slici 5.6, nakon što se završi početno lokalno pretraživanje, svaki procesor ulazi u petlju. U okviru te petlje, prvo se poveća brojač okoline (ili resetuje na 1 ukoliko je dostigao maksimalnu vrednost i time započinje nova iteracija VNS pretraživanja). Zatim se izvršava operacija razmrdavanja u k-toj okolini paralelno, tj. istovremeno na svakom od procesora, kako bi se uštedelo na komunikaciji. Za određivanje slučajnog rešenja u k-toj okolini tekućeg najboljeg rešenja koristi se ista baza generatora slučajnih brojeva, čime se obezbeđuje da je polazno rešenje lokalnog pretraživanja na svim procesorima isto.

Osim opisane realizacije operacije razmrdavanja, može se implementirati i stvarna paralelna varijanta. Ona bi se sastojala u tome da svaki procesor generiše različito rešenje u k-toj okolini (koristeći različitu bazu generatora slučajnih brojeva) i da se onda među njima po nekom pravilu izabere jedno koje predstavlja početnu tačku za paralelni LS. Pravila mogu biti različita, mada je uobičajeno da se uzima najbolje rešenje dobijeno razmrdavanjem. Ovakva implementacija ustvari predstavlja paralelnu verziju poznate varijante operatora razmrdavanja definisanog u [74] sa sopstvenim parametrom b-broj slučajnih rešenja koja se generišu u svakoj okolini. Za odgovarajuću paralelnu implementaciju važilo bi $b = q(+1)$. Obzirom da je sekvencijalna implementacija predložena u [45] i opisana u prethodnoj glavi pokazala da ova varijana daje loše rezultate, u paralelnoj implementaciji neće biti razmatrana.

Kada se odredi rešenje dobijeno operacijom razmrdavanja, započinje paralelno lokalno pretraživanje: svaki procesor pretražuje deo okoline u potrazi za poboljšanjem ne samo tog rešenja, nego i trenutno globalno najboljeg. Proces paralelnog lokalnog pretraživanja kontroliše nadređeni procesor na isti način kao kod implementacije prostog paralelnog lokalnog pretraživanja. opisanog ranije. Nakon završetka procedure paralelnog lokalnog pretraživanja, nadređeni procesor vrši poređenje novog lokalnog minimuma sa tekućim globalnim minimumom (x_{opt}) i ažurira njegovu vrednost i vred-

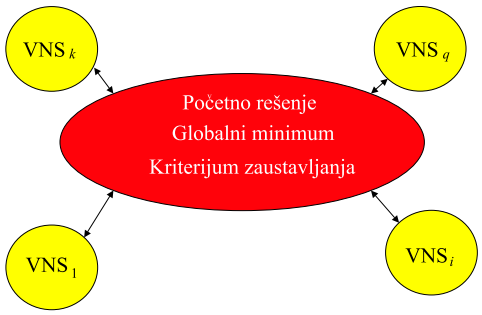
nost brojača okolina u skladu sa pravilima VNS metode. Pre nastavka izvršavanja, proverava se i da li je kriterijum zaustavljanja zadovoljen, tj. da li je prekoračeno maksimalno dozvoljeno vreme izvršavanja. Ukoliko kriterijum zaustavljanja nije zadovoljen, tekući indeks okoline i globalno najbolje rešenje prosleđuju se podređenim procesorima i nastavlja se pretraživanje po koracima VNS metode. Ukoliko je zadovoljen kriterijum zaustavljanja, podređenim procesorima se šalje STOP poruka, a korisnik se izveštava o najboljem pronađenom rešenju i ostalim relevantnim rezultatima pretraživanja.

Izlazni kriterijum definisan kao maksimalno dozvoljeno vreme izvršavanja (max CPU time) omogućava pošteno poređenje raznih varijanti paralelne VNS metode, kao i poređenje VNS metode sa drugim paralelnim metaheuristikama (PMLS, na primer). Vremenski kriterijum omogućava poređenje rezultata izvršavanja za različit broj procesora, tj. utvrđivanje faktora ubrzanja i eventualnog poboljšanja kvaliteta rešenja zavisno od broja procesora $q(+1)$ koji učestvuju u paralelnom izvršavanju VNS metode. Naravno, važno je da se dozvoljeno vreme pravilno zadaje, tj. da se smanjuje proporcionalno povećanju broja procesora.

5.5. Kooperativni rad različitih VNS metoda

Drugi način paralelizacije VNS metode je kooperacija više različitih (sekvencijalnih) VNS procedura koje se izvršavaju paralelno (istovremeno) na različitim procesorima. Ovako dobijena paralelna VNS metoda nazvana je *Cooperative VNS* (CVNS). Suština ideje kooperativnog rada je da se na raznim procesorima izvršavaju razne varijante VNS metode (na primer, neke od onih opisanih u poglavlju 4.) koje u unapred zadatim situacijama razmenjuju međurezultate relevantne da nastavak izvršavanja (slika 5.7).

Komunikacije među procesorima (tj. različitim varijantama VNS metode) treba pažljivo isplanirati, što podrazumeva da se precizno definiše šta, kome i kada se prenosi. Uobičajeno je da se prenosi tekuće najbolje rešenje, mada je moguće dostavljati i druge relevantne informacije, kao što je na primer, rešenje dobijeno razmrdavanjem koje može da predstavlja diversifikaciju pretraživanja za druge varijante VNS metoda koje učestvuju u kooperaciji. Obzirom na usvojenu strukturu veza među procesorima (podređeni-nadređeni procesor) na kojima se izvršavaju paralelne varijante metode promenljivih okolina (pa i CVNS), relevantni podaci šalju se uvek nadređenom procesoru i od njega primaju podaci i instrukcije za dalji rad.



Sl. 5.7: Kooperativni rad $q(+1)$ VNS procedura

Moguće situacije u kojima se vrši razmena relevantnih podataka su:

1. Na kraju prvog izvršavanja LS procedure, tj. onog koje počinje od inicijalnog rešenja. To je trenutak kada se dobija potencijalno novi tekući globalni minimum. Kako su i same LS procedure koje se izvršavaju na različitim procesorima međusobno različite, očekivano je da i dobijeni lokalni minimumi budu različiti, te je stoga razmena podataka na ovom mestu gotovo neizbežna.

2. U momentu kada se dobije novo tekuće najbolje rešenje, o tome treba izvestiti ostale procesore. Pri tome je moguće da je to rešenje najbolje samo za dati procesor, a da je neki drugi već pronašao bolje. Stoga je komunikacija u takvim trenucima od izuzetnog značaja. CVNS će postaviti okolinu za razmrdavanje na $k = 1$, ali u odnosu na trenutno zaista najbolje rešenje dobijeno od svih VNS procedura koje učestvuju u kooperaciji.

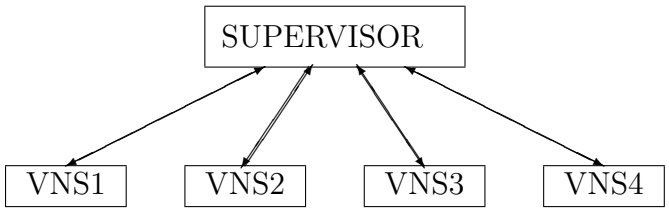
3. U momentu kada se menja okolina za razmrdavanje (kada se zahteva uvećanje indeksa okoline, tj. kada se izvršava $k++$ operacija). Ova situacija podrazumeva da nije došlo do poboljšanja globalno najboljeg rešenja pretraživanjem u tekućoj okolini. Komunikacija je intenzivna, ali doprinosi očuvanju FI filozofije VNS metode.

4. Kada indeks okoline dostigne maksimalnu vrednost $k = k_{max}$, što znači

da sekvencijalna VNS procedura počinje novu iteraciju bez poboljšanja tekućeg globalnog minimuma. U tom trenutku korisno je proveriti da li je neki drugi procesor možda našao bolje rešenje. To znači da CVNS takođe kreće od $k = 1$, ali od potencijalno novog tekućeg najboljeg rešenja.

Komunikacije, kao i u prethodnom slučaju, mogu biti sinhronizovane ili asinhrone, ali realno je očekivati da je asinhrona implementacija efikasnija [26]. Obzirom na to da su LS procedure na različitim procesorima različite, kao i operacije razmrdavanja, vrlo je verovatno da je njihova dužina izvršavanja različita. Sinhronizovanje procesora u odgovarajućim komunikacionim tačkama dovelo bi do dugačkih intervala čekanja na onim procesorima koji su svoja pretraživanja završili (na procesor koji zahteva najviše vremena za izvršavanje LS procedure i/ili razmrdavanja). Imajući sve to u vidu, implementirana je samo asinhrona varijanta.

Jedan primer kooperativnog rada četiri jednostavne varijante VNS metode prikazan je na slici 5.8. Jasno je da ne postoje prepreke za angažovanje i više od pet procesora i da se u kooperaciju može uključiti bilo koja varijanta VNS metode opisana u glavi 4. kao i bilo koja druga.



VNS1	VNS2	VNS3	VNS4
LS: 1-Swap	LS: 1-Swap	LS: IntCh	LS: IntCh
Sh: k-Swap	Sh: k-IntCh	Sh: k-Swap	Sh: k-IntCh
FORWARD	BACKWARD	BACKWARD	FORWARD

Sl. 5.8: Primer kooperativnog rada VNS metoda

Kao i kod ostalih paralelnih verzija, inicijalno rešenje x_0 određuje se pomoću CPES heuristike na svakom od procesora. Polazeći od toga rešenja, svaki od procesora izvršava svoju varijantu procedure lokalnog pretraživanja. Tako dobijeni lokalni minimum šalje se (asinhrono) nadređenom procesoru, i od njega dobija trenutno najbolje rešenje (globalni minimum) od kojega

započinje VNS petlju. U toku izvršavanja VNS procedure, svaki podređeni procesor nalazi se u petlji u okviru koje obavlja sledeće osnovne korake:

1. Proverava da li je poslata poruka za zaustavljanje;
2. Komunicira sa nadređenim procesorom radi razmene najboljeg rezultata;
3. Izvršava odgovarajuću varijantu VNS procedure do naredne komunikacione tačke u skladu sa informacijama dobijenim od nadređenog procesora.

Nadređeni procesor u petlji izvršava sledeće korake:

1. Proverava kriterijum zaustavljanja i vsalje odgovarajuću poruku svim podređenim procesorima ukoliko je zadovoljen;
2. Prihvata novu poruku od nekog podređenog procesora;
3. Ukoliko je to novo najbolje rešenje, šalje ga svim podređenim procesorima;
4. Ažurira bazu dobrih rešenja i ostalih relevantnih podataka.

Asinhrona komunikacija realizovana je pomoću neblokirajućih komunikacionih rutina MPI biblioteke [65, 66]. Ove rutine omogućavaju da se procesor koji šalje poruku ne blokira operacijom komunikacije (do trenutka dok prijemni procesor ne prihvati poslatu poruku). Naprotiv, pošiljalac samo "ubaci poruku u poštansko sanduče primaoca" i nastavlja dalje izvršavanje. Primalac će poruku pročitati kada njemu to bude odgovaralo. Postoje i odgovarajuće neblokirajuće rutine za prijem poruke, koje se mogu interpretirati kao "provera da li ima poruke u poštanskom sandučetu". Naravno, ukoliko je nastavak izvršavanja nemoguć bez informacije koju sadrži poruka, prijemni procesor mora da se blokira na operaciji čitanja do momenta stizanja poruke. "Poštansko sanduče" predstavlja u stvari jedan (ograničeni) deo memorije, rezervisan za smeštanje poruka. Stoga je važno da se pri asinhronoj komunikaciji ima na umu upravo ograničenost tog prostora, tj. mogućnost da neke od poruka budu izgubljene.

U konkretnoj implementaciji asinhronog kooperativnog rada VNS procedura za raspoređivanje, komunikacije su realizovane imajući na umu sve te

činjenice. Za podređene procesore od značaja je samo poslednja pristigla poruka: svaka poruka nadređenog procesora sadrži novo najbolje rešenje, dakle svako novo rešenje je bolje od prethodnih, pa od poslednjeg treba nastaviti pretragu. Druga mogućnost je da je u pitanju poruka zaustavljanja, a to je ionako poslednja poruka koju nadređeni procesor šalje. Dakle, eventualni gubitak nekih poruka u ovom slučaju nije od značaja.

Za nadređeni procesor, sve poruke su jednako važne jer je nemoguće proceniti koji će procesor i u kojoj fazi izvršavanja VNS procedure pronaći najbolje rešenje. Stoga se minimizira broj poruka koje se šalju nadređenom procesoru kako bi se smanjio broj eventualno izgubljenih poruka na najmanju moguću meru. Naime, svaki podređeni procesor već poseduje neko "najbolje" rešenje (o odnosu na koje trenutno traži popravku), stoga će u datom momentu poslati poruku nadređenom samo ukoliko je dobio bolje rešenje u odnosu na to koje već poseduje. U suprotnom, nema potrebe da nadređeni procesor izveštava o svom neuspehu i time zagušuje njegovo poštansko sanduče nebitnim porukama.

Ovakvom implementacijom omogućeno je ostvarivanje nekoliko ciljeva. Prvo, minimizira se količina podataka koji se prenose, drugo minimizira se vreme koje procesori provode u čekanju na poruke. Zatim, forsira se "filozofija prvog poboljšanja (FI)" koja je u osnovi VNS metode: pretraga se koncentriše oko trenutno najboljeg rešenja. Međutim, kako je to "trenutno najbolje rešenje" dinamička kategorija tretirana asinhrono, ostvareno je istovremeno pretraživanje okolina različitih "najboljih" rešenja, što značajno proširuje prostor pretrage nad kojim se izvršava CVNS.

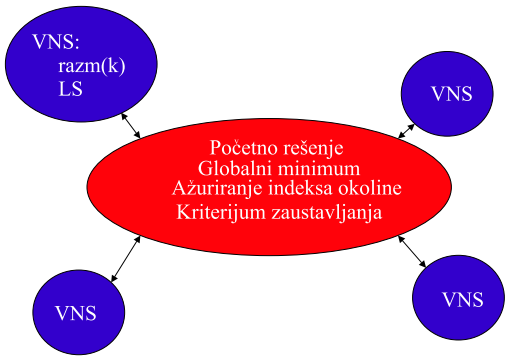
Specijalan slučaj kooperativnog rada različitih VNS procedura predstavlja nezavisno izvršavanje (independent run). Ono podrazumeva da se različite VNS procedure izvršavaju istovremeno na raznim procesorima, a da podatke razmenjuju tek na kraju izvršavanja. Preciznije, svaki VNS izvršava se zasebno, a na kraju se uporede svi dobijeni rezultati i najbolji od njih usvaja se za konačno rešenje. Ova varijanta označena je sa IVNS, i implementirana je radi poređenja se pravim CVNS. Na osnovu ranijih iskustava pretpostavlja se da ova varijanta nije toliko uspešna kao CVNS i implementirana je da bi se ta očekivanja proverila. Implementacija nezavisnog izvršavanja (IVNS) znatno je jednostavnija, svaki procesor izvrši svoju varijantu VNS procedure (brinući istovremeno i o kriterijumu zaustavljanja) i pošalje najbolji rezultat nadređenom procesoru. On uporedi sve rezultate i najbolji od svih prijavi kao konačni rezultat.

Kao što je već napomenuto, kriterijum zaustavljanja je maksimalno doz-

voljeno vreme izvršavanja i o tom kriterijumu brine nadređeni procesor. Osim toga, svaki procesor meri za sebe vreme izvršavanja i ukupno vreme, kako bi se odredilo koliko je vremena potrošeno na komunikacije.

5.6. Distribuirano izvršavanje VNS metode

Poslednja strategija paralelizacije VNS metode koja je u okviru ovoga rada implementirana, nazvana je paralelni VNS (PVNS). Ideja je bazirana na ranijim radovima [25, 58], ali je donekle modifikovana. Polazi se od činjenice da su pretraživanja u raznim okolinama međusobno nezavisna i da se mogu izvršavati istovremeno. Ova ideja realizovana je tako što različiti procesori izvršavaju kombinaciju operacije razmrdavanja (u različitim okolinama) i odgovarajućeg (sekvencijalnog) lokalnog pretraživanja počev od tako dobijenog rešenja (slika 5.9). Razmena podataka obavlja se nakon završetka LS procedure i može biti sinhrona ili asinhrona. Na osnovu sličnog razmatranja kao u prethodnom slučaju, zaključuje se da bi asinhrona implementacija trebalo da bude efikasnija.



Sl. 5.9: Paralelni VNS

Implementaija PVNS izvršena je na istoj višeprocesorskoj arhitekturi kao i prethodne (jedan nadređeni i q podređenih procesora, slika 5.3). Opis implementacije dat je za slučaj kada se radi o jednostavnoj varijanti VNS procedure (lokalno pretraživanje u Swap-1 okolini, unapred do prvog poboljšanja

i k-Swap okolina za razmrdavanje), mada je implementacija identična bez obzira od koje VNS procedure se polazi.

Nakon što se odredi početno rešenje x_{opt} primenom CPES heuristike raspoređivanja i popravi primenom lokalnog pretraživanja, započinje izvršavanje PVNS. Uloga nadređenog procesora je da vodi računa o kriterijumu zaustavljanja, ažuriranju globalno minimalnog rešenja i promeni vrednosti indeksa okoline za razmrdavanje. Podređeni procesori izvršavaju operaciju razmrdavanja tekućeg globalnog minimuma u zadatoj okolini i proceduru lokalnog pretraživanja počev od rešenja dobijenog tim razmrdavanjem.

Koraci koje izvršava nadređeni procesor mogu se opisati na sledeći način. U petlji se čeka na poruku od podređenog procesora. Ta poruka sadrži najbolje rešenje koje poseduje dati procesor (bilo da je to novodobijeno rešenje, bilo da je stari globalni minimum). Kada ona stigne, izvršavaju se sledeći koraci:

- proveri se najpre kriterijum zaustavljanja i, ukoliko je zadovoljen, pošalje se svim procesorima poruka STOP;
- proverava se kvalitet novodobijenog rešenja;
  - Ukoliko je to novi globalni minimum ažuriraju se vrednosti svih relevantnih parametara: pamti se nova vrednost globalno najboljeg rešenja, resetuje se vrednost brojača okolina i datom podređenom procesoru šalje se poruka da počne pretraživanje u okolini $k = 1$ od tog novog rešenja.
  - Ukoliko je globalno najbolje rešenje (koje poseduje nadređeni procesor) bolje od onog koje je poslao podređeni, to bolje rešenje se šalje podređenom, zajedno sa odgovarajućom vrednošću indeksa okoline za razmrdavanje.
- uveća se vrednost brojača okolina k. Ukoliko je premašena vrednost k_{max}, k dobija vrednost 1 (čime se vrše pripreme za eventualni početak nove iteracije PVNS pretraživanja, ukoliko se u međuvremenu ne pojavi novo bolje rešenje).

Petlju, koju izvršava svaki od podređenih procesora, čine sledeći koraci:

1. Čitanje poruke od nadređenog procesora, ukoliko je STOP, prekida se izvršavanje.

2. Ukoliko poruka sadrži novo rešenje x_{opt} i indeks okoline za razmrdavanje k izvrši se operacija razmrdavanja, tj. generiše se slučajno rešenje u k-toj okolini. i dobija novo polazno rešenje x_k.

3. Izvršava se LS procedura počev od x_k. Dobijeno rešenje označava se sa x_{min}.

4. Ako je došlo do popravke globalno najboljeg rešenja, tj. ukoliko važi $f(x_{min}) < f(x_{opt})$, rešenje x_{min} šalje se nadređenom procesoru i čeka se odgovor. Mogući odgovori su:

   (a) To je zaista novo najbolje rešenje i podređeni procesor dobija poruku da nastavi pretraživanje od tog rešenja u okolini $k = 1$.

   (b) U bazi nadređenog procesora postoji bolje rešenje i ono se šalje podređenom procesoru zajedno za tekućom vrednošću k indeksa okoline za razmrdavanje.

   Nakon toga, podređeni procesor nastavlja izvršavanje od koraka 1.

5. Ukoliko je $f(x_{min}) > f(x_{opt})$, stara vrednost x_{opt} šalje se nadređenom procesoru, kako bi se utvrdilo da li je to još uvek najbolje rešenje ili se u međuvremenu pojavilo neko bolje. Dakle, moguće situacije su:

   (a) Ni ostali procesori nisu popravili x_{opt}, pa se stoga pretraživanje nastavlja u odnosu na to rešenje, samo u nekoj novoj okolini.

   (b) U bazi nadređenog procesora postoji novo bolje rešenje x_{opt}, pa se pretraživanje nastavlja od tog rešenja u okolini koja je sledeća na redu (prema evidenciji nadređenog procesora).

   Po donošenju odluke, podređeni procesor vraća se na korak 1.

Opisana implementacija jeste asinhrona, u smislu da podređeni procesori ne sačekuju jedan drugoga nakon izvršene kombinacije razmrdavanje–lokalno pretraživanje. Sinhrona varijanta značila bi da se q okolina pretražuje istovremeno, zatim se prikupe podaci od svih podređenih procesora i na osnovu tih podataka odredi najbolji rezultat u datom trenutku. To određuje pravac daljeg pretraživanja: ili će se pretraživati narednih q okolina, ili će se pretraga vratiti na prvih q okolina, ali u odnosu na novo rešenje. U asinhronom slučaju, nema sačekivanja procesora, ali se zato tekuće najbolje rešenje

određuje samo na osnovu jednog novog i onog postojećeg u bazi. Sinhronizacija na nivou podređeni–nadređeni procesor mora da postoji, da bi se pravilno pratile promene vrednosti indeksa okoline za razmrdavanje.

Evidentno je da se implementacija paralelnih verzija VNS procedura može prilagoditi i drugim višeprocesorskim arhitekturama, kao i da se kod CVNS (IVNS) i PVNS procedura lokalnog pretraživanja može paralelizovati (na neki od načina opisanih ranije) i izvršavati na grupi procesora koju će zatim nadređeni procesor smatrati jedinstvenim podređenim procesorom.

Rezultati paralelizacije lokalnog pretraživanja i VNS metode koja koristi paralelne LS procedure, saopšteni su u radovima [23, 36]. Primene kooperativnog rada različitih VNS procedura referisane su u [37], dok su prvi rezultati u vezi sa PVNS-om izloženi u [38]. Osnovne varijante CVNS i PVNS opisane su u radu [41], a planirano je da se nastavi implementacija novih varijanti.

5.7. Eksperimenti sa paralelnim verzijama VNS metode

U ovom odeljku opisani su rezultati raspoređivanja slučajno generisanih grafova zadataka, dobijeni primenom paralelnih verzija metode promenljivih okolina. Izvršena je analiza i diskusija dobijenih rezultata, kao i modifikacije i poboljšanja koji su se nametnuli kao rezultat tih analiza.

Svi eksperimenti izvođeni su u sledećem radnom okruženju

– Linux operativni sistem;

– Programski jezik C;

– MPI komunikaciona biblioteka;

– SUN Enterprise 10 000 mikroprocesori, sa 64Gb RAM-a i na 400MHz;

– 1-20 procesora;

– slučajno generisani grafovi sa 50-500 zadataka i poznatim optimalnum rešenjima.

Dobijeni rezultati prikazani su u obliku brojnih tabela i grafika. Neke od tabela navedene su u ovoj glavi, dok su druge (pomoćne) date u prilogu B na kraju ovog rada. To je učinjeno radi očuvanja preglednosti i olakšanja praćenja materijala izloženog u ovoj glavi.

5.7.1. Eksperimenti sa paralelnim LS procedurama

Za testiranje paralelnih verzija procedure lokalnog pretraživanja (kao i metode promenljivih okolina) u navedenom okruženju, najpre je vršeno raspoređivanje "teških" primera sa poznatim optimalnim rešenjem [42], tj. primera male gustine. Na tim primerima izvršeno je fino podešavanje parametara, a zatim su analizirani efekti uvođenja paralelizma (ubrzavanje izračunavanja, kvalitet rešenja dobijenog paralelnim pretraživanjem, uticaj povećavanja broja procesora i slično).

Podešavanje parametara

Prva faza svake eksperimentalne analize svakako je izbor vrednosti parametara. U ovom slučaju, potrebno je izabrati parametre ne samo metode koja se paralelizuje, nego i parametre same paralelizacije, tj. parametre paralelnog izvršavanja metode.

Parametri lokalnog pretraživanja su, kako je to opisano u glavi 4., početno rešenje, heuristika raspoređivanja, smer pretraživanja (FOR-BACK) i stepen poboljšanja (FI-BI). Izbor parametara koji se pokazao najbolji u sekvencijalnoj varijanti metode, ne mora dati najbolje rezultate kada se primeni u paralelnom izvršavanju metode. Stoga je potrebno proveriti sve moguće vrednosti parametara i prilikom paralelnog izvršavanja. Razmatranja opisana u ovom odeljku odnose se na Swap-1 okolinu, ali se analogno mogu primeniti i na ostale okoline, kao i na njihove kombinacije razmatrane u glavi 4.

Kada se razmatraju parametri paralelizacije, treba voditi računa o broju angažovanih procesora (koji ulaze u sastav višeprocesorskog sistema opisanog u odeljku 5.2.), komunikacionim i sinhronizacionim detaljima (kada i koliko podataka se prenosi).

Obzirom na razne varijante paralelizacije, koje su u ovome radu predložene, nije moguće izvršiti jedinstvenu parametarsku analizu. Stoga će ona biti rezultat pojedinačnih eksperimenata izvršenih za svaku od predloženih strategija paralelizacije.

Paralelno pretraživanje okoline

U tabeli 5.1 prikazani su prvi rezultati raspoređivanja primenom PNE procedure. Tabela sadrži podatke vezane za sinhrono izvršavanje PNE procedure na $q + 1$ procesora (dakle i nadređeni procesor izvršava deo lokalnog pretraživanja). Svaki red tabele 5.1 sadrži prosečne vrednosti rezultata za 10

primera grafova sa poznatim optimalnim rešenjem, sa gustinom veza među zadacima $\rho = 30\%$ i sa različitim brojem zadataka u grafu (n ide od 50 do 500 sa korakom 50).

Varirani su parametri stepen poboljšanja (BI/FI) i broj procesora q, dok je pretraga vršena u oba smera. Preklapanja koja se pri takvom pretraživanju pojavljuju dozvoljena su zbog paralelnog izvršavanja. Time je omogućeno detaljnije pretraživanje okolina i poređenje sa slučajevima jednosmernog pretraživanja, koje je takođe implementirano i opisano kasnije.

U tabeli su navedeni razni podaci koji treba da ilustruju efekte paralelnog izvršavanja procedure lokalnog pretraživanja. Prva kolona sadrži broj podređenih procesora q (broj angažovanih procesora je za jedan veći). U drugoj koloni naveden je prosečan broj izvršenih iteracija lokalnog pretraživanja. U sledećoj koloni data je prosečna vrednost (za 10 primera) minimalne dužine raspodele određene PNE procedurom (prosečna vrednost dužine optimalne raspodele je 1500 ciklusa izračunavanja, dok je prosečna dužina početnog rešenja, dobijenog primenom CPES heuristike 2840).

Četvrta kolona tabele 5.1 sastoji se iz dva dela, prvi sadrži ukupno vreme rada svakog od angažovanih procesora (t_{tot}), dok je u drugom delu navedeno vreme koje su procesori trošili na izračunavanja (t_{CPU}). Razlika ta dva vremena predstavlja vreme koje su procesori trošili na razmenu podataka i sinhronizaciju. Pretposlednja kolona sadrži zbir vremena rada svih procesora (tzv. total workload) i realno vreme rada (wall-clock time) koje se u nekim slučajevima koristi kao referentno vreme rada. U poslednjoj koloni nalazi se faktor ubrzanja ostvaren paralelnim izvršavanjem (dobija se kao količnik sekvencijalnog vremena izvršavanja, tj. za $q = 0$ i vremena rada najopterećenijeg procesora kada ih je angažovano više).

Gornja polovina tabele 5.1 sadrži rezultate BI pretraživanja, a u donjoj se nalaze odgovarajući podaci za FI pretraživanje. Paralelno BI pretraživanje rezultuje istim konačnim rešenjem kao i sekvencijalno, jedino se ono dobija brže. Kod FI pretraživanja, rezultati su različiti jer se sa povećanjem broja procesora povećava i broj rešenja među kojima se bira novo polazno rešenje za nastavak pretraživanja. Između BOF (best-of-firsts) i FOF (first-of-firsts), izabrano je BOF, jer obe strategije daju slične rezultate, a BOF zahteva manje komunikacije (kod FOF strategije, osim najboljih rešenja, šalju se i odgovarajuća vremena potrebna za njihovo nalaženje). Rezultati pretraživanja okolina samo u jednom smeru (FOR-BACK) prikazani su u tabelama B1 i B2. Ove tabele pokazuju da su rezultati prilično loši, ubrzanje malo, a kod FI pretraživanja ne dobija se na kvalitetu rešenja uvođenjem novih procesora.

Tabela 5.1: Rezultati raspoređivanja paralelnom LS procedurom na $q+1$ procesora

BI pretraživanje				$\sum$ CPU	
q	br.it.	f_{min}	CPU time	WTime	S
0	4.1	2750.9	646.94	646.94	1.00
			(646.93)	652.95	
1	4.1	2750.9	479.91 502.75	982.66	1.29
			(146.49) (502.74)	509.00	
2	4.1	2750.9	366.10 373.53 383.36	1122.99	1.69
			(62.31) (205.62) (383.31)	389.72	
3	4.1	2750.9	281.52 288.33 292.34 297.62	1159.80	2.17
			(33.70) (110.31) (202.91) (297.55)	303.30	
4	4.1	2750.9	234.13 237.48 240.54 241.62 246.66	1200.43	2.62
			(21.59) (68.88) (120.06) (187.29) (246.41)	250.53	

FI pretraživanje				$\sum$ CPU	
q	br.it.	f_{min}	CPU time	WTime	S
0	10.7	2758.8	749.66	749.66	1.00
			(749.66)	774.34	
1	7.6	2778.2	325.97 322.79	648.76	2.30
			(220.18) (211.69)	339.37	
2	6.6	2779.2	325.94 331.60 318.07	975.61	2.26
			(165.01) (279.25) (79.19)	341.06	
3	6.5	2779.3	289.76 297.36 290.35 282.89	1160.36	2.25
			(124.66) (271.96) (150.54) (43.24)	302.62	
4	6.4	2772.8	272.80 279.53 276.76 274.04 271.83	1374.95	2.68
			(108.40) (260.08) (161.42) (91.65) (28.51)	280.26	

Kod pretraživanja samo u jednom smeru, rezultati BI pretrage unapred su nešto lošiji nego kada se pretražuje cela okolina (u oba smera), ali je ubrzanje veće. Pretraživanje unatrag rezultuje popravljanjem minimalne dužine raspodele, ali uz smanjenje faktora ubrzanja. FI pretraživanje se uglavnom ponaša kao "slučajno", ne mogu se uhvatiti nikakve pravilnosti u promeni kvaliteta rešenja sa dodavanjem novih procesora. Ubrzanje je malo i ne raste sa porastom broja angažovanih procesora. Što se tiče broja iteracija,

kod BI pretraživanja je konstantan i manji je nego pri FI pretraživanju, što je očekivani rezultat (isti zaključak važi i u sekvencijalnom slučaju). U slučaju FI pretrage, primećuje se da broj iteracija opada sa povećanjem broja procesora. To se može objasniti činjenicom da se u svakoj iteraciji najbolje rešenje određuje na osnovu više kandidata, što paralelnu pretragu čini bližom BI strategiji i dovodi do smanjenja broja iteracija.

U tabelama 5.1, B1 i B2 prikazani su rezultati dobijeni polazeći od CPES heurističkog rešenja. Odgovarajući rezultati kada se polazi od slučajnog rešenja, dati su u tabelama B3, B4 i B5. Rezultati, pa samim tim i izvedeni zaključci su veoma slični u oba slučaja.

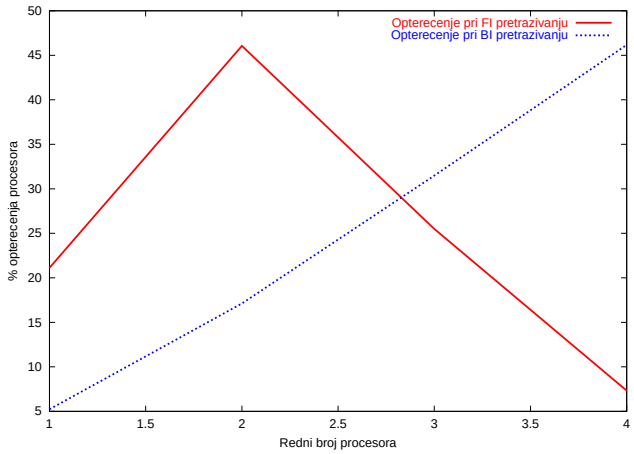
Upoređujući rezultate iz navedenih tabela, zaključuje se da BI strategija pretraživanja uglavnom daje bolje razultate i to uz manji broj iteracija i za približno isto vreme izvršavanja. Ova činjenica (koja ne važi u sekvencijalnom slučaju) posledica je paralelnog izvršavanja jer FI strategija pokazuje tendenciju "slučajnog izbora" kada se izvršava na više procesora.

U tabelama sa rezultatima može se uočiti da je realno vreme izvršavanja (wall-clock time, dato u pretposlednjoj koloni) veoma često značajno veće od stvarnog vremena (datog dužinom rada najopterećenijeg procesora, tj. najvećom vrednosti u prvom redu četvrte kolone). To je posledica time-sharing režima u kome je organizovano izvršavanje procesa na ENTERPRISE 10000 višeprocesorskom sistemu. Pojava procesa višeg prioriteta prekida (suspenduje) izvršavanje nekih procesa nižeg prioriteta. Ukoliko je reč o paralelnom procesu (onom koji zahteva više procesora), prekidaju se izvršavanja na svim procesorima. Pri tome se zaustavlja merenje procesorskog vremena, ali ne i stvarnog i otuda potiču uočene razlike. Stoga se može zaključiti da realno vreme u ovom slučaju ne treba uzimati u obzir, već se ograničiti na stvarna vremena izvršavanja izmerena na svakom od procesora.

Na osnovu rezultata prikazanih u tabelama 5.1, B1, B2, B3, B4 i B5 vidi se da je vreme izračunavanja značajno manje od ukupnog vremena izvršavanja paralelne procedure lokalnog pretraživanja (PNE). Shodno tome, vreme komunikacije i vreme čekanja na komunikacije predstavlja značajan deo izvršavanja paralelnog programa. Naravno, potrebno je precisno utvrditi šta je uzrok tome (sama komunikacija ili vreme čekanja). Na osnovu načina podele okoline za paralelno pretraživanje (opisano u prethodnoj glavi, odeljak 5.2.) i vrednosti za t_{tot} i t_{CPU} datih u četvrtim kolonama tabela, podređeni procesor sa najvećim indeksom obavljao je najviše izračunavanja prilikom BI pretrage, a kod njega je razlika vremena izračunavanja i ukupnog vremena izršavanja izuzetno mala. To ukazuje na neadekvatnu raspodelu posla između

procesora. Količina podataka koju podređeni procesori šalju nadređenom je u svim slučajevima ista, pa je realno očekivati da je i vreme potrebno da se ona izvrši podjednako za sve podređene procesore, a vreme koje nadređeni procesor troši na komunikaciju je q puta veće. Prema tome, zaključuje se da je vreme komunikacije malo, ali se mnogo vremena gubi na čekanje zbog neravnomernog opterećenja procesora. Dakle, uniformna podela okoline predložena u odeljku 5.2. nije adekvatna. Kod FI pretraživanja opterećenje je ravnomernije, mada najviše izračunavanja odradi procesor koji pretražuje drugi deo okoline (tj. procesor sa indeksom 1).

Problem je u tome što u početnom delu permutacije ima najviše izračunavanja, najveći deo permutacije se izmeni kada se u tom delu pretražuje i najviše zadataka treba da se preraspodeli. Sa druge strane, najveća verovatnoća da dođe do (prvog) poboljšanja je baš u tom početnom delu, te se stoga on kraće obrađuje od drugog dela prlikom FI pretraživanja.



Sl. 5.10: Neravnomernost opterećenja procesora pri uniformnoj podeli okoline

Na osnovu ovih razmatranja generisane su krive (slika 5.10) koje odražavaju neravnomernost opterećenja procesora u slučaju FI i BI paralelne pretrage okolina prilikom procedure lokalnog pretraživanja. Sa slike se vidi da kod BI pretrage neravnomernost opterećenja procesora raste sa njegovim indeksom, dok su prilikom FI pretraživanja nadređeni i sledeći (procesor sa indeksom 1) su najopterećeniji (drugi procesor i više), a zatim opterećenje opada sa porastom indeksa procesora. Na osnovu toga, predložena je nova

podela okoline među procesorima, ilustrovana tabelom 5.2, koja obezbeđuje ravnomerniju raspodelu izračunavanja među procesorima.

Tabela 5.2: Procenat zadataka dodeljenih svakom procesoru radi ravnomernog opterećenja

	% zadataka	
q	BI pretraga	FI pretraga
0	100	100
1	75 25	50 50
2	60 25 15	30 20 50
3	50 25 15 10	20 10 20 50
4	50 20 15 10 5	17 5 11 17 50
5	50 15 15 10 5 5	13 5 8 14 20 40

Rezultati raspodele dobijene nakon PNE procedure sa poboljšanom raspodelom opterećenja, počev od CPES polaznog rešenja, prikazani su u tabelama 5.3, 5.4 i 5.5. Odgovarajući rezultati za slučajno izabrano početno rešenje dati su u tabelama B6, B7 i B8. Kao što se vidi, poboljšanje u opterećenju procesora je značajno, a izbor najbolje podele zahteva detaljniju analizu koja ovde nije glavni cilj. Kod BI pretraživanja ubrzanje je skoro linearno, a kvalitet rešenja se, naravno, ne menja. Kod FI pretraživanja, koje je nedeterminističko, ubrzanje nije značajno popravljano, jer je nemoguće predvideti koliko izračunavanja koji procesor izvršava u svakoj od iteracija lokalnog pretraživanja.

Rezultati prikazani u dosadašnjim tabelama ukazuju na to da deoba okoline (podela dopustive permutacije) ne treba da bude ista za slučaj pretraživanja cele okoline i za slučaj pretrage samo u jednom smeru. Isti zaključak važi i za FI-BI pretraživanje, podela treba da se razlikuje zavisno od smera pretrage. Idealna podela dobila bi se upotrebom fine granulacije, tj. dodeljivanjem procesorima pojedinačnih zadataka za pretraživanje sve dok se ne iskoriste svi zadaci (u BI slučaju) ili dok se ne zadovolji FI kriterijum. Međutim, ovakav način podele okoline ima za posledicu intenzifikaciju komunikacije. Sa druge strane, dosadašnji eksperimenti pokazali su da komunikacija ne zahteva previše vremena u ukupnom izvršavanju paralelnog lokalnog pretraživanja. Stoga je implementacija ove varijante, kao i redukovanih varijanti opisanih u odeljku 4.10. predviđena u radu [40].

Tabela 5.3: Rezultati sa popravljenim opterećenjem među procesorima počev od CPES polaznog rešenja i pretraživanjem cele okoline

BI pretraživanje				$\sum$ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	4.1	2750.9	729.02	729.02	1.00
			(729.02)	729.02	
1	4.1	2750.9	390.13 387.18	777.30	1.87
			(390.13) (339.49)	390.15	
2	4.1	2750.9	295.51 297.84 298.57	891.92	2.44
			(237.12) (281.22) (294.90)	298.71	
3	4.1	2750.9	226.21 228.49 226.96 224.94	906.60	3.19
			(162.78) (228.48) (192.60) (147.31)	228.44	
4	4.1	2750.9	187.95 188.09 188.91 187.38 185.34	937.67	3.86
			(162.50) (166.97) (188.57) (148.02) (65.23)	188.89	
5	4.1	2750.9	166.55 165.91 166.64 165.74 163.92 164.83	993.59	4.38
			(162.74) (120.67) (164.45) (135.90) (82.25) (65.31)	166.67	

FI pretraživanje				$\sum$ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	10.7	2758.8	844.12	844.12	1.00
			(843.35)	851.99	
1	8.9	2769.1	446.96 440.52	887.48	1.89
			(327.60) (272.19)	462.54	
2	8.4	2766.4	410.79 418.61 413.33	1242.73	2.07
			(226.19) (317.09) (265.69)	429.00	
3	8.5	2764.2	396.15 395.44 398.79 397.22	1587.61	2.12
			(194.64) (233.16) (304.10) (265.67)	409.32	
4	7.6	2770.5	363.52 364.15 367.65 365.39 366.80	1827.51	2.30
			(169.40) (134.52) (235.79) (262.07) (255.74)	375.95	
5	8.1	2767.9	296.20 296.19 297.87 301.61 299.90 295.88	1787.65	2.80
			(182.88) (102.55) (174.04) (241.60) (226.91) (145.39)	305.93	

Tabela 5.4: Rezultati sa popravljenim opterećenjem među procesorima počev od CPES polaznog rešenja i pretraživanjem unapred

BI pretraživanje				∑ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	4.0	2781.4	389.64	389.64	1.00
			(389.64)	392.29	
1	4.0	2781.4	208.25 208.00	416.25	1.87
			(208.25) (182.74)	211.85	
2	4.0	2781.4	143.33 144.50 143.35	431.18	2.70
			(128.79) (144.29) (118.25)	147.65	
3	4.0	2781.4	117.80 119.06 117.54 117.12	471.53	3.27
			(89.61) (119.00) (97.91) (85.62)	121.17	
4	4.0	2781.4	97.14 97.41 97.62 96.85 95.37	484.40	3.99
			(89.55) (87.82) (96.01) (77.40) (41.73)	99.63	
5	4.0	2781.4	89.97 89.48 89.98 89.04 88.60 88.36	535.44	4.33
			(89.42) (64.11) (83.90) (68.91) (44.17) (41.66)	90.76	

FI pretraživanje				∑ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	7.3	2772.5	178.53	178.53	1.00
			(178.53)	181.74	
1	6.7	2781.1	157.61 159.49	317.10	1.12
			(108.81) (106.54)	165.09	
2	5.9	2789.3	123.91 125.70 125.52	375.13	1.42
			(79.05) (87.53) (79.65)	129.98	
3	5.5	2785.7	110.61 111.47 112.89 112.79	447.76	1.58
			(61.69) (61.94) (86.69) (78.66)	116.21	
4	5.2	2784.4	95.30 93.86 96.17 94.81 95.80	475.94	1.86
			(48.58) (35.43) (65.67) (63.85) (68.28)	98.80	
5	5.9	2782.0	101.57 101.62 102.30 103.08 102.40 101.54	612.50	1.73
			(56.45) (38.77) (60.37) (84.07) (73.19) (50.07)	103.79	

Tabela 5.5: Rezultati sa popravljenim opterećenjem među procesorima počev od CPES polaznog rešenja i pretraživanjem unazad

BI pretraživanje				∑ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	3.8	2737.9	315.51	315.51	1.00
			(315.51)	318.32	
1	3.8	2737.9	168.86 167.03	335.89	1.87
			(168.85) (146.74)	171.25	
2	3.8	2737.9	123.39 126.05 123.13	372.56	2.50
			(100.80) (126.02) (89.40)	127.60	
3	3.8	2737.9	99.64 101.13 99.79 99.18	399.73	3.12
			(68.27) (101.02) (87.69) (59.69)	103.11	
4	3.8	2737.9	85.24 85.21 85.79 84.66 83.91	424.81	3.68
			(68.43) (73.30) (85.46) (65.83) (24.09)	87.35	
5	3.8	2737.9	72.94 72.54 73.48 73.16 72.24 71.89	436.25	4.29
			(68.31) (52.62) (73.26) (62.63) (35.68) (24.08)	73.79	

FI pretraživanje				∑ CPU	
q	n.it.	f_{min}	CPU time	WTime	S
0	8.1	2771.6	197.54	197.54	1.00
			(197.54)	197.56	
1	6.9	2780.8	180.25 179.22	359.47	1.10
			(118.54) (112.87)	183.42	
2	6.2	2778.7	165.07 167.10 167.34	499.51	1.18
			(87.61) (126.59) (116.09)	169.08	
3	6.0	2779.6	146.66 146.60 148.43 147.81	589.49	1.33
			(60.82) (96.75) (122.31) (112.47)	149.26	
4	5.3	2776.1	122.92 123.30 124.83 124.79 125.25	621.08	1.58
			(51.35) (52.72) (91.16) (98.99) (101.51)	126.08	
5	4.9	2775.6	103.13 103.48 103.89 104.84 104.70 103.86	623.90	1.88
			(40.01) (45.52) (68.97) (96.22) (88.49) (60.51)	105.01	

Kod pretraživanja počev od slučajno izabranog rešenja, prethodni zaključak takođe važi. Uz to je još izraženije haotično ponašanje FI pretraživanja, kvalitet rezultata varira sa povećanjem broja procesora, dok je ubrzanje neznatno (pa čak i ne postoji za $q = 5$).

Na osnovu prikazanih rezultata, a vezano za izbor parametara, izvedeno je nekoliko važnih zaključaka. Kao prvo, deoba prostora za pretraživanje je izuzetno značajan faktor za uspešnost paralelnog izvršavanja lokalnog pretraživanja. Pretraživanje celih okolina (tj. premeštanje zadataka u oba smera počev od polazne pozicije) rezultuje neznatno boljim lokalnim minimumom, ali skoro udvostručuje vreme pretraživanja. BI PNE obezbeđuje skoro linearno ubrzavanje pretraživanja. Bez obzira što se ne ponaša stabilno, FI strategija unosi dodatni nedeterminizam u izvršavanje, koji unutar VNS (MLS) okruženja može dati pozitivne efekte. Stoga će u narednim odeljcima biti ponovo testirane obe strategije poboljšanja (FI i BI), okoline će biti podeljene imajući na umu ravnomernost opterećenja procesora i pretraživane samo u jednu stranu, tj. unapred (FORWARD).

Rezultati raspoređivanja primenom paralelnog lokalnog pretraživanja u okviru MLS i VNS procedure prikazani su u narednim odeljcima. U ovom odeljku opisani su eksperimenti sa drugom (asinhronom) varijantom paralelnog lokalnog pretraživanja, LCPLS. Iz prethodnog razmatranja jasno je da treba primeniti balansiranu podelu okoline za pretraživanje. Što se tiče smera pretraživanja i stepena poboljšanja, poželjno je probati sve varijante.

Asinhrono paralelno lokalno pretraživanje

Kao što je opisano u odeljku 5.2., LCPLS je asinhrona varijanta paralelnog lokalnog pretraživanja, u kojoj svaki od procesora pretražuje svoj deo permutacije kao da je to cela okolina, tj. izvršava iteraciju za iteracijom lokalnog pretraživanja sve dok ima poboljšanja. Tek kada se pronađe lokalni minimum u odnosu na pridruženi deo okoline, šalje se nadređenom procesoru i traži novo početno rešenje za nastavak izvršavanja. U ovoj varijanti lokalno pretraživanje izvršava samo q podređenih procesora, dok nadređeni obavlja komunikaciju sa njima i ažurira bazu najboljih rešenja. Asinhronost se ogleda u tome da se ne sačekuju svi procesori za obavljanje komunikacije. Naprotiv, kako koji završi dodeljeno lokalno pretraživanje, šalje rezultat nadređenom i uzima novo rešenje za nastavak izvršavanja, bez obzira na fazu izvršavanja u kojoj su ostali procesori. To, između ostalog, znači da se gubi kontrola nad iteracijama lokalnog pretraživanja.

Tabela 5.6: Rezultati raspoređivanja primenom LCPLS na $q+1$ procesora

BI pretraživanje			$\sum$ CPU
q	f_{min}	CPU time	S
0	2750.90	729.02 (729.02)	729.02 1.00
1	2750.90	685.51 729.32 (685.51) (729.32)	1414.84 0.99
2	2749.50	407.99 425.26 425.99 (407.99) (252.87) (334.27)	1259.25 1.71
3	2748.50	409.10 413.46 417.44 409.72 (409.10) (151.70) (187.75) (202.71)	1649.71 1.75
4	2749.60	271.04 279.56 281.96 280.81 277.93 (271.04) (104.37) (118.93) (112.50) (132.94)	1391.31 2.59
5	2595.20	238.73 257.01 253.31 250.74 246.60 242.42 (238.73) (104.69) (87.73) (125.25) (101.62) (79.00)	1488.81 2.84

FI pretraživanje			$\sum$ CPU
q	f_{min}	CPU time	S
0	2758.80	844.12 (844.12)	844.12 1.00
1	2758.80	800.19 843.31 (800.19) (843.31)	1643.50 1.001
2	2760.40	495.97 509.91 505.56 (495.97) (246.34) (135.74)	1511.45 1.66
3	2758.00	416.60 424.09 417.76 421.28 (416.60) (202.20) (254.21) (135.84)	1679.73 1.99
4	2755.70	332.53 337.25 336.64 337.67 338.71 (332.53) (148.24) (140.13) (154.32) (135.63)	1682.80 2.49
5	2756.10	309.77 317.29 313.88 313.92 313.89 318.53 (309.77) (135.69) (121.13) (149.68) (149.82) (135.52)	1887.27 2.65

U tabeli 5.6 dati su rezultati primene LCPLS procedure na raspoređivanje teških grafova sa poznatim optimalnim rešenjem u slučaju da se pretražuje cela okolina, a polazi od CPES početnog rešenja. Prilikom podele okoline, primenjena je strategija balansiranja opterećenja kao za PNE, bez analize njene pogodnosti u asinhronom slučaju. Kako nadređeni procesor ne vrši lokalno pretraživanje, minimalni broj procesora u ovom slučaju je $q + 1 = 2$. To znači da se izvršavanje sa dva procesora u stvari ponaša kao sekvencijalno. U prvoj vrsti tabele 5.6 dati su odgovarajući rezultati iz prethodnog slučaja, kako bi se omogućilo poređenje i na adekvatan način odredio faktor ubrzanja. Obzirom da je na primeru PNE analizirano realno vreme izvršavanja (wall-clock time) i utvrđeno da se ne može pouzdano koristiti u određivanju faktora ubrzanja, u ovom slučaju to vreme nije uzimano u obzir, već je faktor ubrzanja premešten u drugi deo četvrte kolone.

Iz tabele 5.6 može se izvesti nekoliko zaključaka. Kao prvo, ravnomernost opterećenja je narušena, ubrzanje je manje i to ne samo zbog manjeg broja procesora koji efektivno rade na pretraživanju okoline, već i zbog povećanja obima izračunavanja između dve komunikacije i neravnomerne raspodele izračunavanja medju procesorima. U ovom slučaju je još teže predvideti broj suseda koje će svaki od procesora generisati i ispitati i koji će od procesora imati uspeha u popravljanju rešenja (a koji će uglavnom boraviti u HOLD stanju), te je gotovo nemoguće unapred predložiti pogodnu particiju okoline. Sa druge strane, uočljiva je tendencija poboljšanja kvaliteta dobijenog rešenja sa povećanjem broja procesora. Stoga ovu varijantu ne treba olako odbaciti, već raditi na poboljšanju performansi njenog izvršavanja. Kao jedno od mogućih poboljšanja predloženo je da se smanji vreme boravka procesora u HOLD stanju, time što će mu se dodeliti neko dodatno izračunavanje.

Konkretna implementacija predloženog poboljšanja podrazumeva da se u memoriji nadređenog procesora čuva q najboljih rešenja. Svakom rešenju pridružena je lista procesora koji su ga obrađivali. Kada se neki procesor obrati nadređenom sa porukom koja bi ga dovela u HOLD stanje, šalje mu se najbolje neobrađeno rešenje iz baze. Zadatak nadređenog procesora je da ažurira bazu "dobrih" rešenja, kao i prateće nizove o angažmanu procesora oko svakog konkretnog rešenja. Podređeni procesor pretražuje pridruženi deo okoline novodobijenog rešenja, smatrajući da je ono tekući optimum.

Tabela 5.7: Rezultati raspoređivanja primenom poboljšane LCPLS procedure (LCPLSI) na $q + 1$ procesora

BI pretraživanje			
q	f_{min}	CPU time	S
0	2750.90	729.02	1.00
1	2750.90	659.53 740.40 (659.53) (740.39)	0.98
2	2741.70	768.47 807.06 826.74 (768.47) (502.47) (705.62)	0.88
3	2729.90	774.12 818.77 819.53 825.60 (774.12) (619.63) (691.69) (726.50)	0.88
4	2733.10	739.04 768.52 777.28 775.00 779.59 (739.04) (548.02) (662.78) (619.32) (662.21)	0.94
5	2577.80	1117.12 1177.95 1154.25 1160.77 1153.69 1141.29 (1117.12) (1099.54) (837.10) (867.83) (734.40) (441.04)	0.62

FI pretraživanje			
q	f_{min}	CPU time	S
0	2758.80	844.12	1.00
1	2758.80	1049.73 1138.95 (1049.73) (1138.95)	0.74
2	2747.00	692.15 731.19 701.61 (692.15) (708.91) (324.31)	1.15
3	2745.70	811.52 846.74 819.90 828.99 (811.52) (819.80) (358.35) (479.20)	0.99
4	2744.90	692.55 719.31 707.16 711.43 712.30 (692.55) (689.51) (331.73) (452.73) (567.88)	1.17
5	2738.30	692.66 718.23 703.88 708.31 708.40 714.61 (692.66) (709.38) (246.27) (413.28) (457.16) (619.18)	1.18

Time se povećava verovatnoća nalaženja kvalitetnijeg konačnog rešenja (jer se pretražuju okoline više rešenja), a smanjuje se vreme u kome su procesori blokirani u stanju čekanja (HOLD). Doduše, produžava se ukupno vreme izvršavanja procedure lokalnog pretraživanje, pa i o tome treba voditi računa. Produženje vremena izvršavanja lokalnog pretraživanja posledica je ne samo činjenice da se pretraživanje vrši u okolini više od jednog rešenja, nego i toga što se sa svakom popravkom nekog od tih rešenja pojavljuju nova "dobra" rešenja, koja bivaju ubačena u bazu i obrađivana od strane ostalih procesora. Baza u svakom trenutku sadrži q najboljih rešenja, ali se dinamički menja u toku pretraživanja, što znači da će više od q rešenja učestvovati u procesu pretraživanja. Neka od njih biće delimično obrađena, a samo q zaista najboljih rešenja proći će kompletnu pretragu, tj. biće obrađeno na svim procesorima. Rezultati ove varijante paralelnog lokalnog pretraživanja počev od CPES rešenja dati su u tabeli 5.7. U tabeli 5.8 upoređeni su rezultati obe varijante asinhronog paralelnog lokalnog pretraživanja.

Tabela 5.8: Uporedni rezultati za dve varijante asinhronog paralelnog lokalnog pretraživanja

BI pretraživanje					FI pretraživanje			
	LCPLS		LCPLSI		LCPLS		LCPLSI	
q	f_{opt}	S	f_{opt}	S	f_{opt}	S	f_{opt}	S
0	2750.90	1.00	2750.90	1.00	2758.80	1.00	2758.80	1.00
1	2750.90	0.99	2750.90	0.98	2758.80	1.001	2758.80	0.74
2	2749.50	1.71	2741.70	0.88	2760.40	1.66	2747.00	1.15
3	2748.50	1.75	2729.90	0.88	2758.00	1.99	2745.70	0.99
4	2749.60	2.59	2733.10	0.94	2755.70	2.49	2744.90	1.17
5	2595.20	2.84	2577.80	0.62	2756.10	2.49	2738.30	1.18

Kao što se iz datih tabela vidi, ravnomernost opterećenja značajno je popravljena, a kvalitet konačnog rešenja raste sa povećanjem broja procesora. Vreme izvršavanja jeste produženo, što se ogleda u nedostatku ubrzanja, ali to može biti zanemarljiva posledica, ukoliko dobijeni rezultati budu kvalitetniji kada se ova varijanta ubaci u metaheurističko okruženje, pri čemu je kriterijum zaustavljanja maksimalno dozvoljeno vreme izvršavanja.

Može se razmišljati i o fino granulisanom asinhronom lokalnom pretraživanju koje bi se realizovalo tako što nadređeni procesor šalje zadatak po

zadatak svakom od podređenih, oni pretražuju okolinu definisanu samo tim zadatkom i šalju rezultat nadređenom procesoru. Ukoliko nema poboljšanja, prelazi se na novi zadatak, a ukoliko je nađeno novo najbolje rešenje vrši se njegova podela na pojedinačne zadatke koji se šalju podređenim procesorima. Ova varijanta je prilično složena, kako za implementaciju, tako i za praćenje toka pretraživanja, pa neće biti razmatrana u okviru ovoga rada.

5.7.2. Rezultati VNS i MLS koje koriste paralelni LS

Kao prva varijanta, implementirano je paralelno višestartno lokalno pretraživanje koje koristi PNE na svih $q+1$ procesora. Rezultati raspoređivanja retkih grafova sa poznatim optimalnim rešenjima dati su u tabeli 5.9. Pretraživanje se izvršava u petlji, koja se sastoji od izbora slučajnog rešenja i izvršavanja PNE procedure u njegovoj Swap-1 okolini, sve dok se ne zadovolji kriterijum zaustavljanja. Podela okoline među procesorima prati tabelu za balansiranje opterećenja.

U prvoj koloni tabele 5.9 naveden je broj podređenih procesora, a u drugoj broj ponavljanja procedure lokalnog pretraživanja (ujedno i broj slučajno generisanih početnih rešenja). Treća kolona sadrži dva podatka, prvi je minimalna dobijena prosečna dužina raspodele, a drugi prosečno vreme potrebno da se dobije odgovarajuće najbolje rešenje. Podaci o vremenu izvršavanja i vremenu izračunavanja dati su kao $q+1$ podatak u poslednjoj koloni.

Obzirom da je izlazni kriterijum maksimalno dozvoljeno vreme izvršavanja, faktor ubrzanja kao eksplicitan podatak ne postoji. Ostvareno ubrzanje evidentno je u slučaju BI pretraživanja na osnovu broja ponavljanja LS procedure. Sa povećanjem broja procesora raste i broj izvršenih lokalnih pretraživanja, a samim tim i kvalitet dobijenog rešenja.

Iako ohrabrujući, rezultati BI pretraživanja nisu zadovoljavajući, jer je najbolje dobijeno reženje (za $q = 5$) još uvek lošije od sekvencijalnog FI pretraživanja. Sa druge strane, FI pretraživanje sadrži isuviše slučajnosti i rezultati variraju bez neke pravilnosti. Dodatno, poremećen je balans opterećenja procesora, jer se razlike nagomilavaju tokom uzastopnih izvršavanja lokalnih pretraživanja. Stoga se može zaključiti da prilikom paralelnog izvršavanja MLS gubi dobre osobine koje su se mogle primetiti u sekvencijalnom slučaju. Stoga neće biti uzimana u razmatranje u daljim analizama upotrebe paralelnog lokalnog pretraživanja.

Tabela 5.9: Rezultati PMLSPNE, pri balansiranom opterećenju procesora

q	n.it.	f_{min} t_{min}	BI pretraživanje CPU time
0	11.5	2750.50	5222.88
		2387.38	(5219.64)
1	22.40	2743.20	5090.16 5092.80
		1559.94	(4926.51) (4997.66)
2	30.10	2725.40	5077.13 5110.93 5080.01
		1304.44	(4046.64) (5050.14) (4473.46)
3	37.60	2721.00	5063.59 5135.77 5124.51 5076.72
		2609.86	(3430.94) (5068.65) (4754.34) (3843.50)
4	45.20	2710.10	5080.12 5100.55 5117.10 5088.05 5016.04
		2172.64	(4112.26) (4606.00) (5043.37) (4294.29) (2439.60)
5	50.40	2710.10	5070.26 5047.80 5078.40 5066.00 4989.36 5027.13
		1972.64	(4582.31) (3614.44) (5005.31) (4531.53) (2429.90) (2725.23)
q	n.it.	f_{min} t_{min}	FI pretraživanje CPU time
0	20.70	2708.10	5131.18
		2756.47	(5124.79)
1	21.40	2737.50	5123.62 5022.67
		1831.99	(4398.59) (2036.21)
2	24.50	2736.60	5122.41 5045.30 5065.94
		2639.73	(3878.15) (2782.71) (2296.85)
3	28.90	2661.70	5517.75 5485.83 5484.11 5454.72
		4085.10	(4024.46) (2530.57) (3266.15) (2768.98)
4	31.90	2651.60	5353.73 5248.86 5300.30 5307.96 5300.98
		4048.89	(3802.25) (1698.47) (2850.64) (3014.68) (3008.58)
5	37.20	2722.70	5092.39 5040.43 5075.93 5098.95 5086.06 5036.97
		2603.11	(3238.98) (2094.71) (2744.35) (3526.13) (3184.51) (2125.44)

VNS metoda koja koristi PNE sa $q+1$ procesora testirana je za različite vrednosti parametra k_{max}. Korišćena je konstantna vrednost $k_{max} = 20$ za sve primere, bez obzira na njihovu dimenziju i dve vrednosti koje zavise od dimenzije primera $k_{max} = n/4$ i $k_{max} = n/2$. Rezultati su prikazani redom u tabelama 5.10, 5.11 i 5.12. U svim slučajevima pretraživanje počinje od CPES heurističkog rešenja, a u toku izvršavanja, obilazi se Swap-1 okolina unapred.

Rezultati prikazani u tabeli 5.10 navode na slične zaključke kao u slučaju paralelne MLS procedure. Prilikom BI pretraživanja, zahvaljujući velikom ubrzanju, kvalitet rešenja popravlja se sa povećanjem broja angažovanih procesora. Sa druge strane, ne treba zanemariti neefikasnost BI pretraživanja (opisanu na sekvencijalnom slučaju u glavi 4.). Ta neefikasnost se ne gubi pri paralelnom izvršavanju, što dokazuje činjenica da je najbolji paralelni rezultat, dobijen angažovanjem 6 procesora na BI pretraživanju, samo za nijansu bolji od sekvencijanog FI pretraživanja.

FI pretraživanje se u paralelnom slučaju ponaša prilično nepredvidljivo, te stoga kvalitet konačnog rešenja varira pri promeni broja procesora. Dodatno, uočava se povećavanje razlike u balansiranosti opterećenja procesora, koja se nagomilava tokom mnogobrojnih iteracija lokalnog pretraživanja. Kada je reč o iteracijama, primećuje se da je broj VNS iteracija izuzetno mali u oba slučaja. Bez obzira što je maksimalni broj okolina svega 20, on se u većini slučajeva dostigne samo jedanput. Stoga je očito da je pretraga koncentrisana u regionu bliskom trenutno najboljem rešenju, te nije realno očekivati značajnija poboljšanja konačnih rezultata.

Da bi se obezbedila diversifikacija pretraživanja u udaljenije regione, povećana je vrednost k_{max}, ali korišćen je i parametar k_{step}, koji treba da obezbedi da se maksimalna vrednost za indeks okoline u operaciji razmrdavanja i dostigne. U tabelama 5.11 i 5.12 prikazani su odgovarajući rezultati.

Kao što se iz rezultata prikazanih u tabeli 5.11 vidi, diversifikacijom su postignuta oba cilja, dostizanje vrednosti za k_{max} više puta i bolji rezultati u slučaju BI pretraživanja. Međutim, kada je pretraga još više proširena (tabela 5.12), dobijen je efekat nasumičnog pretraživanja.

Tabela 5.10: Rezultati PVNSPNE, pri balansiranom opterećenju procesora, $k_{max} = 20$

q	n.it.	f_{min}	BI pretraživanje CPU time
0	1.0	2713.80	5110.02 (5108.07)
1	1.30	2537.90	5370.98 5340.86 (5367.44) (4786.41)
2	1.40	2476.10	5040.65 5061.51 5030.59 (4519.13) (5041.26) (4223.81)
3	1.80	2462.70	5084.50 5131.02 5097.48 5065.47 (3886.94) (5125.21) (4392.40) (3643.43)
4	2.10	2435.70	5063.12 5054.78 5069.63 5033.49 4993.51 (4673.87) (4611.53) (4967.34) (4104.93) (2210.61)
5	2.20	2414.50	5041.51 5006.50 5034.40 5016.74 4984.88 4966.11 (5012.12) (3538.05) (4704.01) (4106.06) (2355.60) (2377.54)
q	n.it.	f_{min}	FI pretraživanje CPU time
0	1.10	2427.90	5308.02 (5306.02)
1	1.20	2554.50	5259.53 5024.16 (4931.97) (1744.28)
2	1.30	2553.20	5093.36 5022.53 4962.39 (4518.43) (2524.66) (2176.97)
3	1.90	2434.00	5239.99 5167.66 5184.22 5195.80 (4444.01) (2264.70) (3172.89) (2840.78)
4	1.30	2384.80	5069.12 4982.44 5015.81 5028.07 5029.29 (4224.56) (1501.16) (2501.95) (2819.71) (3074.62)
5	1.50	2420.90	5106.66 5028.93 5080.77 5100.21 5080.09 5080.85 (4136.63) (1886.90) (2625.59) (3485.86) (3375.13) (2440.36)

Tabela 5.11: Rezultati PVNSPNE, pri balansiranom opterećenju procesora, $k_{max} = n/4$, $k_{step} = k_{max}/5$

q	n.it.	f_{min}	BI pretraživanje CPU time
0	2.30	2477.60	5467.71 (5466.59)
1	3.10	2383.00	5163.64 5135.27 (5149.81) (4725.93)
2	4.00	2319.30	5209.77 5237.07 5215.44 (4611.41) (5206.38) (4580.04)
3	5.10	2302.40	5033.70 5089.08 5066.40 5032.55 (4149.96) (5079.68) (4531.56) (3761.66)
4	6.00	2172.40	5252.32 5260.45 5284.16 5256.67 5176.05 (4749.37) (4835.58) (5172.88) (4636.55) (2347.98)
5	6.80	2154.60	5050.22 5017.89 5047.09 5031.26 4995.89 4971.38 (4983.34) (3607.94) (4858.97) (4367.38) (2562.60) (2467.50)
q	n.it.	f_{min}	FI pretraživanje CPU time
0	3.20	2410.90	5403.35 (5401.33)
1	3.40	2554.50	5374.87 5179.15 (4676.19) (2230.43)
2	3.70	2604.60	5112.03 5060.80 5074.14 (3746.68) (3076.93) (2566.17)
3	4.80	2484.70	5093.91 5003.56 5050.74 5056.59 (3958.94) (2270.04) (3128.91) (2605.50)
4	5.60	2527.40	5162.75 5113.41 5127.48 5152.67 5132.54 (3737.42) (1665.52) (2832.01) (3007.54) (3102.04)
5	6.00	2573.50	5143.66 5053.99 5086.22 5119.89 5119.07 5082.20 (3672.82) (2006.56) (2726.80) (3614.94) (3487.75) (2409.20)

Tabela 5.12: Rezultati PVNSPNE, pri balansiranom opterećenju procesora, $k_{max} = n/2$, $k_{step} = k_{max}/4$

q	n.it.	f_{min}	BI pretraživanje CPU time
0	2.00	2598.90	5244.34 (5243.32)
1	4.50	2589.10	5053.64 5025.12 (5050.51) (4596.49)
2	6.60	2567.20	5118.01 5140.87 5110.78 (4454.63) (5124.70) (4284.81)
3	8.00	2458.40	5109.90 5151.00 5129.89 5090.62 (3648.31) (5147.16) (4419.78) (3626.12)
4	9.80	2457.00	5051.60 5060.29 5068.57 5042.34 4991.99 (4450.91) (4790.85) (4990.73) (4092.32) (2351.51)
5	10.90	2446.10	5026.35 4985.59 5026.39 5010.16 4934.50 4962.45 (4841.04) (3716.82) (4925.31) (4229.30) (2312.14) (2575.04)
q	n.it.	f_{min}	FI pretraživanje CPU time
0	3.30	2526.40	5111.48 (5109.95)
1	4.70	2535.40	5136.06 5028.06 (4263.65) (2231.07)
2	4.90	2555.40	5091.68 5047.88 5027.02 (4123.63) (2654.38) (2178.52)
3	6.00	2545.70	5086.69 5069.49 5070.87 5043.06 (3721.05) (2389.03) (3163.55) (2604.25)
4	7.20	2536.40	5120.78 5092.92 5097.37 5100.61 5128.96 (3460.48) (1718.98) (2970.28) (3118.71) (3189.55)
5	7.70	2639.70	5095.19 5092.74 5099.41 5105.76 5099.56 5080.74 (3055.78) (2061.68) (2826.64) (3857.87) (3631.11) (2427.03)

Prilikom korišćenja asinhronih verzija lokalnog pretraživanja u okviru paralelnog izvršavanja MLS i VNS metoda, najbolji rezultati dobijeni su primenom LCPLSI. MLS, kao ni u prethodnom slučaju nije dalo dobre rezultate, pa oni nisu ni navedeni u okviru ovoga rada. U tabeli 5.13 prikazani rezultati paralelnog izvršavanja VNS sa LCPLSI lokalnim pretraživanjem i parametrima $k_{max} = n/4$ i $k_{step} = k_{max}/5$.

Osnovni zaključak koji se iz ove tabele može izvesti je da izostanak nadređenog procesora u izvršavanju lokalnog pretraživanja ima negativne posledice. Ravnomernost opterećenja procesora izračunavanjima postignuta je u slučaju BI pretraživanja (nadređeni ne vrši izračunavanja vezana za lokalno pretraživanje i zato je kod njega vremena nisu balansirana). Bez obzira na to, nezanemarljivo je vreme koje se troši na komunikaciju i prazan hod (HOLD stanje). FI pretraživanje je, kao i kod ostalih varijanti paralelizacije LS procedure, nepouzdano u pogledu zavisnosti kvaliteta konačnog rešenja od broja procesora.

U tabeli 5.14 upoređene su najbolje varijante PVNSPNE i PVNSLCPLSI. Kao što se iz ove tabele može videti, asinhrono izvršavanje nije se pokazalo kao dobro. Jedan od razloga za to je što je nadređeni procesor korišćen samo za komunikaciju i upravljačke svrhe. Drugi razlog je što izvršene modifikacije produžavaju proceduru lokalnog pretraživanja i to je evidentno na osnovu manjeg broja izvršenih VNS iteracija. Asinhrono izvršavanje i uvedena modifikacija imaju za posledicu i nemogućnost praćenja toka pretraživanja pa se ne može predvideti krajnji ishod izvršavanja VNS procedure. Kao što se vidi, pri asihnronom izvršavanju rezultati se ne popravljaju nužno uvođenjem novih procesora, čak ni pri BI pretraživanju.

Konačno, na osnovu opisane implementacije i izvršenih eksperimenata, u vezi sa paralelizacijom lokalnog pretraživanja, može se zaključiti da sinhrona varijanta u kojoj svi procesori izvršavaju po deo lokalnog pretraživanja može da omogući popravljanje kvaliteta konačnog rešenja dodavanjem novih procesora. Dobijene rezultate moguće je dodatno popraviti primenom neke druge strategije za uravnotežavanje opterećenja procesora izračunavanjima, možda čak i nekom koja će to opterećenje određivati dinamički tokom samog izvršavanja VNS procedure sa paralelnim lokalnim pretraživanjem. Ostalo je da se ispita i finija distribucija okoline po procesorima, kao i varijante sa pretraživanjem smanjene okoline.

Tabela 5.13: Rezultati za PVNSLCPLSI

BI pretraživanje			
q	n.it.	f_{min}	CPU time
1	1.00	2480.50	6646.54 7047.39 (2988.83) (7046.90)
2	2.10	2564.70	5329.73 5528.14 5649.22 (1392.68) (3137.80) (5024.59)
3	3.60	2442.80	5187.19 5373.31 5388.84 5426.27 (1110.78) (3612.69) (3459.14) (4522.08)
4	4.90	2347.40	5060.43 5152.51 5209.17 5199.03 5216.28 (942.87) (3025.01) (3822.95) (3601.75) (4092.91)
5	5.00	2540.40	5083.93 5168.28 5150.16 5139.88 5142.68 5071.31 (1357.89) (4326.41) (3468.25) (3355.62) (2935.08) (1543.30)
FI pretraživanje			
q	n.it.	f_{min}	CPU time
1	1.10	2579.00	5653.09 5942.19 (3310.75) (5941.62)
2	2.30	2388.50	5173.39 5466.47 5256.71 (1235.79) (5139.64) (1473.87)
3	3.60	2389.10	5184.96 5416.23 5234.43 5317.10 (1304.58) (5075.53) (1426.31) (2105.48)
4	4.90	2583.30	5101.59 5313.39 5216.45 5244.01 5246.87 (827.52) (4855.81) (1409.63) (2202.56) (3055.58)
5	5.40	2431.00	5092.31 5198.35 5064.18 5136.15 5102.43 5161.59 (938.56) (4559.31) (827.93) (1460.68) (1661.99) (2986.60)

Tabela 5.14: Uporedni rezultati za dve varijante paralelizovane VNS procedure

BI pretraživanje						FI pretraživanje				
	PVNSPNE		PVNSLCPLS				PVNSPNE		PVNSLCPLS	
q	n.it.	f_{min}	n.it.	f_{min}		q	n.it.	f_{min}	n.it.	f_{min}
1	3.10	2383.00	1.00	2480.50		1	3.40	2554.50	1.00	2579.00
2	4.00	2319.30	2.10	2564.70		2	3.70	2604.60	2.30	2388.50
3	5.10	2302.40	3.60	2442.80		3	4.80	2484.70	3.60	2389.10
4	6.00	2172.40	4.90	2347.40		4	5.60	2527.40	4.90	2583.30
5	6.80	2154.60	5.00	2540.40		5	6.00	2573.50	5.40	2431.00

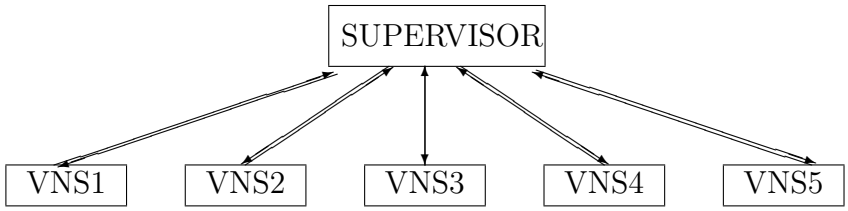
5.7.3. Eksperimenti sa CVNS-om

Već je rečeno da kooperativni rad VNS procedura znači da se na svakom od procesora izvršava jedna sekvencijalna varijanta VNS procedure i da se u unapred definisanim koracima izvršavanja (asinhrono) razmenjuju dobijeni rezultati. Podaci se razmenjuju u sledećim trenucima izvršavanja: (1) svaki put kada se popravi tekuće najbolje rešenje i (2) u dva slučaja ukoliko nema popravke, (a) kada se uvećava indeks okoline za razmrdavanje i (b) na kraju VNS iteracije, tj. kada se obiđe svih k_{max} okolina bez popravke. Uticaj i značaj komunikacija vide se poređenjem (a) i (b) varijante međusobno i sa nezavisnim izvršavanjem.

Što se tiče varijanti VNS metoda koje se izvršavaju paralelno, testirane su dve kombinacije (mada su moguće i druge). Prva kombinacija je ona prikazana na slici 5.7, dok je za drugu korišćeno pet procesora i dodata je varijanta VNS metode koja koristi VND (tj. Swap-321 kombinaciju okolina kao što je to predloženo u glavi 4.) u procesu lokalnog pretraživanja (slika 5.11).

Parametri VNS metoda fiksirani su (do na stepen poboljšanja, FI-BI) izborom varijanti koje se izvršavaju na svakom od procesora, broj procesora je takođe fiksiran, a komunikacioni deo se testira. Testovi se odnose kako na komunikacione tačke tako i na informacije koje se prenose. Ispitivane varijante su prenošenje najboljeg rešenja i prosleđivanje slučajno izabranog rešenja iz baze dobrih rešenja koju čuva i ažurira nadređeni procesor.

U nekoliko narednih tabela prikazani su rezultati raspoređivanja "teških"



VNS1	VNS2	VNS3	VNS4	VNS5
LS: Swap-1	LS: Swap-1	LS: IntCh	LS: IntCh	LS: VND
Sh: k-Swap	Sh: k-IntCh	Sh: k-Swap	Sh: k-IntCh	Sh: k-Swap
FORWARD	BACKWARD	BACKWARD	FORWARD	FORWARD

Sl. 5.11: Primer izvršavanja kooperativne VNS-VND procedure

primera sa poznatim optimalnim rešenjima. Tabele sadrže podatke za svaki primer pojedinačno jer, u ovom slučaju, zbirni podaci nisu dovoljno informativni. Kriterijum zaustavljanja je maksimalno dozvoljeno vreme. Parametri VNS metode u svim primerima su $k_{max} = 10$, $k_{step} = 1$ i $plateaux = 0.0$.

U prvoj koloni dat je broj zadataka u grafu koji se raspoređuje. Prvi podatak u drugoj je broj VNS-iteracija. Pri tome je relevantan podatak broj započetih, a ne završenih iteracija i to prema evidenciji nadređenog procesora. Naime, ukoliko se iz baze dobije novo najbolje rešenje, svaki od podređenih procesora prelazi na pretraživanje najbliže okoline tog novog rešenja u skladu sa koracima VNS metode. Komanda za zaustavljanje se, takođe, može obraditi pre kraja tekiće VNS iteracije.

U drugom redu druge kolone date su najmanje pronađene dužine raspodela (optimalne vrednosti mogu se videti u tabeli 4.3), a narednih $q + 1$ kolona sadrže vremena izvršavanja i vremena izračunavanja za svaki od procesora koji učestvuje u kooperativnom radu. U q kolona koje slede, navedeno je koliko puta je koji od podređenih procesora popravio tekuće najbolje rešenje i (u sledećem redu) koliko je od tih popravki dalo novo najbolje rešenje. U svim primerima polazi se od CPES heurističkog rešenja.

U prve četiri tabele dati su rezultati kooperativnog rada 4 i 5 VNS procedura koje komuniciraju pri svakoj popravci ili pri svakom uvećanju indeksa okoline za razmrdavanje, dok naredne četiri tabele sadrže odgovarajuće rezultate u slučaju da se komunikacija inicira pri popravci ili prilikom dostizanja maksimalne vrednosti indeksa okoline za razmrdavanje.

Tabela 5.15: Rezultati za 4 VNS, BI, komunikacije na k++

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput. time					Novo najbolje			
50	31	6.75	6.49	6.58	7.07	7.03	11	7	6	7
	767	6.02	6.492	6.582	7.07	7.03	9	5	5	4
100	39	76.26	74.82	79.61	78.27	79.18	14	10	9	6
	1283	70.98	74.82	79.61	78.27	79.18	10	7	4	4
150	65	437.85	433.87	431.05	435.05	454.08	22	15	15	13
	1053	403.82	433.87	431.05	435.05	454.08	15	10	6	8
200	52	655.52	647.34	652.70	680.04	656.59	18	11	14	9
	2111	607.46	647.34	652.70	680.04	656.59	11	9	10	4
250	45	1148.82	1107.57	1191.98	1076.63	1090.11	16	9	11	9
	2540	1037.33	1107.57	1191.98	1076.62	1090.11	13	7	10	7
300	41	2219.37	2310.02	2310.98	2205.19	2174.64	10	7	14	10
	2222	2013.22	2310.02	2310.98	2205.18	2174.64	7	5	8	6
350	54	4198.32	4297.01	4209.44	4363.65	4323.45	21	9	14	10
	3370	4044.77	4297.01	4209.44	4363.65	4323.45	15	2	8	4
400	62	6815.45	6384.10	7092.49	6743.71	6721.42	19	13	19	11
	3728	6079.73	6384.10	7092.49	6743.71	6721.41	16	6	12	8
450	64	10575.78	10705.94	10595.90	10385.23	10968.27	22	12	16	14
	4151	10006.08	10705.94	10595.90	10385.23	10968.27	18	6	13	10
500	79	17756.02	16906.35	17899.16	18509.64	17988.93	33	11	22	13
	3428	16135.07	16906.35	17899.15	18509.63	17988.93	31	8	19	10
q =4	53.2	4389.01	4287.35	4446.99	4448.45	4446.37				
	2465.3	4040.45	4287.35	4446.99	4448.44	4446.37				

Tabela 5.16: Rezultati za 4 VNS, FI, komunikacije na k++

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	33	7.63	5.34	8.67	7.55	8.69	8	5	9	11
	699	6.22	5.34	8.67	7.55	8.69	4	2	4	3
100	44	82.87	76.80	87.02	86.95	77.86	15	4	14	11
	1227	70.15	76.80	87.02	86.95	77.85	6	2	7	7
150	93	469.22	419.87	479.52	421.85	422.00	39	4	21	29
	1336	400.44	419.86	479.52	421.85	422.00	19	0	10	18
200	51	1036.74	672.59	1017.25	784.78	645.22	23	2	9	17
	2128	610.84	672.59	1017.25	784.78	645.22	11	0	6	9
250	69	1047.66	1186.29	1173.67	1162.50	1155.49	31	1	16	21
	2505	1019.28	1186.29	1173.67	1162.49	1155.48	22	0	11	15
300	44	3121.25	2172.97	3190.26	2217.88	2711.85	18	1	12	13
	2359	2009.41	2172.97	3190.26	2217.87	2711.85	10	0	6	8
350	67	4319.09	4265.23	4346.03	4502.52	4328.62	26	1	20	20
	3331	4070.82	4265.23	4346.03	4502.52	4328.62	19	0	15	9
400	61	11430.55	6953.25	12008.85	6471.37	6567.73	26	1	18	16
	3777	6126.61	6953.25	12008.85	6471.37	6567.73	15	1	9	6
450	82	12405.91	10861.23	12100.63	10701.38	13042.49	42	1	18	21
	3938	10153.55	10861.23	12100.63	10701.38	13042.49	33	1	10	17
500	79	29942.56	21904.67	31164.06	21749.26	18032.70	41	1	27	10
	3470	16064.28	21904.64	31164.06	21749.24	18032.70	28	1	22	8
q =4	62.3	6386.35	4851.82	6557.60	4810.60	4699.26				
	2477.0	4053.16	4851.82	6557.60	4810.60	4699.26				

Tabela 5.17: Rezultati za 5 VNS, BI, komunikacije na k++

n	n.it.	CPU time						Poboljšanja				
	f_{min}	Comput time						Novo najbolje				
50	35	6.90	6.49	6.57	7.08	7.03	6.44	11	7	6	7	4
	767	6.11	6.49	6.57	7.08	7.03	6.44	9	5	5	4	2
100	45	77.83	74.86	79.64	78.29	79.11	74.95	14	10	9	6	6
	1283	72.23	74.86	79.64	78.29	79.11	74.95	10	7	4	4	3
150	71	444.59	415.41	432.28	419.01	454.66	442.47	21	15	13	13	9
	1053	403.18	415.41	432.28	419.01	454.66	442.47	14	10	6	8	5
200	57	643.62	642.45	654.42	630.07	657.10	629.33	18	11	13	9	6
	2111	600.14	642.45	654.42	630.07	657.10	629.33	11	9	9	4	3
250	54	1291.34	1087.53	1193.79	1079.95	1090.05	1326.08	17	9	11	9	8
	2540	1049.01	1087.53	1193.79	1079.95	1090.05	1326.07	13	7	10	7	5
300	45	3026.13	2101.87	2315.13	2207.08	2180.91	3096.41	9	7	14	10	5
	2405	2007.02	2101.87	2315.13	2207.07	2180.91	3096.40	6	5	8	6	2
350	58	4267.15	4208.78	4286.22	4365.01	4325.35	4208.01	18	9	14	10	7
	3370	4023.20	4208.78	4286.22	4365.00	4325.34	4208.00	12	2	9	4	4
400	75	6948.53	6662.06	7101.64	6316.32	6727.55	6719.69	21	13	18	11	12
	3728	6104.00	6662.06	7101.64	6316.32	6727.55	6719.69	17	6	11	8	7
450	68	10481.54	10721.58	10605.43	10386.06	10375.09	10406.70	22	12	16	13	5
	4151	10087.04	10721.57	10605.43	10386.06	10375.09	10406.70	18	6	13	10	3
500	91	18891.81	19350.98	19273.78	18482.95	17989.82	18347.09	31	10	22	13	15
	3452	16655.83	19350.98	19273.77	18482.95	17989.81	18347.09	28	8	19	10	11
q =5	59.9	4607.94	4527.20	4594.89	4397.18	4388.67	4525.72					
	2486.0	4100.78	4527.20	4594.89	4397.18	4388.66	4525.71					

Tabela 5.18: Rezultati za 5 VNS, FI, komunikacije na k++

n	n.it.	CPU time						Poboljšanja				
	f_{min}	Comput time						Novo najbolje				
50	38	8.14	6.20	6.91	6.78	8.03	6.45	10	4	8	11	5
	699	6.06	6.20	6.91	6.78	8.03	6.45	4	2	5	5	1
100	54	79.74	77.93	81.82	73.23	81.46	78.18	14	4	14	12	10
	1191	70.02	77.92	81.81	73.23	81.46	78.18	6	1	5	7	6
150	97	436.73	424.36	451.15	453.95	420.84	445.82	35	3	23	21	15
	1095	405.00	424.36	451.15	453.95	420.83	445.82	19	0	9	13	8
200	62	847.27	633.38	869.11	621.66	677.44	719.00	21	2	13	16	10
	2095	601.39	633.38	869.11	621.65	677.42	719.00	10	0	8	8	6
250	76	1137.72	1130.17	1171.07	1089.45	985.03	1121.34	29	1	14	18	14
	2508	1023.37	1130.17	1171.07	1089.44	985.03	1121.34	21	0	10	14	9
300	68	3078.35	2139.18	3194.35	2340.60	2404.45	2162.07	24	1	12	18	13
	2940	2000.49	2139.18	3194.35	2340.60	2404.45	2162.04	13	0	7	11	6
350	86	5242.35	4355.35	5462.23	4324.06	4326.32	4283.38	27	1	19	22	17
	3342	4016.01	4355.34	5462.23	4324.06	4326.32	4283.38	21	0	16	13	16
400	74	7017.31	6388.35	6852.44	6500.56	7159.21	6519.20	25	1	17	17	14
	3773	6070.23	6388.34	6852.44	6500.56	7159.21	6519.20	15	0	10	8	8
450	99	15930.54	10505.21	12105.97	10608.16	16280.36	10531.66	37	1	16	27	18
	4046	10046.39	10505.20	12105.97	10608.16	16280.36	10531.66	31	1	11	18	10
500	74	61785.61	31584.60	62930.12	36962.48	36986.18	31851.97	24	1	20	13	16
	3042	30759.93	31584.59	62930.12	36962.45	36986.18	31851.97	18	1	15	7	12
q =5	72.8	9556.38	5724.47	9312.52	6298.09	6932.93	5771.91					
	2473.1	5499.89	5724.47	9312.52	6298.09	6932.93	5771.90					

Tabela 5.19: Rezultati za 4 VNS, BI, komunikacije na k_{max}

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	6	36.42	15.04	37.49	8.77	28.36	2	2	1	1
	768	7.76	15.03	37.49	8.77	28.36	2	2	1	1
100	13	167.92	174.24	109.90	114.32	134.96	4	5	2	2
	1189	76.25	174.24	109.90	114.32	134.96	1	5	1	1
150	17	1188.02	447.72	652.32	740.35	1229.92	3	8	2	4
	1066	421.94	447.72	652.32	740.35	1229.91	2	5	1	3
200	19	2605.08	2698.90	1753.48	1340.25	2036.29	4	3	8	4
	2106	637.42	2698.89	1753.48	1340.23	2036.29	1	3	1	0
250	13	2243.78	2333.79	2225.06	2156.02	2068.76	4	2	5	2
	2554	1127.81	2333.79	2225.05	2156.02	2068.76	1	1	5	1
300	17	9907.25	10241.77	9472.58	9288.03	3833.98	4	5	4	4
	2351	2453.82	10241.77	9472.56	9288.03	3833.98	2	5	1	1
350	14	7457.23	6033.65	7719.24	5832.25	4588.18	4	3	6	1
	3343	4116.71	6033.64	7719.24	5832.24	4588.18	1	1	6	0
400	17	27087.16	7984.39	6660.77	13944.79	28187.11	4	5	4	4
	3793	6091.49	7984.38	6660.77	13944.79	28187.11	2	3	3	2
450	14	46637.79	19404.19	48468.68	20547.42	13440.55	4	5	3	2
	4184	10017.06	19404.19	48468.67	20547.42	13440.54	2	4	2	1
500	13	111242.78	46243.40	24403.58	61825.45	115840.42	2	3	6	2
	3579	23015.55	46243.40	24403.58	61825.41	115840.42	2	1	5	1
q =4	14.3	20857.34	9557.71	10150.31	11579.76	17138.85				
	2493.3	4796.58	9557.70	10150.31	11579.76	17138.85				

Tabela 5.20: Rezultati za 4 VNS, FI, komunikacije na k_{max}

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	9	17.81	8.70	17.38	9.20	18.41	2	2	2	3
	722	7.92	8.70	17.38	9.20	18.40	0	2	1	0
100	11	313.56	142.69	327.06	106.99	145.79	2	1	4	4
	1218	71.89	142.69	327.06	106.99	145.79	2	1	1	1
150	16	1883.25	515.74	1967.27	1056.17	991.82	4	1	3	8
	1350	478.85	515.74	1967.27	1056.16	991.80	4	0	0	5
200	9	5038.86	1419.80	5241.59	1243.82	1122.93	3	1	3	2
	2002	632.02	1419.80	5241.58	1243.82	1122.93	1	0	3	2
250	11	12238.06	2731.70	12615.71	3314.13	3038.11	4	1	3	3
	2503	2509.27	2731.70	12615.71	3314.13	3038.10	3	0	0	3
300	11	26698.06	2596.82	27805.67	6893.10	12050.62	4	1	3	3
	2949	2041.37	2596.82	27805.67	6893.10	12050.61	4	0	2	1
350	10	39427.29	5595.24	40612.91	10059.03	5845.41	2	1	3	4
	3325	4072.02	5595.24	40612.91	10059.03	5845.41	2	0	2	3
400	13	121804.00	11607.79	126286.60	10771.18	15190.03	6	1	4	2
	3774	7971.50	11607.78	126286.60	10771.18	15190.02	1	1	2	1
450	11	108207.92	36983.12	112074.71	31482.85	74371.50	4	1	2	4
	3064	10094.90	36983.12	112074.70	31482.84	74371.48	0	1	1	4
500	13	396284.95	61035.97	411921.16	18120.82	67932.69	3	1	4	5
	2968	16776.19	61035.96	411921.16	18120.82	67932.67	3	1	0	4
q =4	11.4	71191.38	12263.76	73887.01	8305.73	18070.73				
	2387.5	4465.59	12263.76	73887.00	8305.73	18070.72				

Tabela 5.21: Rezultati za 5 VNS, BI, komunikacije na k_{max}

n	n.it.	CPU time						Poboljšanja				
	f_{min}	Comput time						Novo najbolje				
50	7	36.52	14.96	37.52	8.78	28.36	17.16	2	2	1	1	1
	768	7.81	14.96	37.52		8.78	17.16	2	2	1	1	1
100	15	342.72	174.42	109.97	114.11	134.98	352.71	4	5	2	2	2
	1189	76.65	174.42	109.97	114.11	134.98	352.71	1	5	1	1	0
150	18	1459.36	447.18	652.56	741.59	1229.82	1502.92	3	7	2	4	2
	1066	403.02	447.18	652.56	741.59	1229.82	1502.92	2	5	1	3	1
200	20	6360.25	2716.37	1753.44	1340.73	2035.37	6555.54	4	3	7	4	2
	2101	610.83	2716.37	1753.44	1340.73	2035.37	6555.53	1	3	1	0	1
250	14	4746.10	2332.68	2224.27	2156.55	2068.48	4875.00	4	2	5	2	1
	2554	1135.25	2332.68	2224.27	2156.55	2068.48	4874.99	1	1	5	1	0
300	19	9926.31	10240.80	9473.81	2536.41	3832.83	4372.37	3	5	4	4	3
	2562	2442.13	10240.80	9473.81	2536.40	3832.83	4372.37	1	5	1	1	1
350	15	7502.91	6035.51	7719.27	5833.39	4591.38	6668.87	4	3	6	1	1
	3343	4136.38	6035.51	7719.27	5833.39	4591.38	6668.87	1	1	6	0	0
400	18	27058.69	7985.05	6661.58	13936.03	28197.68	20543.40	4	5	4	4	1
	3793	6108.16	7985.05	6661.58	13936.03	28197.68	20543.40	2	3	3	2	0
450	16	46699.38	19370.97	48475.44	20542.54	13435.71	14952.65	4	5	3	2	2
	4184	10125.21	19370.96	48475.44	20542.54	13435.71	14952.65	2	4	2	1	1
500	14	112143.23	46234.20	24388.51	61798.61	115836.37	114887.60	2	3	6	2	1
	3579	23119.45	46234.20	24388.50	61798.59	115836.36	114887.60	2	1	5	1	1
q =5	15.6	21627.55	9555.21	10149.64	10900.87	17139.10	17472.82					
	2513.9	4816.49	9555.21	10149.64	10900.87	17139.10	17472.82					

Tabela 5.22: Rezultati za 5 VNS, FI, komunikacije na k_{max}

n	n.it.	CPU time						Poboljšanja				
	f_{min}	Comput time						Novo najbolje				
50	15	22.09	8.17	17.42	21.40	24.16	18.55	2	2	3	5	3
	722	7.47	8.17	17.42	21.40	24.15	18.55	1	2	0	1	0
100	12	309.79	142.89	327.35	107.05	145.98	232.40	2	1	4	4	1
	1218	70.62	142.88	327.35	107.05	145.97	232.39	2	1	1	1	0
150	22	1876.21	517.17	1971.10	1057.06	988.59	596.15	4	1	3	8	6
	1350	481.23	517.16	1971.10	1057.06	988.59	596.14	2	0	0	2	4
200	11	9852.20	934.80	10231.46	1484.05	1123.19	2062.87	3	1	3	2	2
	1968	701.62	934.80	10231.45	1484.05	1123.19	2062.86	1	0	0	1	2
250	14	11984.69	2729.97	12659.39	3301.62	3040.07	2627.40	4	1	3	3	3
	2503	2399.60	2729.97	12659.39	3301.62	3040.07	2627.40	3	0	0	3	0
300	13	26774.47	2601.86	27844.88	6896.83	12056.21	4145.92	4	1	3	3	2
	2949	2061.16	2601.86	27844.88	6896.82	12056.21	4145.92	4	0	2	1	1
350	11	85623.65	6167.18	88993.95	5330.22	5857.38	5478.64	2	1	2	4	2
	3325	4072.27	6167.18	88993.95	5330.22	5857.36	5478.61	2	0	0	3	2
400	18	82303.99	7009.30	85586.89	9518.91	7353.19	9991.27	6	2	4	2	4
	3777	6616.63	7009.30	85586.88	9518.91	7353.18	9991.27	1	0	0	1	1
450	15	107960.06	12334.33	112190.96	13330.13	74355.75	76417.16	4	1	1	4	5
	3064	10098.23	12334.32	112190.95	13330.13	74355.74	6417.15	0	1	0	3	3
500	16	394402.04	61058.69	412061.43	18121.27	67932.19	109401.05	3	1	4	5	3
	2968	16610.98	61058.69	412061.42	18121.26	67932.19	109401.03	3	1	0	4	1
q =5	14.7	72110.92	9350.44	75188.48	5916.85	17287.67	21097.14					
	2384.4	4311.98	9350.43	75188.48	5916.85	17287.67	21097.13					

Kao što se vidi iz navedenih tabela, odstupanje dobijenih rešenja od poznatih optimalnih je veliko (oko 65%), a dodavanje nove varijante nema nikakvih pozitivnih efekata. Time se potvrđuje zaključak dobijen u analizi sekvencijalne varijante da je VNS-VND kombinacija loša zbog veličine okoline koja se pretražuje i složenosti postupka za preračunavanje dužine raspodele transformisanih rešenja koja nastaju prilikom lokalnog pretraživanja.

U većini slučajeva FI strategija poboljšavanja daje bolje rezultate, što se takođe poklapa sa sekvencijalnim izvršavanjem. Na osnovu broja iteracija i broja popravki može se zaključiti koje od varijanti VNS metode najviše doprinose konačnom rešenju. Na osnovu toga, kombinovane su VNS1 i VNS2 u obe varijante komunikacija da bi se popravili rezultati kooperacije. Dobijeni rezultati prikazani su u tabeli 5.23. U sledećoj tabeli navedeni su rezultati za istu kombinaciju metoda, ali dobijeni za duže vreme izvršavanja. Poboljšanja su evidentna. Pri tome naravno, VNS1 i VNS2 koje se obraćaju nadređenom procesoru prilikom svakog uvećanja indeksa okoline više doprinose kako ukupnom broju iteracija tako i broju popravki i pronalaženju novog najboljeg rešenja. U tabelama B9 do B12, datim u prilogu B, prikazani su rezultati još nekih kombinacija koje su formirane od postojećih varijanti VNS metoda.

Upoređivanje kooperativnog rada sa sekvencijalnim izvršeno je indirektno. Generisano je tzv. nezavisno izvršavanje (independent run) IVNS, kod kojega se istovremeno na svakom procesoru izvršava po jedna od navedenih varijanti VNS metode i na kraju se najbolji dobijeni rezultat proglasi za konačan. Ova implementacija omogućila je da se kooperativni rad upoređuje sa sekvencijanim kao i da se analizira značaj razmene podataka tokom izvršavanja. Rezultati nezavisnog izvršavanja VNS metoda dati su u tabelama B13 do B18 pri čemu su u poslednje dve tabele navedeni rezultati dobijeni za četvorostruko duže vreme kako bi se omogućilo poređenje sa sekvencijalnim izvršavanjem u odnosu na kriterijum koji zavisi ne samo od vremena izvršavanja, nego uzima u obzir broj angažovanih procesora.

Opis rezultata, koje sadrže tebele B13 do B18, dat je u prilogu B, a ovde je prikazana uporedna tabela (tabela 5.25) za nezavisna izvršavanja.

U prvoj koloni naveden je broj zadataka u primerima koji se raspoređuju, a u preostale četiri kolone date su najkraće dužine raspodele i varijante VNS metode koje su ih generisale. Kao što se iz date tabele vidi, najveći broj najboljih rešenja ima varijanta VNS2, koja za duže vreme izvršavanja daje referentni podatak za poređenje, tj. prosečnu dužinu najbolje dobijene raspodele na 10 primera $SL_{min} = 1979.30$.

Tabela 5.23: Rezultati za 4 najbolje VNS metode (VNS1 i VNS2), FI, komunikacije u obe tačke (k++ i k_{max})

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	92	7.95	7.84	7.91	6.91	8.05	66	16	5	5
	734	7.58	7.84	7.91	6.91	8.05	20	8	3	5
100	133	89.80	90.90	89.86	90.30	91.22	100	21	5	7
	1019	87.75	90.90	89.86	90.29	91.21	21	7	2	4
150	283	509.46	515.37	518.35	514.43	524.01	228	45	5	5
	1198	501.15	515.35	518.32	514.43	524.01	84	21	3	2
200	179	787.59	779.92	813.27	776.23	784.88	138	35	3	3
	2011	751.25	779.91	813.27	776.23	784.87	64	18	3	1
250	191	1275.14	1297.80	1315.40	1287.29	1300.51	138	46	2	5
	2434	1252.87	1297.75	1315.39	1287.29	1300.50	73	22	2	4
300	219	2631.43	2584.09	2583.71	2586.33	2696.12	166	43	5	5
	2928	2508.75	2584.03	2583.70	2586.29	2696.11	87	26	3	3
350	264	5055.79	5150.53	5184.97	5143.75	5189.50	204	50	5	5
	3363	5005.56	5150.49	5184.97	5143.72	5189.48	119	27	4	4
400	281	7726.19	7720.87	7935.90	7774.49	7903.33	207	57	9	8
	3693	7511.88	7720.84	7935.89	7774.49	7903.33	120	33	4	5
450	216	14041.93	12956.44	13063.71	12898.81	14434.28	171	28	7	10
	2894	12543.69	12956.40	13063.71	12898.80	14434.28	70	16	1	5
500	261	20919.81	20554.27	20726.25	20591.43	21498.82	199	39	16	7
	3062	20008.49	20554.22	20726.25	20591.34	21498.79	88	22	7	4
q =4	211.9	5304.51	5165.80	5223.93	5167.00	5443.07				
	2333.6	5017.90	5165.77	5223.93	5166.98	5443.06				

Tabela 5.24: Rezultati za 4 najbolje VNS metode (VNS1 i VNS2), FI, komunikacije u obe tačke (k++ i k_{max}) za duže vreme izvršavanja

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	330	30.92	30.69	30.69	30.06	31.74	259	61	5	5
	675	30.06	30.68	30.69	30.05	31.74	56	15	2	3
100	496	352.71	364.07	364.75	363.49	363.24	395	80	6	15
	960	350.21	363.98	364.74	363.49	363.23	79	18	1	6
150	976	2017.14	2065.58	2076.34	2077.49	2087.91	801	169	3	3
	1240	2000.14	2065.43	2076.30	2077.45	2087.89	206	49	3	2
200	560	3220.23	3119.29	3331.94	3110.06	3151.07	454	93	5	8
	1334	3007.08	3119.23	3331.92	3110.00	3151.06	144	38	4	3
250	812	5104.90	5147.89	5266.12	5159.13	5177.68	603	199	3	7
	2459	5006.06	5147.81	5266.09	5159.05	5177.65	310	91	3	7
300	821	10097.85	10325.28	10385.15	10339.58	10390.55	638	173	6	4
	2893	10026.07	10325.19	10385.14	10339.50	10390.54	296	84	3	3
350	936	20136.94	20598.41	20673.33	20593.82	20616.43	754	167	7	8
	2595	20032.16	20598.26	20673.29	20593.77	20616.40	372	76	4	6
400	704	30104.22	30861.05	30908.47	30792.15	30872.40	558	122	10	14
	2194	30006.69	30860.97	30908.46	30792.08	30872.38	212	57	9	9
450	781	51785.03	51415.73	53103.34	51344.84	52777.11	672	92	7	10
	2855	50077.83	51415.67	53103.33	51344.81	52777.10	179	33	1	5
500	725	80173.53	82281.98	82117.89	82269.63	82410.12	609	89	19	8
	2904	80064.62	82281.85	82117.88	82269.49	82410.04	111	31	10	5
q =4	714.1	20302.35	20621.00	20825.80	20608.03	20787.83				
	2010.9	20060.09	20620.91	20825.78	20607.97	20787.80				

Tabela 5.25: Uporedni rezultati nezavisnih VNS metoda

n	SL(VNS_i)			
	4-BI	4-FI	5-BI	5-FI
50	766(VNS_2)	722(VNS_2)	796(VNS_2)	722(VNS_2)
100	1189(VNS_2	1102(VNS_2)	1189(VNS_2)	1262(VNS_3)
150	1185(VNS_2)	1356(VNS_1)	1224(VNS_2)	1356(VNS_1)
200	1744(VNS_2)	2002(VNS_3)	2106(VNS_2)	1968(VNS_5)
250	2547(VNS_3)	2506(VNS_1)	2554(VNS_3)	2506(VNS_1)
300	2562(VNS_2)	2949(VNS_1)	2562(VNS_2)	2949(VNS_1)
350	3350(VNS_3)	3340(VNS_4)	3350(VNS_2)	3342(VNS_5)
400	3788(VNS_2)	2633(VNS_2)	3803(VNS_2)	3696(VNS_4)
450	3949(VNS_2)	2889(VNS_4)	4170(VNS_5)	3064(VNS_4)
500	3273(VNS_2)	3324(VNS_3)	4538(VNS_4)	3274(VNS_5)
av. SL	2435.30(VNS_2)	2282.30($VNS_{1,2}$)	2616.00(VNS_2)	2413.90($VNS_{1,5}$)

Rezultati poređenja kooperativnih verzija VNS metode dati su u uporednim tabelama 5.26 i 5.27, dok je poređenje nezavisnog izvršavanja i najboljih kombinacija kooperativnog rada sa sekvencijalnim izvršavanjem u četvorostrukom vremenu dato u tabeli 5.28. Kao što se vidi iz navedenih tabela pogodno izabrane kombinacije u kooperativnom radu daju bolje rezultate nego nezavisno izvršavanje i veoma su bliske najboljem sekvencijalnom izvršavanju za proporcionalno duže vreme. Sekvencijalno izvršavanje dalo je za nijansu bolje rezultate, ali treba imati na umu da je tu izabrana najbolja od 5 testiranih varijanti. Važno je napomenuti da su sve ostale varijante bile lošije od kooperativnog rada (tabela B17), a proceniti koja od varijanti će dati najbolje rezultate nije nimalo jednostavno.

Na slici 5.12, kao ilustracija, dat je grafik poboljšanja tekućeg minimalnog rešenja u vremenu, za nekoliko kombinacija kooperativnog rada VNS metoda na primeru sa $n = 200$ zadataka. Za ostale primere grafici su vrlo slični. Kod nezavisnog izvršavanja (koje je prikazano punom linijom) ne može se pratiti popravka rešenja tokom izvršavanja, jer nema razmene podataka sve do kraja izvršavanja. Zato je grafik za IVNS ravna linija paralelna sa vremenskom osom na visini koja odgovara dužini polazne (CPES) raspodele do kraja izvršavanja, koja se tada naglo spušta do konačnog rešenja. Kod ostalih metoda, poboljšanja se evidentiraju u komunikacionim tačkama.

Tabela 5.26: Rezultati poređenja kooperativnog rada 4 VNS metode

n	SL(VNS_i)			
	4-FI, $k++$	4-FI, k_{max}	4-BI, $k++$	4-BI, k_{max}
50	699(VNS_1)	722(VNS_2)	767(VNS_1)	768($\text{VNS}_{1,2}$)
100	1227($\text{VNS}_{3,4}$)	1218(VNS_1)	1283(VNS_1)	1189(VNS_2)
150	1336(VNS_1)	1350(VNS_4)	1053(VNS_1)	1066(VNS_2)
200	2128(VNS_1)	2002(VNS_3)	2111(VNS_1)	2106(VNS_2)
250	2505(VNS_1)	2503($\text{VNS}_{1,4}$)	2540(VNS_1)	2554(VNS_3)
300	2359(VNS_1)	2949(VNS_1)	2222(VNS_3)	2351(VNS_2)
350	3331(VNS_1)	3325(VNS_4)	3370(VNS_1)	3343(VNS_3)
400	3777(VNS_1)	3774(VNS_3)	3728(VNS_1)	3793($\text{VNS}_{2,3}$)
450	3938(VNS_1)	3064(VNS_4)	4151(VNS_1)	4184(VNS_2)
500	3470(VNS_1)	2968(VNS_4)	3428(VNS_1)	3579(VNS_3)
av.	2477.00(VNS_1)	2387.50(VNS_4)	2465.30(VNS_1)	2493.30(VNS_2)

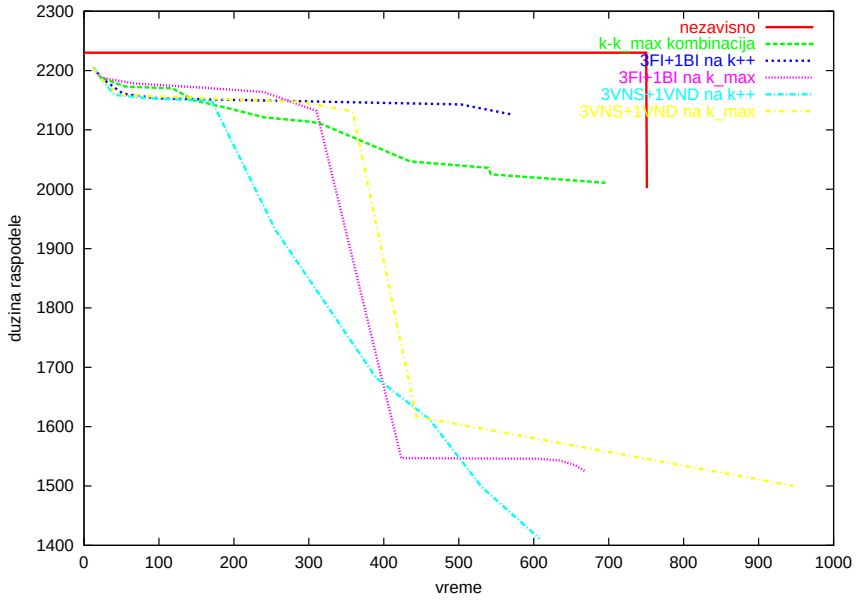
Na osnovu ranijih iskustava sa paralelnim izvršavanjem metaheurističkih metoda [26, 28], isprobane su još neke strategije. Na primer, da se novo najbolje rešenje ne prosleđuje svim procesorima nego se šalje samo na zahtev. To podrazumeva sinhronizaciju nadređenog procesora sa odgovarajućim podređenim i stoga daje lošije rezultate. Drugi slučaj baziran je na ideji da se ne šalje uvek najbolje rešenje, nego da se svakom podređenom procesoru, na zahtev (tj. u slučaju da nema popravke trenutno važećeg rešenja) pošalje slučajno izabrano iz baze dobrih rešenja koja čuva nadređeni procesor. Ideja potiče od težnje da se ispita što je moguće više dobijenih rešenja, ne bi li se proširio prostor pretrage i time omogućilo dobijanje kvalitetnijih konačnih rešenja. Ova implementacija takođe podrazumeva sinhronizaciju dva procesora koji komuniciraju, a sem toga slučajnost koja se tom prilikom uvodi je isuviše velika da bi dala pozitivne rezultate. Dodatna kontrola, kojom bi se obezbedila sistematičnost u pretraživanju, mogla bi eventualno da obezbedi poboljšanje dobijenih rezultata. Na primer, favorizuje se najbolje rešenje, a ostala se šalju sa nekom verovatnoćom (< 0.5) i/ili uvodi se statistika o tome koliko puta (i do koje okoline) je svako od rešenja iskorišćeno, tako da se pri narednom izboru nekog od rešenja uzimaju u obzir i odgovarajući parametri te statistike.

Tabela 5.27: Rezultati poređenja kooperativnog rada 5 VNS metoda

n	SL(VNS_i)			
	5-FI, $k++$	5-FI, k_{max}	5-BI, $k++$	5-BI, k_{max}
50	699(VNS_2)	722(VNS_2)	767(VNS_1)	768($VNS_{1,2}$)
100	1191(VNS_2)	1218(VNS_1)	1283(VNS_1)	1189(VNS_2)
150	1095(VNS_2)	1350(VNS_5)	1053(VNS_1)	1066(VNS_2)
200	2095(VNS_2)	1968(VNS_5)	2111(VNS_1)	2101(VNS_2)
250	2508(VNS_3)	2503($VNS_{1,4}$)	2540(VNS_1)	2554(VNS_3)
300	2940(VNS_2)	2949(VNS_1)	2405(VNS_3)	2562(VNS_2)
350	3342(VNS_2)	3325(VNS_4)	3370(VNS_1)	3343(VNS_3)
400	3773(VNS_2)	3777($VNS_{1,4,5}$)	3728(VNS_1)	3793($VNS_{2,3}$)
450	4046(VNS_5)	3064($VNS_{4,5}$)	4151(VNS_1)	4184(VNS_2)
500	3042(VNS_4)	2968(VNS_4)	3452(VNS_1)	3579(VNS_3)
av.	2473.10(VNS_1)	2384.40(VNS_4)	2486.00(VNS_1)	2513.90(VNS_2)

Tabela 5.28: Rezultati poređenja kooperativnog rada sa nezavisnim i sekvencijalnim izvršavanjem

n	SL(VNS_i)			
	IVNS	3FI+1BI, k_{max}	3VNS+1VND, k_{max}	BEST SEQ. ($t*4$)
50	722(VNS_2)	734(VNS_2)	734(VNS_2)	730(VNS_2)
100	1102(VNS_2)	1085(VNS_2)	1096(VNS_2)	931(VNS_2)
150	1356(VNS_1)	1147(VNS_4)	1460(VNS_2)	1204(VNS_2)
200	2002(VNS_3)	1525(VNS_1)	1500(VNS_4)	1588(VNS_2)
250	2506(VNS_1)	2511(VNS_1)	2511(VNS_1)	2569(VNS_2)
300	2949(VNS_1)	2895(VNS_2)	2072(VNS_2)	2102(VNS_2)
350	3340(VNS_4)	3342($VNS_{3,4}$)	3351(VNS_2)	2412(VNS_2)
400	2633(VNS_2)	2633(VNS_2)	2633($VNS_{1,2}$)	2279(VNS_2)
450	2889(VNS_4)	2715(VNS_1)	2679(VNS_1)	3000(VNS_2)
500	3324(VNS_3)	3035(VNS_1)	3315(VNS_1)	2978(VNS_2)
av.	2282.30 ($VNS_{1,2}$)	2162.20 ($VNS_{1,2}$)	2135.10 (VNS_2)	1979.30 (VNS_2)



Sl. 5.12: Poređenje kooperativnog rada VNS metoda za primer sa n=200

5.7.4. Eksperimenti sa PVNS-om

Poslednja varijanta paralelizacije VNS metode, koja je implementirana u ovom radu, je distribuirano izvršavanje i nazvana je paralelni VNS (PVNS). Za ovu varijantu može se reći da je po karakteristikama negde između prethodne dve. Sastoji se u tome da se na različitim procesorima istovremeno izvršava kombinacija koraka razmrdavanja u k-toj okolini i odgovarajućeg lokalnog pretraživanja za različite vrednosti parametra k. Dakle, izvršava se jedinstvena VNS procedura kao kod PVNSPLS, ali sa krupnijom granulacijom za paralelizaciju.

Implementacija podrazumeva $q + 1$ procesora, pri čemu nadređeni vodi računa o indeksu okoline za razmrdavanje, čuva najbolja rešenja i proverava kriterijum zaustavljanja. Ostatak VNS procedure obavljaju podređeni procesori. Komunikacija je asinhrona i podrazumeva razmenu rešenja i indeksa okoline za razmrdavanje. Očigledno je da kvalitet rešenja zavisi od broja procesora i da se paralelni rezultati razlikuju od sekvencijalnih za bilo koju kombinaciju parametara.

U okviru izvršenih eksperimenata varirani su parametri FI-BI, k_{max}, k_{step}, *plato*, kao i okoline za LS (Svap-1, VND). U svim eksperimentima kriterijum zaustavljanja je maksimalno dozvoljeno vreme izvršavanja. Prva tabela sa rezultatima je tabela 5.29. U ovoj tabeli navedeni su rezultati dobijeni pretraživanjem Swap-1 okoline unapred do prvog poboljšanja, pri čemu je za razmrdavanje takođe korišćena Swap okolina. Početno rešenje je CPES heuristično.

U prvoj koloni naveden je broj podređenih procesora, a u naredne dve rezultati raspoređivanja "teških" primera sa poznatim optimalnim rešenjem za dve vrednosti parametra k_{max}. Prva vrednost je fiksirana, a druga zavisi od dimenzije primera. U oba slučaja primećuje se tendencija poboljšanja rezultata sa porastom broja procesora u početku, međutim, kada se broj procesora isuviše poveća uticaj slučajnosti postaje izraženiji i rezultati ne pokazuuju nikakvu pravilnost.

Tabela 5.29: Rezultati PVNS, FI, FOR, ls_swap, sh_swap

q	SL_{av}	
	$k_{max} = 10$	$k_{max} = n/2$
1	2640.90	2605.80
2	2565.30	—
3	2559.20	—
4	2482.00	—
5	2375.50	2450.10
6	2571.20	—
7	2572.60	—
8	2523.60	—
9	2599.30	—
10	2592.20	2528.70
15	2598.70	2402.40
20	2498.10	2516.70

Detaljni rezultati u slučaju $q = 5$ i $k_{max} = 10$ prikazani su u tabeli 5.30. Ova tabela sadrži iste podatke kao i tabele za CVNS u prethodnom odeljku.

Tabela 5.30: Rezultati za PVNS na $q = 5$ procesora

n	n.it.	f_{min}	CPU time Comput time						Improvements Bests				
50	43	738	6.60	6.75	6.67	6.44	6.51	6.45	1	1	1	3	1
			6.31	5.85	5.77	5.55	5.67	5.56	0	1	1	3	1
100	81	1256	79.84	79.11	76.78	73.16	72.89	73.96	1	2	1	2	2
			70.04	75.91	73.51	69.97	69.66	70.74	0	1	0	1	2
150	137	1251	419.85	432.73	428.74	420.82	421.96	438.34	3	3	2	5	3
			402.93	421.62	418.52	409.67	411.54	427.48	2	1	0	5	3
200	80	2003	641.82	625.53	637.32	635.86	663.95	631.54	2	3	3	2	3
			602.60	590.74	603.60	601.67	629.57	596.81	2	2	2	0	1
250	134	2507	1065.72	1067.03	1066.90	1065.33	1097.44	1087.04	2	2	3	3	5
			1031.95	1036.55	1036.13	1035.45	1066.77	1055.92	2	1	3	2	4
300	130	2920	2073.09	2099.14	2099.14	2111.17	2125.04	2096.33	3	1	1	2	1
			2001.89	1958.30	1958.26	1968.98	1986.78	1953.17	3	1	1	1	0
350	101	3342	4212.05	4196.72	4231.98	4370.83	4254.66	4254.68	3	4	3	3	2
			4023.89	3830.13	3871.54	4005.03	3893.54	3894.96	2	4	2	2	1
400	139	3743	8150.06	6565.69	6482.84	8342.26	8471.98	6703.03	3	2	2	2	4
			6117.30	6394.82	6308.47	8170.75	8301.55	6528.21	2	2	1	2	4
450	133	2799	12428.26	12131.31	10706.70	11214.65	12859.78	10635.40	4	4	2	4	3
			10095.55	11528.05	10102.92	10614.98	12263.86	10034.39	3	2	1	4	3
500	145	3196	18087.61	16768.50	17773.06	16664.86	18636.44	16887.88	4	4	4	3	5
			16109.17	16416.31	17419.84	16309.54	18284.75	16537.32	4	4	2	2	4
q =5	112.30	2375.50	4716.49	4397.25	4351.01	4490.54	4861.07	4281.46					
			4046.16	4225.83	4179.86	4319.16	4691.37	4110.46					

Broj započetih VNS iteracija je, kao što se može videti u drugoj koloni, izuzetno veliki. Obzirom na mali broj uspešnih lokalnih pretraživanja (podaci u poslednjim kolonama), može se zaključiti da je k_{max} dostignut mnogo puta bez popravke trenutno najboljeg rešenja. Osim toga, tabela 5.30 pokazuje da je kod ove strategije paralelizacije postignut dobar balans opterećenja između procesora. To je posledica relativno male granulacije modula koji se izvršavaju između komunikacija kao i asinhronog izvršavanja. Međutim, kao i kod sekvencijalnog izvršavanja, u ovoj kombinaciji parametara odstupanje dobijene prosečne dužine raspodele od prosečne optimalne je i dalje veliko: $1 - \frac{SL_{av}}{SL_{opt}} = 58.37\%$.

Stoga su dalji eksperimenti vršeni sa smanjenim okolinama za pretraživanje (kako je to opisano u glavi 4.). Svaki od procesora izvršava pretraživanje nekoj podokolini Swap-1 okoline. Korišćena su smanjenja koja su dala najbolje rezultate u sekvencijalnom slučaju:

–(RVNS): premeštanje zadataka dodeljenih najopterećenijem procesoru i

–(RCVNS): premeštanje prethodnika zadatka koji zahteva najviše komunikacija.

U tabeli 5.31 dati su neki rezultati dobijeni prilikom pretraživanja po smanjenim okolinama za različite vrednosti parametra k_{max} i za različit broj podređenih procesora. Navedene su samo prosečne dužine dobijenih najboljih raspodela.

Tabela 5.31: Rezultati za paralelni RVNS i RCVNS

q	SL_{av}			
	PRVNS		PRCVNS	
	$k_{max} = 10$	$k_{max} = n/2$	$k_{max} = 10$	$k_{max} = n/2$
1	2718.00	2607.30	2414.90	2401.00
5	2716.20	2366.90	2330.10	2187.30
10	2624.50	2326.80	2364.00	2291.80
15	2568.00	2173.70	2332.10	2200.70
20	2663.70	2187.50	2348.70	2155.90

Kao što se iz tabele 5.31 može zaključiti rezultati su bolji kod varijante koja smanjuje okoline na osnovu komunikacionog kriterijuma i ukoliko se dozvoli veći broj okolina za razmrdavanje.

Da bi se izvršila detaljnija analiza dobijenih rezultata i adekvatno poređenje sa sekvencijalnim izvršavanjem (ono koje uključuje i kriterijum vremena izvršavanja), vršena je raspodela skupa primera sa poznatim optimalnim rešenjem, sa $n = 200$ zadataka i različitim gustinama veza među zadacima.

U tabeli 5.32 prikazani su rezultati raspoređivanja primenom najbolje paralelne varijante VNS metode za konkretne vrednosti parametara. U prvoj koloni dat je broj zadataka, u drugoj gustina veza među zadacima, treća kolona sadrži broj započetih VNS iteracija, a četvrta minimalnu dobijenu dužinu raspodele. Optimalna raspodela ima dužinu $SL_{opt} = 1200$ ciklusa izračunavanja u svih devet korišćenih primera. U kolonama 5–10 dato je vreme izvršavanja i vreme izračunavanja na osnovu kojih se potvrđuje dobra balansiranost opterećenja procesora. Poslednjih pet kolona sadrži učinak svakog procesora u popravljanju tekućeg minimalnog rešenja tokom paralelnog izvršavanja. U odnosu na broj iteracija, broj prijavljenih popravki je mali, a od toga je oko 70% uspešnih, tj. onih koje su dovele do prihvaćenog novog minimuma.

Odstupanje dobijenog heurističkog rešenja od optimalnog je nešto ispod 20% u proseku, što je lošije od rezultata sekvencijanog izvršavanja prikazanih u glavi 4. Međutim, rezultati nisu uporedivi jer su za izvršavanja korišćeni različiti parametri i procesori različitih karakteristika. Stoga je ispitan uticaj broja procesora na kvalitet dobijenog rešenja i ilustrovan podacima sadržanim u tabeli 5.33. Vršeno je 10 ponavljanja (sa različitom početnom vrednošću generatora slučajnih brojeva) za različit broj procesora i u tabeli su prikazani rezultati prvog izvršavanja, srednja vrednost svih 10 izvršavanja i, u poslednjoj koloni, najbolji dobijeni rezultat.

Kao što se iz rezultata prikazanih u tabeli 5.33 vidi, kvalitet rešenja popravlja se sa dodavanjem novih procesora. Međutim, to nije dovoljan zaključak jer je dozvoljeno vreme izvršavanja u svim slučajevima isto. Stoga je praćena popravka trenutno najboljeg rešenja u vremenu i prikazana grafički na slikama 5.13 i 5.14. Na ovim slikama može se videti da je poboljšanje izuzetno značajno na samom početku izvršavanja, a kasnije se malo toga menja. Ovaj zaključak važi i u sekvencijalnom slučaju, ali je izraženiji u paralelnom, jer više procesora prijavljuje poboljšanja i uzima se najbolje od njih.

Dozvoljeno vreme izvršavanja u svim slučajevima je bilo $t_{max} = 600s$. Da bi se uveo vremenski kriterijum u poređenje dobijenih rezultata, vertikalnim linijama obeleženi su proporcionalno manji intervali vremena.

Tabela 5.32: Rezultati PRCVNS, za $q = 5$, n=200, $k_{max} = 10$

n	ρ	n.it.	f_{min}	CPU time						Improvements				
				Comput time						Bests				
200	10	28370	1504	750.32	797.31	797.49	782.85	794.29	791.93	3	2	1	6	5
				750.08	758.51	761.97	746.01	757.52	753.93	2	1	1	4	4
200	20	31732	1799	750.26	778.94	780.94	784.76	784.41	778.16	4	5	2	4	6
				750.00	761.70	762.99	767.02	767.84	760.38	3	4	2	3	6
200	30	41119	1774	750.18	783.95	795.95	785.90	789.48	791.29	7	6	10	6	4
				750.02	748.15	761.74	748.45	755.24	754.33	5	4	8	5	4
200	40	37729	1520	750.24	794.54	782.37	783.83	784.01	781.98	3	4	5	3	4
				750.02	751.15	742.42	740.22	740.60	739.52	2	3	5	1	3
200	50	36366	1533	750.35	794.89	792.79	804.10	800.68	805.51	7	4	3	8	3
				750.01	745.53	744.63	752.32	750.38	755.95	7	4	3	8	2
200	60	3786	1206	759.88	789.69	775.15	790.45	791.15	791.31	13	10	12	12	14
				750.04	785.55	772.16	786.88	787.94	787.70	12	5	8	8	8
200	70	5538	1210	759.65	788.61	795.73	798.74	798.82	786.80	6	4	4	6	6
				750.05	783.95	790.58	793.09	792.74	781.76	5	2	3	5	5
200	80	1773	1200	762.25	778.86	787.40	771.26	783.04	771.92	7	12	10	4	7
				751.09	776.74	785.73	769.83	781.45	770.79	7	11	10	2	5
200	90	9930	1200	752.73	804.00	799.17	788.52	786.53	791.02	11	12	12	10	13
				750.05	789.21	783.57	774.06	770.42	779.80	9	10	10	9	10
q =5		21815.89	1438.44	753.98	790.09	789.67	787.82	790.27	787.77					
				750.15	766.72	767.31	764.21	767.13	764.91					

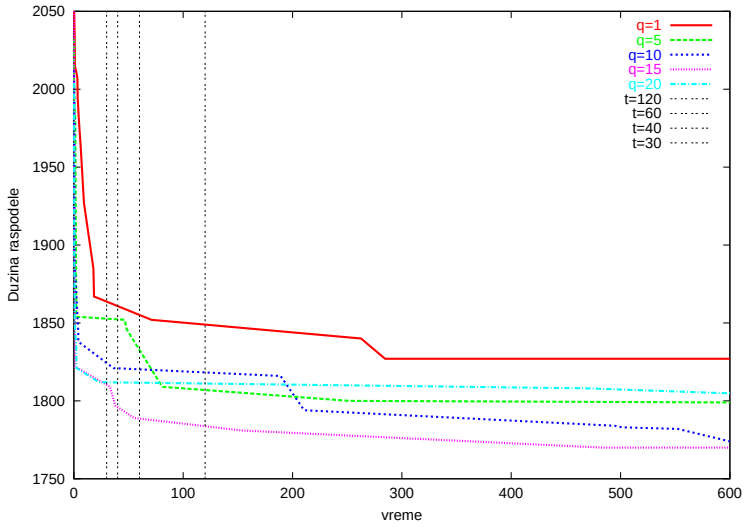
Tabela 5.33: Rezultati PRCVNS za primere različitih gustina i $n = 200$ zadataka

q	prosečna dužina raspodele		
	PRCVNS sa ponavljanjem		
	$k_{max} = n/2$	SL_{av} za 10	SL_{min} od 10
1	1465.9	1465.9	1465.9
5	1417.9	1448.57	1410.44
10	1397.7	1438.42	1395.9
15	1384.8	1418.76	1362.9
20	1383.3	1403.37	1322.9

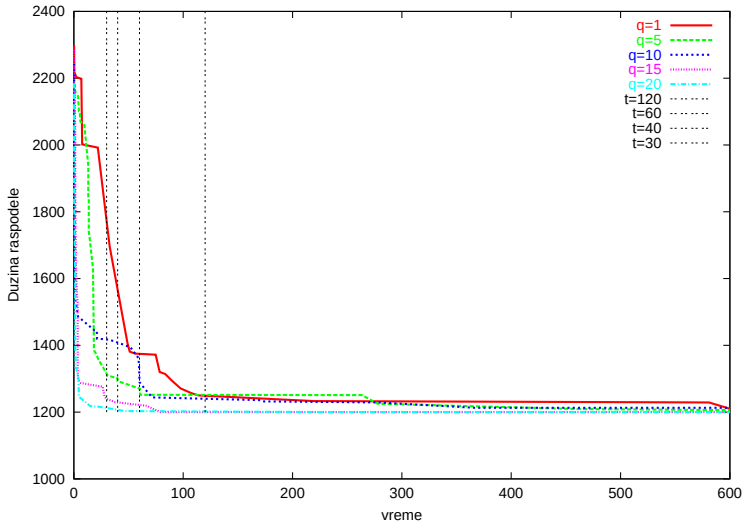
Činjenica da je u svim tačkama preseka odgovarajućih krivih i vremenske vertikale kriva koja odgovara sekvencijalnom izvršavanju iznad odgovarajuće krive za paralelno izvršavanje, upućuje na zaključak da se ovom varijantom paralelne VNS procedure dobija kvalitetnije rešenje u proporcionalno kraćem vremenu izvršavanja.

Strategija paralelizacije koja podrazumeva istovremeno izvršavanje kombinacije koraka razmrdavanja u različitim okolinama i odgovarajuće procedure lokalnog pretraživanja po smanjenim okolinama u odnosu na komunikacioni kriterijum, omogućila je ostvarivanje kombinacije dva postavljena cilja paralelizacije: dobijanje kvalitetnijih rešenja u kraćem vremenu.

Sve implementirane strategije paralelizacije pokazale su se manje ili više uspešnima što ohrabruje dalja istraživanja u tom smeru, bez obzira na to da li su bazirana na idejama predloženim u ovom radu ili nekim potpuno novim.



Sl. 5.13: Poboljšanje u vremenu rezultata dobijenih PRCVNS metodom za primer iz prve grupe sa $n = 200$ i $\rho = 20$



Sl. 5.14: Poboljšanje u vremenu rezultata dobijenih PRCVNS metodom za primer iz prve grupe sa $n = 200$ i $\rho = 60$

6. Završne napomene

U ovom radu razmatran jedan od značajnijih problema u računarstvu: problem statičke paralelizacije za homogene višeprocesorske sisteme. Ovaj problem spada u klasu NP-teških problema kombinatorne optimizacije i stoga su za njegovo rešavanje najpogodnije metaheurističke metode. Naročito je to značajno ako se razmatraju neke složenije varijante problema, kao što je to ovde slučaj: nepotpune veze među procesorima i značajno (nezanemarljivo) vreme koje se troši na komunikaciju. Dati problem uobičajeno se definiše pomoću grafova, a ovde je po prvi put razvijen model na bazi linearnog programiranja sastavljen od celobrojnih 0-1, i od realnih promenljivih. Odlika modela je, osim postojanja necelobrojnih promenljivih, i njihov polinomni broj u odnosu na veličinu problema.

Glavni rezultati rada su implementacija tri poznate metaheurističke metode (VNS, TS, GA) bazirane na istoj reprezentaciji rešenja, što omogućuje poređenje, kako osnovnih verzija metoda, tako i njihovih poboljšanja i kombinacija (hibrida). Obzirom da ne postoje adekvatni zvanični test primeri (benchmarks) koji bi omogućili poređenje heurističkih i metaheurističkih metoda, slučajno generisani grafovi zadataka različitih veličina (broja čvorova) i gustina (broja lukova), kao i oni sa poznatim optimalnim rešenjem, generisani za potrebe ovog rada, predloženi su za zvanične test primere i već se nalaze dostupni preko Interneta. Slučajno generisani skupovi grafova pokazali su se veoma raznovrsnim, tako da među njima ima kako lakih, tako i teških primera za svaku od implementiranih metaheurističkih metoda.

Obzirom da se problem pokazao izuzetno teškim za rešavanje, nekoliko autora je pokušalo sa primenom paralelnih metaheuristika. U okviru ovog rada, analizirane su postojeće paralelne metaheurističke metode i predloženo više strategija za paralelizaciju metode promenljivih okolina (VNS) koje su omogućile poboljšanje rezultata dobijenih sekvencijalnom metodom.

Pored svih analiza sprovedenih u ovom radu, poznato optimalno rešenje za slučajno generisane grafove nije dostignuto u svim slučajevima. Stoga je

ostalo prostora za nastavak istraživanja u ovoj oblasti. Potencijalne ideje bile bi: promena reprezentacije rešenja, dekompozicija polaznog problema (u smislu fiksiranja nekog dela rešenja) i razmatranje potproblema manjih dimenzija, a zatim kombinovanje dobijenih rezultata u rešenje polaznog problema velikih dimenzija. Naravno, mogu se primeniti i metode koje u ovom radu nisu razmatrane, kao i dalje poboljšanje nekih već implementiranih metoda.

Osim toga, u okviru ove oblasti, mnogo je otvorenih problema i među zadacima koji ovde nisu razmatrani, kao što je problem granulacije zadataka, mogućnost redundantnih izračunavanja i prekidanja, koji bi u kombinaciji sa raspoređivanjem mogli da dovedu do značajnih poboljšanja. Otvoren i izuzetno složen je i problem klasifikacije grafova zadataka. Rešavanje ovog problema omogućilo bi da se, na osnovu analize polaznog grafa, izabere efikasna (meta)heuristika za njegovo raspoređivanje. Ugrađivanjem ovakvog klasifikatora u postojeći softver, u okviru kojega su implementirane, kako metaheurističke, tako i konstruktivne heurističke metode, dobio bi se ekspertni sistem koji bi povećao efikasnost raspoređivanja.

LITERATURA

[1] *SunTune: Application Performance Optimization on Sun Systems*. Sun Microsystems, 2001.

[2] *Metaheuristics International Conference*. Proceedings, 1997,1999,2001, 2003.

[3] Hertz A., E. Taillard, and D. de Werra. Tabu search. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial optimization*, pages 121–136. John Wiley & Sons Ltd., 1997.

[4] I. Ahmad and M. K. Dhodhi. Multiprocessor scheduling in a genetic paradigm. *Parallel Computing*, 22:395–406, 1996.

[5] I. Ahmad and A. Ghafoor. Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Trans. Software Eng.*, 17:138–153, 1990.

[6] I. Ahmad, A. Ghafoor, and G. C. Fox. Hierarchical scheduling of dynamic parallel computations on hypercube multicomputers. *J. Parallel and Distributed Computing*, 20:317–329, 1994.

[7] S. M. Alaoui, O. Frieder, and T. El-Ghazawi. A parallel genetic algorithm for task mapping on parallel machines. In *10-th International Parallel and Distributed Processing Symposium, Workshop on Bio-Inspired Solutions to Parallel Processing Problems*, page http://ipdsp.eece.unm.edu/1999/biosp3/alaoui.pdf, San Juan, Puerto Rico, April 12–16 1999.

[8] S. Aljančić. *Uvod u realnu i funkcionalnu analizu*. Gradjevinska knjiga, Beograd, 1968.

[9] M Ayed and J-L. Gaudiot. An efficient heuristic for code partitioning. *Parallel Computing*, 26(4):399–426, Mar. 2000. algorithm for determination of inherent parallelism and generation of DAG.

[10] S. K. Baruah. The multiprocessor scheduling of precedence–constrained task systems in the presence of interprocessor communication delays. *Oper. Res.*, 46(1):65–72, Jan.–Feb. 1998. with task duplication.

[11] J. Blazevicz, M. Drabowski, and J. Weglarz. Scheduling independent 2-processor tasks to minimize schedule length. *Information Processing Letters*, 18:267–273, 1984.

[12] J. Blazewicz, M. Drozdowski, and K. Ecker. Management of resources in parallel systems. In J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, editors, *Handbook on Parallel and Distributed Processing*, pages 263–341. Springer, 2000.

[13] E. H. Bowman. The schedule-sequencing problem. *Oper. Res.*, 7(5):621–624, 1959.

[14] W. Bozejko and M. Wodecki. Parallel tabu search method approach for very difficult permutation scheduling problems. In *Proc. Int. Conf. Parallel Computing in Electrical Engineering (PARELEC'04)*, pages 156–161, Dresden, Germany, Sept. 07–10 2004.

[15] P. Brucker. *Scheduling Algorithms*. Springer, 1998.

[16] P. Brucker, S. Knust, D. Roper, and Y. Zinder. Scheduling uet task systems with concurrency on two parallel identical processors. *Mathematical Methods of Operations Research*, 52:369–387, 2001.

[17] J. Bruno, E. G. Coffman Jr, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387, 1974.

[18] G. Caporossi, D. Cvetković, I. Gutman, and P. Hansen. Variable neighborhood search for extremal graphs, 2. finding graphs with extremal energy. *J. Chem. Inform. Comp. Sci.*, 39:984–996, 1999.

[19] N. Carriero and D. Gelernter. *How to Write Parallel Programs*. MIT Press, Cambridge, MA, 1990.

[20] G-I. Chen and T-H. Lai. Preemptive scheduling of independent jobs on a hypercube. *Information Processing Letters*, 28(4):201–206, 1988.

[21] W. W. Chu and L. M-T. Lan. Task allocation and precedence relations for distributed real-time systems. *IEEE Trans. Computers*, C-36(6):667–679, June 1987.

[22] J. Y. Colin and P. Chrétienne. C.P.M. scheduling with small communication delays and task duplication. *Oper. Res.*, 39(4):680–684.

[23] T. Crainic and T. Davidović. Parallel vns for multiprocessor task scheduling. In *Book of Abstracts, Optimization Days*, Montreal, 2002.

[24] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Parallel variable neighborhood search for the p-median. Technical report, GERAD report G-2003-04, 2003.

[25] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative parallel variable neighborhood search for the p-median. *J. Heur.*, 10(3):293–314, 2004.

[26] T. G. Crainic and M. Toulouse. Parallel strategies for meta-heuristics. In F. Glover and G. Kochenberger, editors, *Handbook in Metaheuristics*, pages 475–513. Kluwer Academic Publishers, 2003.

[27] T. G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search algorithms. *INFORMS J. Computing*, 9(1):61–72, 1997.

[28] T.G. Crainic. Parallel computation, co-operation, tabu search. In C. Rego and B. Alidaee, editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, 2002.

[29] Z. Cvetanovic. The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Trans. Computers*, C-36(4):421–432, Apr. 1987.

[30] D. Cvetković and S. Simić. *Odabrana poglavlja iz diskretne matematike*. Akademska misao, Beograd, 2002.

[31] D. Cvetković, S. Simić, G. Caporossi, and P. Hansen. Variable neighborhood search for extremal graphs, 3. on the largest eigenvalue of color-constrained trees. *Linear and Multilinear Algebra*, 49(2):143–160, 2001.

[32] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević-Vujčić, S. Simić, and J. Vuleta. *Kombinatorna optimizacija (Matematička teorija i algoritmi*. DOPIS, Beograd, 1996.

[33] T. Davidović. Hibridizacija metaheurističkih metoda za rešavanje problema raspoređivanja. In *Proc. Symp. on information technology, YUINFO 2005, (on CD 130.pdf)*, Kopaonik.

[34] T. Davidović. Inteligentni paralelizator. In *Zbornik Jug. Simp. YU Info*, Brezovica, 1995.

[35] T. Davidović. Exhaustive list–scheduling heuristic for dense task graphs. *YUJOR*, 10(1):123–136, 2000.

[36] T. Davidović and T. Crainic. Parallelization of local search for scheduling with communications. In *Proc. Symp. on information technology, YUINFO 2004, (on CD 201.pdf)*, Kopaonik.

[37] T. Davidović and T. Crainic. Task allocation and scheduling by cooperative vns procedure. In *Book of Abstracts, INFORMS Joint International Meeting*, San Jose.

[38] T. Davidović and T. Crainic. Parallel vns for scheduling tasks with precedence and communications. In *Book of Abstracts, EURO/INFORMS Joint International Meeting*, page 80, Istanbul, 2003.

[39] T. Davidović and T. G. Crainic. New benchmarks for static task scheduling on homogeneous multiprocessor systems with communication delays. Technical report, Centre de Recherche sur les Transports, CRT-2003-04.

[40] T. Davidović and T. G. Crainic. Parallel local search for vns applied to scheduling communicating tasks on homogeneous multiprocessors. *Centre de Recherche sur les Transports, CRT-2006-??, (submitted for publication)*.

[41] T. Davidović and T. G. Crainic. Parallel vns for scheduling communicating tasks to homogeneous multiprocessors. *Centre de Recherche sur les Transports, CRT-2006-??, (submitted for publication).*

[42] T. Davidović and T. G. Crainic. Benchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systems. *Comput. & OR*, 33(8):2155–2177, Aug. 2006.

[43] T. Davidović, P. Hansen, and N. Mladenović. Scheduling by VNS: Experimental analysis. In *Proc. Yug. Symp. on Oper. Res., SYM-OP-IS 2001*, pages 319–322, Beograd, 2001.

[44] T. Davidović, P. Hansen, and N. Mladenović. Variable neighborhood search for multiprocessor scheduling problem with communication delays. In *Proc. MIC'2001,* 4^th *Metaheuristic International Conference*, pages 737–741, Porto, Portugal, 2001.

[45] T. Davidović, P. Hansen, and N. Mladenović. Permutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delays. *Asia-pacific Journal of Operational Research*, 22(3):297–326, Sept. 2005.

[46] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović. Mathematical programming-based approach to scheduling communicating tasks. *GERAD Tech. Report, G-2004-99, (submitted for publication).*

[47] T. Davidović, N. Maculan, and N. Mladenović. Mathematical programming formulation for the multiprocessor scheduling problem with communication delays. In *Proc. Yug. Symp. on Oper. Res.*, pages 331–334, Herceg–Novi, 2003.

[48] T. Davidović and N. Mladenović. Genetic algorithms for multiprocessor scheduling problem with communication delays. In *Proc. 10. Congr. Yugosalv Matematicians*, pages 321–324, Beograd, 2001.

[49] M. den Besten and T. Stützle. Neighborhood revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. In *Proc. MIC'2001,* 4^th *Metaheuristic International Conference*, pages 545–549, Porto, Portugal, 2001.

[50] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Eng.*, 15(12):1497–1506, December 1989.

[51] G. Djordjević and M. Tošić. A compile-time scheduling heuristic for multiprocessor architectures. *The Computer Journal*, 39(8):663–674, 1996.

[52] K. A. Dowsland. Genetic algorithms – a tool for or? *Journal of the Operational Research Society*, 47:550–561, 1996.

[53] M. Drozdowski. Scheduling multiprocessor tasks - an overview. *Europ. J. Oper. Res.*, 94:215–230, 1996.

[54] V. Filipović, J. Kratica, D. Tošić, and I. Ljubić. Fine grained tournament selection for the simple plant location problem. In *Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications - WSC5*, pages 152–158, 2000.

[55] G. S. Fishman. *Monte Carlo concepts, alforithms and applications.* Springer series in Operations Research, New York, 1996.

[56] K. Fleszar and K. S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighborhood search. *Europ. J. Oper. Res.*, 155(2):402–413, 2004.

[57] Coffman, Jr. E. G. and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica*, 1:200–213, 1972.

[58] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *J. Heur.*, 8(3):375–388, May 2002.

[59] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness.* W. H. Freeman and Company, 1979.

[60] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers Ops. Res.*, 5:533–549, 1986.

[61] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[62] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publ. Comp., Inc., 1989.

[63] R. E. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 4:287–326, 1979.

[64] R. L. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Applied Math.*, 17:416–429, 1969.

[65] W. Gropp and E. Lusk. *Users Guide for* `mpich` *a Portable Implementation of MPI.* University of Chicago, Argonne National Laboratory, 1996.

[66] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message–Passing Interface.* The MIT Press, 1994.

[67] N. G. Hall and M. E. Posner. Sensitivity analysis for scheduling problems. *(revised for Operations Research)*, 2002.

[68] P. Hansen. The steepest ascent mildest descent heuristics for combinatorial programming. In *Congress on Numerical Methods ni Combinatorial Optimization*, Capri, Italy, 1986.

[69] P. Hansen and N. Mladenović. First improvement may be better than best improvement: an empirical study. Technical report, GERAD report G-99-54, (Discrete Applied Math. - to appear), 1999.

[70] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In Voss, S. *et al.*, editor, *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Dordrecht, 1999.

[71] P. Hansen and N. Mladenović. Fundamentals of variable neighborhood search. In *Proc. 10. Congr. Yugoslav Mathematicians*, pages 57–72, Beograd, 2001.

[72] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *Europ. J. Oper. Res.*, 130:449–467, 2001.

[73] P. Hansen and N. Mladenović. Developments of the variable neighborhood search. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, 2002.

[74] P. Hansen and N. Mladenović. Variable neighbourhood search. In F. Glover and G. Kochenagen, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer Academic Publishers, Dordrecht, 2003.

[75] P. Hansen, N. Mladenović, and D. Urošević. Variable neighbourhood search and local branching. *Computers Ops. Res. (to appear)*, 2005.

[76] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[77] J. A. Hoogeveen, J. K. Lenstra, and S. L. van de Velde. Sequencing and scheduling. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 181–197. John Wiley & Sons, 1997.

[78] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel and Distributed Systems*, 5(2):113–120, Feb. 1994.

[79] T. C. Hu. Parallel sequencing and assembly line problems. *Oper. Res*, 9(6):841–848, Nov. 1961.

[80] R. Hübscher and F. Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers Ops. Res.*, 21(8):877–884, 1994.

[81] C. A. J. Hurkens and T. Vredeveld. Local search for multiprocessor scheduling: How many moves does it take to a local optimum? *Operations Research Letters*, 31(2):137–141, March 2003.

[82] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, 1977.

[83] N. Kirćanski, T. Davidović, and M. Vukobratović. A contribution to parallelization of symbolic robot models. *Robotica*, 13:411–421, 1995.

[84] J. Kratica. Improving performances of the genetic algorithm by caching. *Computers and Artificial Intelligence*, 18(3):271–283, 1999.

[85] V. Krishnamoorthy and K. Efe. Task scheduling with and without communication delays: A unified approach. *European Journal of Operational Research*, 89:366–379, 1996.

[86] Y.-K. Kwok and I. Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proc. 7th IEEE Symposium of Parallel and Distributed Processing (SPDP'95)*, pages 36–43, Dallas, Texas, USA, Oct. 1995.

[87] Y.-K. Kwok and I. Ahmad. Dynamic critical path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[88] Y.-K. Kwok and I. Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *J. Parallel and Distributed Computing*, 47:58–77, 1997.

[89] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel and Distributed Computing*, 59(3):381–422, Dec. 1999.

[90] K. Li. Analysis of the list scheduling algorithm for precedence constrained parallel tasks. *Journal of Combinatorial Optimization*, 3:73–88, 1999.

[91] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College, London, UK, 2004.

[92] N. Maculan, C. C. Ribeiro, S.C.S. Porto, and C. C. de Souza. A new formulation for scheduling unrelated processors under precedence constraints. *RAIRO Recherche Operationelle*, 33:87–90, 1999.

[93] B. A. Malloy, E. L. Lloyd, and M. L. Soffa. Scheduling DAG's for asynchronous multiprocessor execution. *IEEE Trans. Parallel and Distributed Systems*, 5(5):498–508, May 1994.

[94] A. S. Manne. On the job–shop scheduling problem. *Oper. Res.*, 8(2):219–223, 1960.

[95] S. Manoharan and P. Thanisch. Assigning dependency graphs onto processor networks. *Parallel Computing*, 17:63–73, 1991.

[96] C. McCreary and H. Gill. Automatic determination of grain size for efficient parallel processing. *Communications of the ACM*, 32(9):1073–1078, Sept. 1989.

[97] D. A. Menascé and S. C. S. Porto. Processor assignment in heterogeneous parallel architectures. In *Proc. of the IEEE Int. Parallel Processing Symposium*, pages 186–191, Beverly Hills, 1992.

[98] D. A. Menascé, S.C.S. Porto, and S. K. Tripathi. Static heuristic processor assignment in heterogeneous multiprocessors. *Int. J. High Speed Computing*, 6(1):115–137, 1994.

[99] P. Merz. *Memetic Algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies*. PhD thesis, Dept. Computer Science and Electrical Engineering (FB 12), Research Group PSy, University of Siegen, Germany, 2000.

[100] S. Mitrović-Minić. *The dynamic pickup and delivery problem with time windows*. PhD thesis, School of Computing Science, Simon Fraser University, Vancouver, Canada, 2002.

[101] S. Mitrović-Minić, G. Laporte, and R. Krishnamurti. The double–horizon heuristic for the dynamic pickup and delivery problem. In *Optimization Days*, page 77, Montreal, 2002.

[102] N. Mladenović. A variable neighborhood algorithm – a new metaheuristic for combinatorial optimization applications. In *Optimization Days*, page 112, Montreal, 1995.

[103] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. & OR*, 24(11):1097–1100, 1997.

[104] J. A. Moreno-Pérez, P. Hansen, and N. Mladenović. Parallel variable neighborhood search. In E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley, 2005.

[105] P. Moscato. *On Evolution, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Caltech Concurrent Computation Program, C3P Report 826, 1989.

[106] P. Moscato. An introduction to population approaches for optimization and hierarchical objective function: A discussion on the role of tabu search. *Annals of Operations Research*, 41:85–122, 1993.

[107] H. Mühlenbein. Genetic algorithms. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial optimization*, pages 137–171. John Wiley & Sons Ltd., 1997.

[108] M. G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):263–302, Sept. 1993.

[109] I. Or. *Traveling salesman - type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, 1976.

[110] D. Peng and K. G. Shin. Modeling of concurrent task execution in a distributed system for real-time control. *IEEE Trans. Computers*, C-36(4):500–516, Apr. 1987.

[111] T. Petrović. Jedan prilaz paralelizaciji izračunavanja matematičkih modela robota. Master's thesis, Matematički fakultet Univerziteta u Beogradu, Nov. 1992.

[112] M. Pirlot. General local search methods. *Europ. J. Oper. Res.*, 92:493–511, 1996.

[113] K. Politopoulos, G. F. Georgakopoulos, and P. Tsanakas. Precedence constrained scheduling: A case in *p*. *The Computer Journal*, 44(3):163–173, 2001.

[114] S.C. Porto and C.C. Ribeiro. Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. *J. Heur.*, 1(2):207–225, 1995.

[115] S.C. Porto and C.C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *Int. J. High-Speed Computing*, 7:45–71, 1995.

[116] S.C.S. Porto, Kitajima J.P.F.W., and C.C. Ribeiro. Performance evaluation of a parallel tabu search task scheduling algorithm. *Parallel Computing*, 26(1):73–90, 2000.

[117] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling. Technical report, TUB:1994–408, Technical University of Berlin, 1994.

[118] M. J. Quinn. *Designing efficient algorithms for parallel computers.* McGraw-Hill, 1987.

[119] K. Ramamritham, J. A. Stanković, and P-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. Parallel and Distributed Systems*, 1(2):184–194, Apr. 1990.

[120] K. Ramamritham, J. A. Stanković, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Computers*, 38(8):1110–1122, Aug. 1989.

[121] I. Rodriguez, M. Moreno-Vega, and J. Moreno-Perez. Heuristics for roting–median problems. Technical report, SMG report, Université Libre de Bruxelles, Belgium, 1999.

[122] S. K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

[123] A. K. Sarje and G. Sagar. Heuristic model for task allocation in distributed computer systems. *IEE Proceedings-E*, 138(5):313–318, Sept. 1991.

[124] G. C. Sih. *Multiprocessor Scheduling to Account for Interprocessor Communication.* PhD thesis, University of California, Berkeley, 1991.

[125] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[126] J. A. Stanković, K. Ramamritham, and S. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Trans. Computers*, C-34(12):1130–1143, Dec. 1985.

[127] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *J. Heur.*, 8:541–564, 2002.

[128] A. Thesen. Design and evaluation of a tabu search algorithm for multiprocessor scheduling. *J. Heur.*, 4(2):141–160, 1998.

[129] T. Tobita and H. Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[130] M. Toulouse, T. G. Crainic, and M Gendreau. Communication issues in designing cooperative multi thread parallel searches. In I. H. Osman and J. P. Kelly, editors, *Meta-heuristics 98: Theory & Applications*, pages 501–522. Kluwer Academic Publishers, 1996.

[131] D. Tošić, N. Mladenović, J. Kratica, and V. Filipović. *Genetski Algoritmi*. Matematički Institut SANU, Beograd, (u štampi).

[132] W. T. Trotter. *Combinatorics and partially ordered sets*. The Johns Hopkins University Press, Baltimore, 1992.

[133] J. D. Ullman. NP-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393, 1975.

[134] D. Urošević. *Rešavanje nekih problema teorije grafova metodom promenljivih okolina*. PhD thesis, Matematički fakultet Univerziteta u Beogradu, 2004.

[135] T. A. Varvarigou, V. P. Roychowdhury, T. Kailath, and E. Lawler. Scheduling in and out forests in the presence of communication delays. *IEEE Trans. Parallel and Distributed Systems*, 7(10):1065–1074, Oct. 1996.

[136] B. Veltman, B. J. Lageweg, and J. K. Lenstra. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16:173–182, 1990.

[137] M. G. A. Verhoeven and E. H. L. Aarts. Parallel local search. *J. Heur.*, 1:43–65, 1995.

[138] H. M. Wagner. An integer linear-programming model for machine scheduling. *Nav. Res. Log. Quart.*, 6(2):131–140, 1959.

[139] C-M. Wang and S-D. Wang. Structured partitioning of concurrent programs for execution on multiprocessors.

[140] H. Wu, J. Jaffar, and R. Yap. A fast algorithm for scheduling instructions with deadline constraints on risc processors. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'00)*, page 281, Philadelphia, Pennsylvania, USA, Oct. 15-19. 2000.

[141] W. Zhao, K. Ramamritham, and J. A. Stanković. Preemptive scheduling under time and resource constraints. *IEEE Trans. Computers*, C-36(8):949–960, Aug. 1987.

A Generator test primera grafova zadataka

Uvidom u odgovarajuće baze dostupne preko Interneta, utvrđeno je da ne postoje adekvatni test primeri za analiziranje i upoređivanje predloženih metoda za rešavanje specifične varijante problema raspoređivanja (MSPCD) koja se u ovome radu razmatra. Test primeri postoje za neke druge, uglavnom jednostavnije varijante problema. Pokazalo se kao neophodno razvijanje programskog paketa za generisanje grafova zadataka sa datim osobinama. Tri generatora objedinjena su u programski paket `task_gen`. To su `tgen_rnd`, generator grafova kod kojih su zadata ograničenja na visinu i širinu grafa, dužinu izvršavanja zadataka i dužinu komunikacija među zadacima; zatim `tgen_den`, koji generiše grafove sa zadatom gustinom povezanosti i `tgen_opt` za generisanje grafova sa poznatom optimalnom dužinom raspodele na zadatoj višeprocesorskoj arhitekturi. Svi tipovi test primera grafova zadata generišu se na slučajan način uz poštovanje zadatih uslova.

Programski paket `task_gen` implementiran je, kao i paket za raspoređivanje, na programskom jeziku C pod Linux operativnim sistemom. Zavisno od ulaznog indikatora, on poziva jedan od programa za generisanje odgovarajućih grafova zadataka, koji se zatim upisuju u izlazne datoteke. Te datoteke sadrže generisane grafove zapisane u formatu prepoznatljivom od strane programa za raspoređivanje. To je ujedno standardni format u kome se zapisuju primeri u odgovarajućim bazama.

Za generisanje slučajnih brojeva korišćena je standardna funkcija programskog jezika C, a tako generisani slučajni brojevi su zatim prevođeni (normalizacijom) u odgovarajući opseg i odgovarajući tip. Svaki od generatora može odjednom da generiše na slučajan način više različitih grafova sa istim osobinama što je potrebno pri statističkim analizama. Važno je još napomenuti da programski paket `task_gen` može da ima širu primenu u generisanju test primera za mnoge druge probleme, naročito u teoriji grafova.

A1. Program `tgen_rnd`

Slučajno generisani grafovi sa zadatom visinom i širinom pogodni su pri ispitivanju efikasnosti raznih konstruktivnih heuristika za raspoređivanje pojedinih klasa grafova zadataka. Ovi parametri (visina i širina) na neki način definišu stepen paralelizma u polaznom grafu i određuju pogodnost ciljne višeprocesorske arhitekture za izvršavanje datog grafa.

Ulazni podaci ovog generatora su broj grafova koji se generiše, broj zadataka u svakom od njih, visina (broj nivoa) svakog grafa (pri čemu je dozvoljena povezanost i zadataka sa nesusednih nivoa, dakle ne generišu se samo višenivoovski grafovi), maksimalna dužina izvršavanja zadataka, maksimalni broj (neposrednih) sledbenika svakog zadatka i maksimalna dužina komunikacije svakog zadatka sa svojim neposrednim sledbenicima. Ulazni podaci zadaju se u ulaznoj datoteci `tgenr_in.dat`. Evo primera ulazne datoteke.

```
2    broj_grafova_koji_se_generise~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
20   broj_zadataka_u_grafu_koji_se_generise~~~~~~~~~~~~~~~~~~~~~
7    maksimalni_broj_nivoa_u_grafu_zadataka~~~~~~~~~~~~~~~~~~~~~
50   maksimalna_duzina_izvrsavanja_nekog_zadatka~~~~~~~~~~~~~~~~~
5    maksimalni_broj_sledbenika_svakog_zadatka~~~~~~~~~~~~~~~~~~~
40   maksimalna_duzina_komunikacije_nekog_zadatka_sa_sledbenicima
```

Prilikom generisanja svakog od grafova određuje se maksimalan broj zadataka na svakom od nivoa (što ustvari predstavlja širinu grafa) kao količnik broja zadataka i broja nivoa. Zatim se tačan broj zadataka na svakom nivou određuje slučajno (imajući na umu izračunato ograničenje i ukupan broj zadataka u grafu). Svakome zadatku, dužina izvršavanja određuje se slučajno (uniformno između 1 i maksimalne dozvoljene). Takođe na slučajan način, odredi se koliko će svaki zadatak imati sledbenika (vodeći računa o zadatom ograničenju) i koji zadaci će mu biti sledbenici. Prilikom generisanja grafova uveden je parcijalni poredak u numeraciji čvorova, tako da važi $t_i \in Succ(t_j) \Rightarrow i > j$. Dakle, sledbenike nekog zadatka biramo slučajno među čvorovima sa "viših" nivoa, tj. među čvorovima sa indeksima između rednog broja prvog čvora sa narednog nivoa i ukupnog broja čvorova u grafu. Naravno, čvorovi sa poslednjeg nivoa neće imati sledbenike. Svakom od sledbenika pridružuje se slučajno, ali u zadatim granicama, komunikacija, tj. količina podataka koju je neophodno proslediti zadatku sledbeniku nakon završetka izvršavanja datog zadatka da bi se omogućilo njegovo korektno

izvršavanje.[1] Na osnovu ovako generisanih veličina izvrši se rekonstrukcija kompletnog grafa, dakle svakom zadatku se pridruže njegovi neposredni prethodnici i ovako kompletiran graf upiše se u izlaznu datoteku. Ime izlazne datoteke generiše se tako da sadrži tip generisanog grafa (*rnd*, *den* ili *opt*) i redni broj odgovarajuég grafa.

Razlike među ovako generisanim grafovima obezbeđuju slučajni brojevi koji se generišu za broj zadataka na svakom nivou, broj sledbenika svakog zadatka, dužine izvršavanja zadataka i količinu komunikacije. Jedna od izlaznih datoteka, tj. jedan od dva grafa zadataka generisana na osnovu navedene ulazne datoteke izgleda ovako:

```
rb.  ime    duzina  prethodnici_0    sledbenici_0   komunikacije_0
  1   T1    17      0                6  15  13  5  0       6  29  26  37  0
  2   T2    31      0                5  0                  6  0
  3   T3    41      0                5  0                  6  0
  4   T4     6      0                15  14  6  13  5  0   31  12  39  20  21  0
  5   T5    27      4  3  2  1  0     10  11  19  8  0      22  38  3  37  0
  6   T6     5      4  1  0           9  0                  36  0
  7   T7    18      0                11  0                 19  0
  8   T8     4      5  0             20  14  0             11  35  0
  9   T9    27      6  0             16  14  0             22  27  0
 10  T10     2      5  0             13  20  14  0         26  30  12  0
 11  T11    18      7  5  0          16  19  17  15  0     28  15  36  10  0
 12  T12    48      0                17  20  18  0         33  16  37  0
 13  T13    35     10  4  1  0       15  20  19  16  17  0 12  25  18  26  36  0
 14  T14    40     10  9  8  4  0    16  17  0             12  8  0
 15  T15    28     13  11  4  1  0   20  0                 5  0
 16  T16     7     14  13  11  9  0  20  0                 6  0
 17  T17    40     14  13  12  11  0 20  0                 3  0
 18  T18    48     12  0             20  0                 8  0
 19  T19    13     13  11  5  0      20  0                 21  0
 20  T20     4     19  18  17  16  15  13  12  10  8  0  0     0
```

Datoteka je upisana u sledećem formatu: prvi red sadrži komentare o podacima koji se nalaze u datoteci; svaki od narednih n redova sadrži podatke o jednom zadatku i to redni broj, ime zadatka, dužinu izvršavanja i tri liste koje se završavaju nulama, a sadrže spisak prethodnika, spisak sledbenika i količinu komunikacije sa svakim od sledbenika. Prilikom učitavanja, broj zadataka se izračunava i određen je brojem redova sa podacima koje datoteka

[1] Komunikacija se uzima u obzir samo u slučaju da se zadatak i njegov sledbenik izvršavaju na različitim procesorima i pri tome se množi rastojanjem među odgovarajućim procesorima koje se uzima iz matrice rastojanja.

sadrži. Radi uniformisanja sa formatom u kome se zapisuju zvanični test primeri, postoji i drugi format zapisa (u koji se navedeni lako prevodi). Taj format u prvom redu sadrži broj zadataka n, a zatim u narednih n redova redni broj zadatka definisanog u tom redu, za kojim sledi njegova dužina izvršavanja, broj sledbenika n_{succ} i zatim do kraja reda, n_{succ} parova brojeva koji predstavljaju redni broj sledbenika i odgovarajuću količinu komunikacije.

A2. Program `tgen_den`

Grafovi zadataka određene gustine veza među čvorovima omogućavaju nam da generišemo lakše i teže probleme za testiranje metaheuristika. Naime, prostor pretrage kod grafova manje gustine je veći, obzirom na to da postoji više međusobno nezavisnih zadataka koji mogu biti raspoređeni na proizvoljan način. Kod gustih grafova dopustivih rešenja je manje, te ih je moguće brže pretražiti.

Ulazni parametri ovog programa su broj grafova koji se generišu, broj zadataka u svakom od njih, maksimalna dužina izvršavanja svakog od zadataka, maksimalna dužina komunikacije među zadacima i gustina veza među čvorovima. Ulazni podaci zadaju se u ulaznoj datoteci `tgend_in.dat`.

Nakon učitavanja ulaznih podataka, program generiše odgovarajući broj grafova sa istim brojem čvorova i istom gustinom veza među njima na sledeći način. Svakom čvoru pridružuje se slučajno generisana dužina izvršavanja (ograničena odozgo maksimalno dozvoljenom). Imajući na umu parcijalni poredak u numeraciji čvorova, sledbenici nekog čvora određuju se tako što se za svaki čvor sa većim rednim brojem, sa verovatnoćom jednakom zadatoj gustini veza u grafu, generiše luk između ta dva čvora i njegova težina odredi slučajno (ograničena maksimalno dozvoljenom dužinom komunikacija). Nakon kompletiranja informacije o prethodnicima svakog čvora, generisani grafovi se upisuju u izlazne datoteke u istom formatu kao i u prethodnom slučaju.

Na primer, na osnovu ulazne datoteke

```
3  broj_grafova_koji_se_generise~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
10 broj_zadataka_u_grafu_koji_se_generise~~~~~~~~~~~~~~~~~~~~~~~
25 maksimalna_duzina_izvrsavanja_nekog_zadatka~~~~~~~~~~~~~~~~~~
25 maksimalna_duzina_komunikacije_nekog_zadatka_sa_sledbenicima~~
50.00 gustina_povezanosti_grafa~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

generisana su tri grafa prikazana na slici A1.

```
[tanjad@abit task_gen]> cat DGRA_1.td

rb.  ime   duzina    prethodnici_0   sledbenici_0   komunikacije_0
 1   T1    15      0  10  9  8  7  6  4  3  2  0  14  17  17  16  11  13  1  1  0
 2   T2    4       1  0  10  9  8  7  0  12  5  18  25  0
 3   T3    20      1  0  10  9  0  21  10  0
 4   T4    18      1  0  10  6  0  24  9  0
 5   T5    10      0  10  9  8  6  0  8  5  22  22  0
 6   T6    16      5  4  1  0  10  9  8  0  1  25  8  0
 7   T7    11      2  1  0  10  9  8  0  10  3  8  0
 8   T8    12      7  6  5  2  1  0  9  0  5  0
 9   T9    14      8  7  6  5  3  2  1  0  10  0  9  0
 10  T10   20      9  7  6  5  4  3  2  1  0  0  0

[tanjad@abit task_gen]> cat DGRA_2.td

rb.  ime   duzina    prethodnici_0   sledbenici_0   komunikacije_0
 1   T1    7       0  9  6  3  2  0  18  7  12  9  0
 2   T2    18      1  0  5  3  0  10  5  0
 3   T3    17      2  1  0  7  6  5  4  0  21  16  18  18  0
 4   T4    17      3  0  10  9  0  21  6  0
 5   T5    10      3  2  0  10  6  0  19  18  0
 6   T6    3       5  3  1  0  8  0  10  0
 7   T7    2       3  0  10  9  8  0  17  24  10  0
 8   T8    14      7  6  0  10  9  0  14  9  0
 9   T9    23      8  7  4  1  0  0  0
 10  T10   20      8  7  5  4  0  0  0

[tanjad@abit task_gen]> cat DGRA_3.td

rb.  ime   duzina    prethodnici_0   sledbenici_0   komunikacije_0
 1   T1    25      0  10  9  8  7  6  2  0  8  6  7  14  13  14  0
 2   T2    8       1  0  9  6  5  4  3  0  10  10  11  17  18  0
 3   T3    1       2  0  8  7  6  4  0  11  11  23  3  0
 4   T4    17      3  2  0  10  8  7  6  0  1  25  9  25  0
 5   T5    3       2  0  9  7  6  0  20  5  11  0
 6   T6    20      5  4  3  2  1  0  8  0  1  0
 7   T7    12      5  4  3  1  0  9  8  0  1  24  0
 8   T8    11      7  6  4  3  1  0  9  0  16  0
 9   T9    10      8  7  5  2  1  0  10  0  17  0
 10  T10   3       9  4  1  0  0  0
```

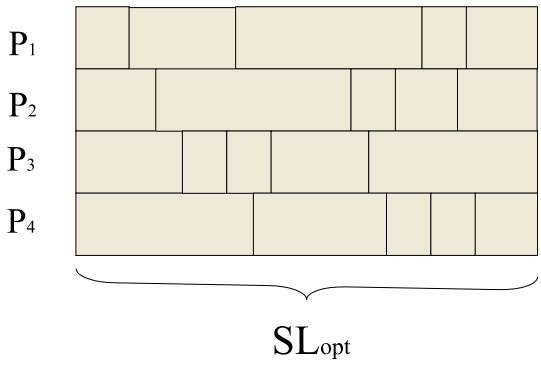
Sl. A1: Primeri grafova sa $n = 10$ zadataka i gustinom povezanosti $\rho = 50\%$

A3. Program `tgen_opt`

Da bi se ispitao kvalitet dobijene heurističke raspodele, neophodno je da se zna minimalna moguća, tj. dužina optimalne raspodele. Stoga je razvijen program koji generiše grafove sa poznatom optimalnom dužinom raspodele na zadatoj višeprocesorskoj arhitekturi.

Ulazni parametri ovog generatora su broj gafova koji se generiše, broj čvorova u svakom od grafova, dužina optimalne raspodele, broj procesora u višeprocesorskom sistemu na koji se vrši raspoređivanje i matrica veza među procesorima.

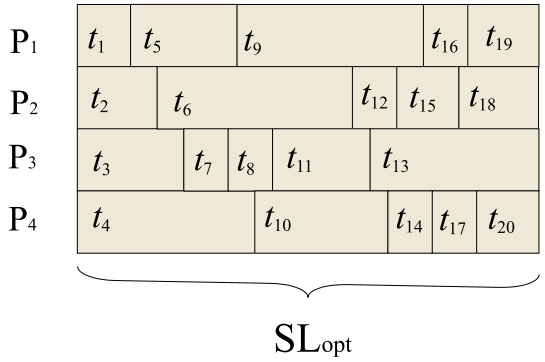
Zadati broj grafova generiše se tako da svaki od njih ima istu optimalnu dužinu raspodele, istu dužinu izvršavanja svakog od zadataka, a različite gustine veza među zadacima. U stvari, u ovom slučaju generiše se najpre sama optimalna raspodela, a zatim zadat broj grafova koji se mogu na taj način optimalno rasporediti. Uvode se sledeće pretpostavke:
1) na svakom od procesora izvršavaće se približno jednak broj zadataka (sa dozvoljenim odstupanjem 10%);
2) svi procesori rade sve vreme izvršavanja paralelizovanog grafa zadataka, tj. ni na jednom procesoru nema intervala čekanja i svi završavaju u isto vreme (u trenuku zadatom dužinom optimalne raspodele SL_{opt}).
Raspodela koja zadovoljava navedene uslove izgleda kao na slici A2 (ovde je uzeto da se višeprocesorski sistem sastoji od 4 identična procesora).



Sl. A2: Primer optimalne raspodele za $p = 4$ procesora

Proces generisanja odgovarajućih grafova odvija se na sledeći način. Na slučajan način odredi se koliko zadataka se izvršava na svakom od procesora

(uzimajući u obzir dozvoljeno odstupanje od 10% u odnosu na jednak broj zadataka na svakom procesoru). Dužine izvršavanja tih zadataka određuju se uniformno sa očekivanjem jednakim količniku zadate dužine raspodele i broja zadataka na odgovarajućem procesoru i odstupanjem od maksimum 10%. Nakon toga vrši se numeracija čvorova izlaznih grafova, imajući u vidu vreme početka izvršavanja svakogs od njih na odgovarajućem procesoru. Preciznije, na prvom procesoru u prvom trenutku počinje izvršavanje zadatka broj 1; na drugom procesoru u prvom trenutku počinje da se izvršava zadatak broj 2 itd. dok se ne obiđu svi procesori. Neka ih ima 4 (kao na slici A2). Dakle, u prvom trenutku na četvrtom procesoru počeo je izvršavanje zadatak broj 4. Broj 5 dobiće zadatak koji sledi iza najkraćeg od ova prva četiri, bez obzira na kom se procesoru nalazi. Šesti će biti zadatak koji takođe bez obzira na redni broj procesora na kome se izvršava ima sledeće najmanje vreme starta, itd. Na taj način za primer sa slike A2 dobićemo numeraciju prikazanu na slici A3.



Sl. A3: Numeracija zadataka kod grafova sa poznatom optimalnom raspodelom

Sledeći korak u generisanju je dodavanje lukova, tj. definisanje veza među čvorovima. Najpre treba odrediti maksimalnu moguću gustinu veza među ovako rapoređenim zadacima. Jasno je da se ne može uvesti luk između zadataka t_i i t_j, $i < j$ za koje važi da je počelo izvršavanje zadatka t_j pre nego što se završilo izvršavanje zadatka t_i. Tako u grafu generisanom na osnovu slike A3 ne može postojati veza između zadataka t_6 i t_{11}.

Veze među zadacima sa istog procesora su dozvoljene, jer je prethodni uslov sigurno zadovoljen. Nakon što se odredi maksimalna dopustiva gustina

grafa, pristupa se generisanju zadatog broja grafova od kojih će svaki imati različitu gustinu. Gustina i-tog grafa dobija se tako što se maksimalna gustina podeli sa zadatim brojem grafova i to pomnoži sa i. Zatim se izračuna broj lukova u takvom grafu na osnovu činjenice da potpuno povezani graf (gustine 100%) ima $n(n-1)/2$ lukova pri čemu je n broj čvorova.

Za svaki od lukova, slučajno se biraju zadaci koje će on povezivati. Ako su zadaci na istom procesoru, dužina komunikacije se određuje slučajno. Ako to nije slučaj, dužina komunikacije se mora izračunati na osnovu vremena početka izvršavanja svakog od zadataka i rastojanja između odgovarajućih procesora.[2] Ovako generisani grafovi različitih gustina upisuju se u izlazne datoteke.

Prilikom generisanja prethodna dva tipa grafova nije vođeno računa o ciljnoj arhitekturi na koju će se oni raspoređivati, tako da je bilo moguće eksperimentisati i sa efikasnošću heuristike zavisno od strukture ne samo polaznog grafa već i višeprocesorskog sistema na koji se vrši raspodela. U ovom slučaju zadata optimalna dužina raspodele odnosi se samo na višeprocesorsku arhitekturu na osnovu koje je vršeno generisanje polaznog grafa. Ukoliko je polazna višeprocesorska arhitektura predstavljala nepotpunu mrežu procesora, tj. nisu postojali direktni komunikacioni kanali između svaka dva procesora, primeri koji su na ovaj način generisani, mogu se koristiti i prilikom raspoređivanja na potpuno povezani višeprocesorski sistem sa istim brojem procesora. Obratno naravno, ne važi, tj. primeri generisani za potpuno povezani višeprocesorski sistem, neće imati istu optimalnu raspodelu ako se raspoređuju na nepotpunu mrežu procesora. Ako se zanemaruje vreme komunikacije, ovi primeri mogu poslužiti u svim slučajevima veza među procesorima (bitan je samo broj procesora).

Grafovi generisani pomoću opisanog paketa i korišćeni u eksperimentima izvršenim u toku izrade ovoga rada predloženi su za zvanične test primere (benchmarks) za testiranje (meta)heurističkih metoda koje se primenjuju na statičko raspoređivanje zadataka u prisustvu komunikacionih kašnjenja (MSPCD), opisani su u radu [42] i dostupni su preko Interneta na adresi `http://www.mi.sanu.ac.yu/~tanjad`.

[2]Da bi izvršavanje bilo korektno, komunikacija se mora završiti pre nego što je predviđeno da sledbenik započne svoje izvršavanje.

B Tabele sa rezultatima paralelnih verzija VNS metode

Prilikom izvršavanja paralelnih varijanti VNS metode, teorijske pretpostavke nisu uvek bile ispravne, pa se pojavila potreba za dodatnim eksperimentalnim utvrđivanjem nekih parametara izvršavanja. Kao proizvod toga, generisano je mnogo tabela sa rezultatima, na osnovu kojih su izvođeni određeni zaključci i vršene ispravke i modifikacije koje su dovodile do poboljšavanja rezultata dobijenih paralelnim izvršavanjem. Ove tabele jesu od značaja, ali opterećuju osnovni tekst i otežavaju njegovo praćenje. Stoga su navedene u ovom prilogu, a u glavi 5.7. poziva se na njih gde god je to potrebno.

B1. Tabele vezane za paralelni LS

U ovom odeljku navedeni su neki rezultati dobijeni primenom paralelnog pretraživanja okoline (PNE). Prvo su date dve tabele za PNE pretraživanje samo u jednom smeru (FOR-BACK) počev od CPES rešenja, zatim tri tabele koje se odnose na pretraživanje počev od slučajnog rešenja. Potreba za balansiranjem opterećenja procesora vidljiva je još iz tabele 5.1, ali su ostale potvrdile taj zaključak i pokazale da ponašanje pretraživanja na smanjenim okolinama nije isto kao kad se pretražuje cela okolina, te i balansiranje treba da zavisi od toga.

U tabelama B6 do B8 dati su rezultati pretraživanja primenom PNE sa popravljenim opterećenjem procesora počev od slučajnog rešenja. Oni su od značaja za primene u okviru paralelnih MLS i VNS metoda, jer su tamo polazna rešenja za LS proceduru uglavnom slučajna.

Tabela B1: Rezultati raspoređivanja primenom PNE pretraživanja unapred

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	4.0	389.92	389.92	1.00
	2781.4	(389.92)	389.96	
1	4.0	196.90 204.85	401.76	1.90
	2781.4	(61.97) (204.85)	204.85	
2	4.0	150.48 153.34 157.12	460.95	2.48
	2781.4	(26.61) (84.10) (157.12)	157.11	
3	4.0	120.98 121.86 123.61 125.35	491.80	3.11
	2781.4	(14.59) (46.98) (80.92) (125.35)	125.39	
4	4.0	102.83 101.70 103.22 103.75 105.08	516.58	3.71
	2781.4	(9.43) (29.42) (50.44) (73.64) (105.08)	105.05	
5	4.0	86.38 86.87 86.82 87.74 88.59 89.10	525.50	4.37
	2781.4	(6.36) (20.08) (34.71) (49.03) (69.45) (89.08)	89.10	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	7.3	177.41	177.41	1.00
	2772.5	(177.41)	177.63	
1	6.0	146.47 148.70	295.17	1.19
	2785.2	(99.60) (103.61)	150.61	
2	5.4	137.94 140.43 135.13	413.51	1.26
	2791.0	(70.18) (118.62) (35.56)	141.72	
3	4.9	120.18 121.51 119.09 119.24	480.02	1.46
	2785.8	(52.32) (107.80) (61.67) (18.86)	122.10	
4	5.0	111.44 112.90 111.64 111.67 110.95	558.60	1.57
	2790.2	(47.91) (100.56) (66.91) (38.74) (12.58)	113.19	
5	4.8	101.82 104.28 103.43 102.57 101.52 101.58	615.20	1.70
	2790.9	(39.75) (95.11) (66.39) (45.33) (24.96) (8.25)	104.46	

Tabela B2: Rezultati raspoređivanja primenom PNE pretraživanja unatrag

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	3.8	315.71	315.71	1.00
	2737.9	(315.71)	315.89	
1	3.8	271.31 283.47	554.78	1.11
	2737.9	(79.43) (283.47)	283.47	
2	3.8	208.48 208.59 216.14	633.21	1.46
	2745.4	(32.77) (114.99) (216.14)	216.25	
3	3.8	163.35 164.78 167.90 170.33	666.36	1.85
	2737.9	(18.10) (61.95) (119.04) (170.32)	170.33	
4	3.8	137.26 136.07 138.65 139.04 140.18	691.19	2.25
	2737.9	(11.58) (37.52) (68.93) (109.87) (139.95)	140.19	
5	3.8	114.94 116.10 117.09 117.56 118.14 118.47	702.31	2.66
	2737.9	(7.67) (24.79) (46.22) (69.17) (102.42) (116.83)	118.58	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	8.1	195.44	195.44	1.00
	2771.6	(195.44)	200.23	
1	6.9	179.41 178.84	358.25	1.09
	2774.7	(118.41) (112.64)	183.85	
2	6.4	189.58 194.49 189.43	573.50	1.01
	2777.7	(89.98) (162.12) (42.14)	196.54	
3	6.8	210.54 217.97 212.04 208.55	849.10	0.90
	2778.3	(76.48) (201.52) (105.16) (29.76)	222.95	
4	6.7	185.95 192.24 188.58 185.69 184.37	936.82	1.02
	2772.4	(65.06) (181.10) (110.87) (59.35) (18.13)	196.04	
5	6.2	174.08 179.12 176.82 175.96 174.60 173.92	1054.50	1.09
	2779.4	(52.00) (168.73) (112.50) (74.43) (37.09) (12.35)	183.01	

Tabela B3: Rezultati raspoređivanja primenom PNE počev od slučajnog rešenja i pretraživanjem u oba smera

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	3.8	616.13	616.13	1.00
	2780.5	(616.13)	632.38	
1	3.8	449.36 471.78	921.14	1.31
	2780.5	(138.12) (471.75)	481.66	
2	3.8	343.48 354.24 362.51	1060.23	1.70
	2780.5	(58.10) (187.85) (362.50)	371.89	
3	3.8	272.47 277.99 280.27 286.44	1117.18	2.15
	2780.5	(30.22) (102.56) (175.47) (286.40)	293.65	
4	3.8	234.66 236.91 238.05 243.14 245.41	1198.16	2.51
	2780.5	(19.56) (66.82) (112.53) (167.51) (245.17)	252.74	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	8.5	396.33	396.33	1.00
	2786.9	(396.33)	405.46	
1	7.4	280.23 283.34	563.57	1.40
	2792.3	(162.70) (201.32)	299.86	
2	7.4	304.78 313.56 303.24	921.59	1.26
	2787.9	(137.66) (261.10) (88.35)	326.26	
3	8.4	414.00 423.05 413.95 404.97	1655.96	0.94
	2779.5	(193.43) (364.75) (228.78) (64.86)	436.88	
4	6.3	248.22 256.22 253.88 248.75 247.54	1254.60	1.55
	2793.5	(94.77) (219.33) (168.51) (95.27) (28.28)	267.51	

Tabela B4: Rezultati raspoređivanja primenom PNE pretraživanja unapred počev od slučajnog rešenja

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	4.4	476.29	476.29	1.00
	2780.6	(476.29)	476.29	
1	4.4	340.89 357.71	698.60	1.33
	2780.6	(92.26) (357.71)	357.71	
2	4.4	281.24 292.59 296.43	870.26	1.61
	2780.6	(39.61) (145.26) (296.43)	296.57	
3	4.4	199.44 201.77 204.51 207.73	813.46	2.29
	2780.6	(17.73) (64.27) (120.41) (207.73)	207.72	
4	4.4	182.23 181.54 182.77 184.29 186.61	917.43	2.55
	2780.6	(11.97) (44.32) (78.84) (120.82) (186.61)	186.60	
5	4.4	158.31 157.19 158.85 159.72 159.69 161.30	955.06	2.95
	2780.6	(7.76) (28.34) (51.53) (81.09) (113.60) (161.29)	161.30	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	8.4	277.26	277.26	1.00
	2793.4	(277.26)	277.35	
1	7.2	219.58 218.45	438.03	1.26
	2798.0	(151.36) (140.40)	223.80	
2	6.9	233.66 236.12 232.14	701.91	1.17
	2799.3	(110.94) (202.86) (54.90)	237.98	
3	6.6	206.54 210.27 207.96 204.73	829.50	1.32
	2798.4	(95.31) (184.14) (102.05) (28.47)	211.59	
4	6.0	177.38 179.51 177.16 176.89 174.04	884.98	1.54
	2800.5	(77.21) (153.95) (108.94) (61.01) (16.90)	180.21	
5	6.2	174.01 176.05 174.87 172.91 171.80 172.22	1041.85	1.57
	2798.9	(73.51) (158.21) (120.71) (73.77) (40.73) (11.48)	176.61	

Tabela B5: Rezultati raspoređivanja primenom PNE pretraživanja unatrag počev od slučajnog rešenja

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	3.5	259.17	259.17	1.00
	2796.9	(259.17)	259.28	
1	3.5	172.11 179.69	351.80	1.44
	2796.9	(61.06) (179.69)	179.73	
2	3.5	144.43 147.36 150.46	442.25	1.72
	2796.9	(29.21) (85.38) (150.45)	150.46	
3	3.5	107.84 110.30 110.60 112.16	440.90	2.31
	2796.9	(14.94) (47.03) (73.68) (112.16)	112.37	
4	3.5	128.37 128.27 129.22 130.75 132.01	648.62	1.96
	2796.9	(13.77) (42.47) (67.51) (95.52) (131.97)	132.07	
5	3.5	99.12 100.08 100.55 101.17 102.09 102.70	605.72	2.52
	2796.9	(8.10) (25.69) (43.92) (56.24) (78.76) (102.14)	102.86	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	6.5	116.49	116.49	1.00
	2801.0	(116.49)	116.49	
1	6.3	126.13 127.75	253.89	0.91
	2803.2	(86.92) (86.39)	129.06	
2	6.4	128.15 129.40 127.36	384.91	0.90
	2797.5	(71.59) (104.55) (40.21)	130.61	
3	5.7	112.84 114.02 113.21 111.79	451.86	1.02
	2807.0	(58.98) (93.62) (63.94) (20.88)	115.21	
4	5.9	107.31 109.42 108.77 106.35 106.45	538.30	1.06
	2806.6	(51.89) (94.90) (74.15) (41.73) (15.38)	109.78	
5	6.0	95.24 95.78 95.15 94.52 94.76 93.27	568.71	1.22
	2800.4	(52.85) (79.35) (60.43) (43.27) (27.57) (8.69)	96.19	

Tabela B6: Rezultati PNE sa popravljenim opterećenjem među procesorima počev od slučajnog polaznog rešenja i pretraživanjem cele okoline

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	3.8	693.76	693.76	1.00
	2780.5	(693.76)	698.73	
1	3.8	355.52 354.01	709.53	1.95
	2780.5	(355.50) (338.85)	359.95	
2	3.8	251.44 253.78 250.52	755.75	2.73
	2780.5	(223.11) (253.74) (218.88)	258.00	
3	3.8	203.04 206.51 205.15 204.55	819.26	3.36
	2780.5	(153.50) (202.69) (192.85) (146.95)	210.87	
4	3.8	167.32 167.53 168.61 166.76 164.37	834.58	4.11
	2780.5	(153.54) (155.08) (168.49) (140.74) (78.28)	171.81	
5	3.8	154.98 153.74 155.11 154.58 152.13 152.04	922.59	4.47
	2780.5	(153.47) (111.62) (145.93) (138.02) (68.83) (78.29)	155.67	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	8.5	447.07	447.07	1.00
	2786.9	(447.07)	451.24	
1	8.5	280.61 283.88	564.49	1.57
	2789.5	(162.63) (201.42)	300.18	
2	7.0	247.70 250.48 251.69	749.87	1.78
	2792.4	(121.06) (125.85) (193.43)	256.80	
3	6.4	232.16 234.44 233.54 236.88	937.02	1.89
	2793.4	(103.54) (128.85) (129.29) (193.95)	240.18	
4	6.4	270.09 264.87 268.07 267.55 270.78	1341.36	1.65
	2789.3	(176.15) (88.59) (142.03) (153.14) (170.94)	277.90	
5	6.9	275.81 274.10 273.40 274.36 276.33 274.63	1648.64	1.62
	2786.5	(177.33) (113.33) (148.24) (179.25) (204.36) (122.64)	278.29	

Tabela B7: Rezultati PNE sa popravljenim opterećenjem među procesorima počev od slučajnog polaznog rešenja i pretraživanjem unapred

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	4.0	390.03	390.03	1.00
	2781.4	(390.03)	390.06	
1	4.0	207.94 206.71	414.65	1.88
	2781.4	(207.94) (182.63)	207.96	
2	4.0	143.73 144.44 143.37	431.54	2.70
	2781.4	(128.72) (144.15) (117.99)	144.52	
3	4.0	117.77 118.98 118.23 117.77	472.75	3.28
	2781.4	(89.55) (118.98) (97.74) (85.27)	118.97	
4	4.0	116.94 116.25 116.18 115.08 114.86	579.32	3.33
	2781.4	(116.89) (87.99) (95.89) (76.98) (41.51)	116.94	
5	4.0	89.97 89.24 89.84 89.52 88.99 88.35	535.92	4.34
	2781.4	(89.50) (63.89) (83.98) (68.78) (44.10) (41.50)	89.99	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	7.3	177.97	177.97	1.00
	2772.5	(177.97)	178.74	
1	6.7	157.74 159.03	316.77	1.12
	2781.1	(108.94) (106.23)	162.44	
2	5.9	123.95 125.51 125.13	374.60	1.42
	2789.3	(78.91) (87.27) (79.54)	127.27	
3	5.5	111.57 111.29 112.55 111.82	447.24	1.58
	2785.7	(61.55) (62.00) (86.74) (78.66)	113.84	
4	5.2	95.06 95.03 95.73 96.07 96.00	477.88	1.85
	2784.4	(48.52) (35.34) (65.50) (63.82) (68.39)	97.38	
5	5.9	101.64 101.26 101.78 102.93 102.87 101.61	612.10	1.73
	2782.0	(56.48) (38.68) (60.47) (83.93) (73.39) (50.25)	104.05	

Tabela B8: Rezultati PNE sa popravljenim opterećenjem među procesorima počev od slučajnog polaznog rešenja i pretraživanjem unazad

	n.it.	BI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	3.8	315.59	315.59	1.00
	2737.9	(315.59)	316.82	
1	3.8	168.95 167.71	336.66	1.87
	2737.9	(168.94) (146.74)	170.38	
2	3.8	124.93 126.04 124.26	375.23	2.50
	2737.9	(100.60) (126.03) (89.54)	127.06	
3	3.8	100.00 101.20 100.59 99.30	401.09	3.12
	2737.9	(68.31) (101.17) (87.47) (59.59)	101.66	
4	3.8	84.54 84.63 85.43 84.38 83.27	422.24	3.69
	2737.9	(68.31) (73.26) (85.22) (65.58) (24.11)	85.56	
5	3.8	73.10 72.41 73.39 72.57 71.88 71.32	434.66	4.30
	2737.9	(68.54) (52.65) (73.17) (62.65) (35.74) (24.11)	73.66	

	n.it.	FI pretraživanje	$\sum$ CPU	
q	f_{min}	CPU time	WTime	S
0	6.5	117.17	117.17	1.00
	2801.0	(117.17)	117.20	
1	6.6	111.80 109.34	221.14	1.05
	2800.6	(89.20) (55.78)	113.02	
2	5.6	94.37 93.82 93.91	282.10	1.24
	2799.4	(64.40) (59.40) (55.05)	95.59	
3	5.8	91.57 91.20 91.72 91.29	365.78	1.28
	2798.5	(48.73) (45.21) (61.62) (58.95)	92.74	
4	6.0	81.30 80.47 81.22 81.69 81.66	406.35	1.43
	2795.5	(48.80) (27.67) (43.48) (52.80) (55.50)	82.70	
5	6.0	72.58 71.40 72.11 72.67 72.63 72.23	433.62	1.61
	2796.2	(40.10) (25.93) (35.83) (42.62) (56.32) (37.81)	73.31	

B2. Tabele u vezi sa kooperativnim radom VNS metoda

U tabelama B9 do B12 dati su rezultati kooperativnog rada različitih varijanti VNS metoda, dok su u preostalim tabelama prikazani rezultati nezavisnog rada četiri odnosno pet VNS procedura. Sve testirane metode polaze od CPES heurističkog rešenja i imaju jedinstven kriterijum zaustavljanja (maksimalno dozvoljeno vreme izvršavanja). Raspoređivanje se vrši na retkim grafovima ($\rho = 30\%$) sa poznatom dužinom optimalne raspodele (koja je data u tabeli 4.3).

Tabele vezane za kooperativni rad ilustruju neke zanimljive kombinacije varijanti koje su za pojedine vrednosti parametara davale najbolje rezultate. Kombinovanjem "najboljih metoda" dobijeni rezultati su se dodatno popravljali. Tabele sadrže iste podatke kao i odgovarajuće tabele u glavi 5.7., a kombinovane su po četiri VNS metode. Koje varijante VNS metode (sa kojim parametrima i za koje izabrane komunikacione tačke) su učestvovale u kooperaciji objašnjeno je u zaglavlju svake tabele. Korišćene skraćenice su iste kao u odeljku 5.7.3.

Tabele u kojima su prikazani rezultati nezavisnog izvršavanja sadrže u prvoj koloni (kao i sve druge tabele) broj zadataka u slučajno generisanom grafu koji se raspoređuje, u narednih četiri ili pet kolona broj VNS iteracija, dužine najboljih dobijenih raspodela za svaku od varijanti i vreme utrošeno na pretraživanje do trenutka kada je ta najbolja raspodela dobijena. U preostalih četiri (odnosno pet) kolona navedeno je ukupno vreme izvršavanja (izračunavanje, komunikacije i intervali čekanja) i posebno vreme izračunavanja za svaku od VNS metoda.

Sumarni podaci navedeni u poslednjoj vrsti predstavljaju prosečan broj iteracija i najbolju (u proseku) dužinu raspodele (bez obzira na to kojom varijantom VNS metode je dobijena svaka od raspodela koja ulazi u prosek). Prosečno najbolja raspodela dobija se kada se saberu najbolje raspodele u svakom redu i podele sa 10. Na kraju su data i prosečna vremena rada i izračunavanja za svaki od procesora. U tabelama B17 i B18 sumarni podaci sadrže i prosečne dužine raspodela dobijenih na svakom od procesora, što predstavlja referentan podatak za poređenja sa sekvencijalnim izvršavanjima.

Tabela B9: Rezultati za najbolje 3 FI VNS metode i jednu BI, komunikacije na k++: VNS1, VNS2, VNS3 FI i VNS1 BI

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	147	7.91	7.77	7.74	8.09	7.89	68	11	11	57
	776	7.51	7.76	7.74	8.08	7.89	15	3	2	18
100	239	95.01	92.31	93.35	98.43	90.98	116	24	16	83
	1033	88.20	92.31	93.35	98.42	90.97	38	11	9	30
150	457	527.72	519.15	549.99	531.02	526.31	224	47	37	149
	1286	500.12	519.11	549.99	531.00	526.30	77	21	19	55
200	293	784.97	784.28	798.01	812.77	776.13	141	35	15	102
	2126	750.67	784.25	798.01	812.76	776.11	53	19	10	45
250	298	1303.09	1310.60	1329.39	1346.16	1307.91	143	28	17	110
	2115	1254.38	1310.59	1329.39	1346.15	1307.89	45	11	6	40
300	357	2822.71	2600.54	2600.59	2920.92	2633.60	176	37	14	130
	2343	2507.84	2600.51	2600.59	2920.92	2633.58	88	24	7	78
350	381	5145.38	5169.86	5305.98	5316.43	5225.39	174	40	27	140
	3338	5003.47	5169.84	5305.97	5316.43	5225.33	72	18	18	71
400	429	7674.08	7785.35	7764.21	7936.71	7754.00	206	52	21	150
	3681	7501.62	7785.30	7764.20	7936.71	7754.00	122	35	14	92
450	481	16523.96	13009.05	13674.58	17174.77	13110.21	235	40	24	182
	2938	12503.77	13009.01	13674.57	17174.77	13110.18	130	24	17	102
500	657	20981.20	20919.61	21360.57	21692.00	20951.12	300	61	42	254
	3768	20108.60	20919.55	21360.57	21691.96	20951.08	223	45	36	203
q =4	373.9	5586.60	5219.85	5348.44	5783.73	5238.35				
	2340.4	5022.62	5219.82	5348.44	5783.72	5238.33				

Tabela B10: Rezultati za najbolje 3 FI VNS i jednu BI, komunikacije na k_{max}: VNS1, VNS2, VNS4 FI i VNS2 BI

n	f_{min}	CPU time					Poboljšanja			
	n.it.	Comput time					Novo najbolje			
50	14	11.60	11.27	12.05	11.43	11.94	5	5	1	3
	734	10.83	11.27	12.04	11.43	11.94	2	5	0	2
100	22	94.73	92.12	96.24	96.28	98.69	5	8	5	4
	1085	88.49	92.12	96.24	96.28	98.68	2	6	1	2
150	20	548.76	533.11	535.90	567.58	556.39	6	7	3	4
	1147	513.11	533.11	535.90	567.58	556.39	3	1	1	4
200	17	938.14	938.08	946.65	957.06	975.11	4	5	3	5
	1525	900.46	938.08	946.64	957.06	975.10	4	3	2	3
250	14	1591.91	1445.01	1476.97	1648.93	1474.03	4	4	2	4
	2511	1382.45	1445.01	1476.97	1648.93	1474.03	4	1	2	1
300	15	2891.32	2897.85	2968.37	2975.77	3009.68	4	5	2	4
	2895	2775.69	2897.85	2968.35	2975.77	3009.68	2	4	1	3
350	19	5389.73	5505.97	5590.13	5505.21	5497.93	5	6	4	4
	3342	5279.89	5505.96	5590.12	5505.21	5497.92	2	2	3	3
400	20	8107.17	7903.03	8408.32	8140.67	7906.18	6	9	1	4
	2633	7553.79	7903.02	8408.32	8140.66	7906.18	4	8	0	2
450	22	18861.11	13965.31	14980.65	19598.79	14214.52	9	8	2	3
	2715	13290.36	13965.29	14980.63	19598.79	14214.52	6	3	1	1
500	32	23075.91	23095.15	23969.72	23651.87	23087.04	16	7	4	5
	3035	22053.24	23095.11	23969.71	23651.85	23087.04	10	5	2	3
q =4	19.5	6151.04	5638.69	5898.50	6315.36	5683.15				
	2162.2	5384.83	5638.68	5898.49	6315.36	5683.15				

Tabela B11: Rezultati za najbolje 3 FI VNS (VNS1, VNS2, VNS3) i FI VND, komunikacije na $k++$

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	109	8.49	8.29	8.50	8.90	8.64	70	15	13	11
	762	7.51	8.29	8.50	8.89	8.64	19	4	4	3
100	153	96.35	94.68	97.23	100.04	98.95	104	19	12	18
	952	87.67	94.67	97.23	100.04	98.95	28	8	4	8
150	349	512.02	526.52	535.18	536.17	527.50	243	40	28	38
	1296	500.90	526.49	535.16	536.16	527.49	79	21	16	19
200	186	816.42	783.02	823.92	852.04	850.48	136	25	7	18
	1411	752.74	783.01	823.91	852.04	850.47	51	11	4	10
250	238	1539.68	1306.17	1324.16	1603.64	1346.36	150	41	21	26
	2398	1250.80	1306.14	1324.15	1603.64	1346.36	87	33	14	19
300	214	2584.67	2619.68	2701.09	2633.25	2629.46	140	31	19	24
	2469	2509.60	2619.66	2701.09	2633.25	2629.45	72	18	9	13
350	303	5349.04	5247.63	5255.53	5602.94	5349.28	198	52	27	26
	3224	5006.89	5247.54	5255.53	5602.94	5349.28	113	26	14	14
400	329	7807.88	7873.54	7853.30	7898.56	8159.98	207	68	23	31
	3724	7511.80	7873.52	7853.30	7898.56	8159.96	126	40	15	21
450	319	14431.71	13115.29	15076.34	13139.76	13525.48	233	47	15	24
	2720	12517.08	13115.22	15076.32	13139.75	13525.48	116	26	10	14
500	363	21282.30	20939.80	22037.89	22189.13	21413.12	249	57	30	27
	3336	20018.22	20939.76	22037.88	22189.11	21413.12	116	34	19	14
q =4	256.3	5442.86	5251.46	5571.31	5456.44	5390.92				
	2229.2	5016.32	5251.43	5571.31	5456.44	5390.92				

Tabela B12: Rezultati za najbolje 3 FI VNS (VNS1, VNS2, VNS4) i FI VND, komunikacije na k_{max}

n	n.it.	CPU time					Poboljšanja			
	f_{min}	Comput time					Novo najbolje			
50	16	7.94	7.90	7.89	8.20	7.51	5	5	5	1
	734	7.62	7.89	7.89	8.20	7.51	3	5	1	0
100	24	92.98	92.39	96.25	96.23	95.58	9	8	5	2
	1096	89.00	92.39	96.23	96.22	95.58	4	6	1	1
150	19	623.01	577.80	651.82	603.79	614.32	3	9	2	5
	1460	549.96	577.77	651.82	603.78	614.29	3	6	1	4
200	12	1238.13	1003.86	1062.20	1292.45	1049.02	2	3	4	3
	1500	946.54	1003.85	1062.19	1292.44	1048.99	2	1	2	3
250	11	1662.22	1593.72	1627.93	1735.19	1746.95	4	4	2	1
	2511	1510.86	1593.70	1627.93	1735.19	1746.95	4	1	2	0
300	14	3311.36	2995.68	3456.98	3076.67	3069.68	4	6	2	2
	2072	2837.94	2995.65	3456.98	3076.67	3069.67	2	5	1	0
350	15	9293.38	9303.61	9415.23	9564.23	9702.64	5	6	1	3
	3351	8879.93	9303.59	9415.23	9564.22	9702.64	4	5	0	2
400	19	8056.96	7938.77	8414.68	8162.50	8188.66	6	9	1	3
	2633	7508.09	7938.75	8414.68	8162.49	8188.65	5	5	0	2
450	23	19236.31	16265.19	16411.48	20125.87	17078.09	9	9	2	3
	2679	15525.61	16265.16	16411.47	20125.85	17078.08	6	4	1	0
500	18	27192.19	25227.41	26542.04	28430.62	25516.27	8	5	3	2
	3315	23815.54	25227.38	26542.03	28430.60	25516.27	6	4	2	1
q =4	17.1	7071.45	6500.63	6768.65	7309.57	6706.87				
	2135.1	6167.11	6500.61	6768.65	7309.57	6706.86				

Tabela B13: Rezultati nezavisnog izvršavanja 4 VNS, BI pretraživanje

n	VNS-it(f_{min})				CPU Time			
	t_{min}				Comput time			
50	5(813)	5(796)	5(813)	5(813)	6.66	6.64	6.05	6.20
	0.00	3.01	0.00	0.00	6.66	6.64	6.05	6.20
100	6(1297)	6(1189)	6(1301)	6(1292)	70.30	79.24	72.66	72.53
	70.30	40.21	36.03	39.87	70.30	79.24	72.66	72.53
150	9(1776)	7(1224)	7(1764)	8(1763)	403.35	402.92	403.88	411.67
	58.42	269.14	331.74	122.49	403.35	402.92	403.88	411.67
200	10(2186)	8(2106)	8(2142)	9(2143)	660.15	610.46	664.34	626.37
	482.94	305.55	664.34	626.37	660.15	610.46	664.34	626.37
250	11(2567)	9(2598)	9(2554)	10(2575)	1025.90	1019.69	1019.02	1026.76
	997.71	305.55	937.38	1026.76	1025.89	1019.69	1019.02	1026.76
300	13(3009)	10(2562)	10(2969)	12(3008)	2113.78	2018.52	2219.51	2046.57
	990.97	1810.36	2219.51	322.20	2113.78	2018.52	2219.51	2046.57
350	14(3376)	11(3360)	11(3350)	14(3405)	4004.96	4039.70	4023.40	4241.15
	3542.58	4039.70	3615.88	322.20	4004.96	4039.70	4023.40	4241.15
400	15(3807)	12(3803)	12(3809)	16(3818)	6233.50	6182.43	6142.60	6175.79
	3910.71	2332.95	5932.41	322.20	6233.49	6182.43	6142.60	6175.79
450	17(4227)	14(4229)	14(4234)	17(4234)	10365.18	10124.51	10121.38	10058.15
	5412.36	3743.76	5202.84	7067.89	10365.18	10124.51	10121.38	10058.15
500	21(4605)	15(4576)	16(4406)	19(4538)	16046.51	16036.89	16806.12	16704.46
	5412.36	16036.89	5709.55	7067.89	16046.51	16036.89	16806.12	16704.46
$q = 4$	11.60			2621.70	4093.03	4052.10	4147.90	4136.97
					4093.03	4052.10	4147.90	4136.97

Tabela B14: Rezultati nezavisnog izvršavanja 4 VNS, FI pretraživanje

n	VNS-it(f_{min})				CPU Time			
	t_{min}				Comput time			
50	5(814)	5(722)	5(802)	5(790)	6.29	6.41	6.55	6.42
	3.00	2.58	6.55	6.42	6.29	6.41	6.55	6.42
100	7(1306)	6(1308)	6(1262)	6(1278)	76.027	78.337	71.58	74.39
	17.49	2.58	259.16	74.39	6.02	8.33	71.58	74.39
150	9(1356)	7(1770)	8(1745)	7(1702)	404.09	481.04	405.72	410.41
	224.58	149.42	140.49	262.97	404.09	481.04	405.72	410.40
200	10(2158)	8(2192)	9(2002)	9(2170)	651.09	702.38	666.52	622.60
	593.96	702.38	666.52	145.04	651.09	702.38	666.52	622.60
250	12(2506)	9(2600)	10(2579)	10(2517)	1023.98	1510.51	1168.91	1033.78
	900.33	1510.51	1168.91	513.67	1023.97	1510.51	1168.91	1033.78
300	13(2949)	10(3001)	11(3006)	11(2994)	2231.85	3094.82	2040.32	2151.80
	1753.57	1510.51	1817.98	2151.80	2231.84	3094.81	2040.32	2151.80
350	15(3351)	11(3386)	12(3404)	13(3346)	4289.01	4328.55	4116.17	4097.15
	4289.01	4328.55	3053.56	3827.62	4289.01	4328.55	4116.17	4097.15
400	16(3779)	12(3819)	13(3803)	15(3696)	6102.11	12172.66	6319.48	6123.52
	5198.83	4328.55	3902.65	6123.52	6102.11	12172.66	6319.47	6123.52
450	18(4061)	13(4213)	14(4251)	16(3064)	10989.18	12270.95	10969.23	10602.74
	10989.18	4328.55	8405.77	10602.74	10989.18	12270.95	10969.22	10602.74
500	22(4567)	14(4605)	15(3335)	17(3476)	16239.57	31566.30	16243.62	16206.65
	8333.15	4328.55	16243.62	16206.65	16239.56	31566.30	16243.61	16206.65
q =4	10.90			2423.80	4201.32	6621.19	4200.81	4132.95
					4201.32	6621.19	4200.81	4132.95

Tabela B15: Rezultati nezavisnog izvršavanja 5 VNS, BI pretraživanje

n	VNS-it(f_{min})					CPU Time				
	t_{min}					Comput time				
50	5(813)	5(796)	5(813)	5(813)	5(813)	6.66	6.64	6.04	6.10	7.60
	0.00	3.02	0.00	0.00	0.00	6.66	6.64	6.04	6.10	7.60
100	6(1297)	6(1189)	6(1301)	6(1292)	6(1327)	70.45	79.30	72.64	72.61	77.72
	70.45	40.19	35.99	39.98	77.72	70.45	79.30	72.64	72.61	77.72
150	9(1776)	7(1224)	7(1764)	8(1763)	7(1776)	402.56	402.30	402.97	411.43	412.69
	58.07	268.97	331.08	122.54	134.99	402.55	402.30	402.96	411.43	412.68
200	10(2186)	8(2106)	8(2142)	9(2143)	8(2189)	660.04	610.27	663.19	626.06	603.74
	483.12	305.32	663.19	626.06	603.74	660.04	610.27	663.19	626.05	603.74
250	11(2567)	9(2598)	9(2554)	10(2575)	9(2567)	1001.53	1018.87	1031.10	1026.72	1012.94
	1001.53	305.32	948.93	1026.72	877.77	1001.53	1018.87	1031.10	1026.72	1012.94
300	13(3009)	10(2562)	10(2969)	12(3008)	10(3009)	2116.88	2017.98	2218.08	2047.02	2329.69
	993.70	1810.42	2218.08	321.88	2329.69	2116.88	2017.98	2218.08	2047.02	2329.69
350	14(3376)	11(3360)	11(3350)	14(3405)	11(3386)	4003.89	4041.96	4023.46	4241.48	4429.31
	3541.63	4041.96	3615.78	321.88	2329.69	4003.89	4041.96	4023.46	4241.48	4429.31
400	15(3807)	12(3803)	12(3809)	16(3818)	12(3811)	6236.21	6181.29	6142.05	6180.21	6231.06
	3912.75	2331.47	5932.06	321.88	3120.81	6236.20	6181.29	6142.05	6180.21	6231.06
450	17(4227)	14(4229)	14(4234)	17(4234)	13(4170)	10357.49	10141.09	10124.42	10060.85	10407.31
	5409.86	3748.74	5204.09	7066.08	6013.26	10357.48	10141.08	10124.41	10060.85	10407.30
500	21(4605)	15(4576)	16(4406)	19(4538)	15(4605)	16041.87	16045.12	16806.05	16719.74	16400.77
	5409.86	16045.12	5711.38	7066.08	6013.26	16041.87	16045.12	16806.05	16719.74	16400.77
q =5	9.60				2616.00	4089.76	4054.48	4149.00	4139.22	4191.28
						4089.76	4054.48	4149.00	4139.22	4191.28

Tabela B16: Rezultati nezavisnog izvršavanja 5 VNS, FI pretraživanje

n	VNS-it(f_{min})					CPU Time				
	t_{min}					Comput time				
50	5(814)	5(722)	5(802)	5(790)	5(740)	6.32	6.44	6.56	6.45	6.51
	3.02	2.58	6.56	6.45	5.23	6.32	6.44	6.56	6.45	6.51
100	7(1306)	6(1308)	6(1262)	6(1278)	6(1305)	76.24	78.31	71.70	74.54	72.91
	17.54	2.58	59.26	74.54	52.15	76.24	78.31	71.70	74.54	72.91
150	9(1356)	7(1770)	8(1745)	7(1702)	7(1549)	405.22	482.52	406.38	410.81	422.96
	225.47	149.56	140.54	263.29	422.96	405.22	482.52	406.38	410.81	422.96
200	10(2158)	8(2192)	9(2002)	9(2170)	8(1968)	652.10	703.65	667.94	622.17	613.88
	594.87	703.65	667.94	145.19	521.90	652.10	703.65	667.94	622.16	613.88
250	12(2506)	9(2600)	10(2579)	10(2517)	9(2561)	1024.87	1512.26	1172.65	1034.20	1031.66
	901.13	1512.26	1172.65	513.99	1031.66	1024.87	1512.26	1172.65	1034.20	1031.66
300	13(2949)	10(3001)	11(3006)	11(2994)	10(2958)	2233.84	3095.86	2042.84	2156.15	2081.34
	1756.00	1512.26	1820.23	2156.15	1850.27	2233.84	3095.86	2042.84	2156.14	2081.34
350	15(3351)	11(3386)	12(3404)	13(3346)	11(3342)	4291.64	4338.90	4118.88	4102.28	4406.55
	4291.63	4338.90	3055.94	3832.50	2813.96	4291.62	4338.90	4118.88	4102.28	4406.55
400	16(3779)	12(3819)	13(3803)	15(3696)	12(3800)	6104.12	12178.11	6325.03	6127.62	6184.14
	5201.38	4338.90	3906.48	6127.62	6184.14	6104.10	12178.11	6325.03	6127.62	6184.14
450	18(4061)	13(4213)	14(4251)	16(3064)	13(4193)	10997.24	12288.25	10972.77	10621.58	10226.91
	10997.24	4338.90	8408.22	10621.58	9769.25	10997.24	12288.25	10972.75	10621.58	10226.91
500	22(4567)	14(4605)	15(3335)	17(3476)	14(3274)	16241.48	31599.65	16253.69	16213.64	21028.31
	8329.91	4338.90	16253.69	16213.64	14629.56	16241.48	31599.65	16253.69	16213.63	21028.31
q =5	9.50				2413.90	4203.31	6628.39	4203.84	4136.94	4607.52
						4203.30	6628.39	4203.84	4136.94	4607.52

Tabela B17: Rezultati nezavisnog izvršavanja 5 VNS, FI pretraživanje, četvorostruko vreme izvršavanja

n	VNS-it(f_{min})					CPU Time				
	t_{min}					Comput time				
50	26(775)	8(730)	7(725)	6(749)	7(682)	30.09	30.60	30.03	30.29	30.67
	16.34	27.92	17.82	29.99	10.54	30.09	30.59	30.02	30.29	30.67
100	66(1027)	11(931)	11(1205)	10(1212)	11(1077)	350.10	350.78	361.61	357.17	356.50
	347.95	256.43	315.50	317.24	356.50	350.07	350.78	361.61	357.17	356.50
150	148(1256)	26(1204)	22(1655)	20(1512)	22(1155)	2001.08	2017.45	2000.83	2014.68	2012.55
	1957.33	1399.57	1071.96	1877.72	538.27	2001.04	2017.43	2000.83	2014.68	2012.55
200	200(2099)	32(1588)	26(2002)	26(1986)	26(2047)	3003.22	3010.14	3031.58	3014.19	3016.51
	783.59	2625.75	663.50	1785.15	2546.46	3003.20	3010.14	3031.56	3014.19	3016.49
250	261(2549)	51(2569)	32(2569)	32(2498)	33(2516)	5002.68	5020.91	5027.31	5009.84	5034.82
	3302.39	1153.94	2630.93	3780.61	1250.00	5002.63	5020.91	5027.31	5009.83	5034.81
300	313(2992)	56(2102)	36(2783)	33(2275)	40(1870)	10011.84	10056.71	10183.94	11000.89	10087.56
	2294.77	9943.47	8761.08	9456.14	8035.60	10011.78	10056.71	10183.93	11000.88	10087.55
350	379(3363)	70(2412)	41(3377)	36(2600)	46(3323)	20018.09	20113.21	20082.48	20028.04	20060.81
	5195.01	19704.08	9796.91	18567.95	17475.64	20018.03	20113.20	20082.47	20028.04	20060.80
400	448(3714)	74(2279)	48(3727)	43(3676)	54(3762)	30018.25	30087.98	30811.09	30057.68	30017.29
	19564.31	29154.78	30811.09	15678.35	12764.16	30018.22	30087.96	30811.09	30057.66	30017.28
450	542(4176)	81(3000)	54(4157)	44(2817)	59(3136)	50016.15	50053.30	50117.68	50122.10	50387.91
	23452.32	49060.11	26808.80	48546.87	50387.91	50015.99	50053.30	50117.67	50122.09	50387.91
500	627(3342)	86(2978)	55(3162)	45(3216)	67(3283)	80041.25	80658.78	80117.44	80903.41	80111.82
	58927.75	79785.31	79500.14	79352.19	78036.74	80041.16	80658.77	80117.44	80903.41	80111.82
q =5	36.50				1921.00	20049.28	20139.99	20176.40	20253.83	20111.64
av. SL	2529.3	1979.3	2536.2	2254.1	2285.1	20049.22	20139.98	20176.39	20253.82	20111.64

Tabela B18: Rezultati nezavisnog izvršavanja 5 VNS, BI, četvorostruko vreme izvršavanja

n	VNS-it(f_{min})					CPU Time				
	t_{min}					Comput time				
50	23(799)	7(708)	6(800)	6(764)	7(810)	30.10	30.10	31.63	30.64	30.75
	21.69	13.08	22.90	17.51	16.04	30.09	30.10	31.63	30.64	30.75
100	53(1271)	10(1195)	10(1301)	10(1283)	12(1329)	350.75	354.40	358.79	353.02	351.24
	228.29	354.40	35.56	347.82	38.12	350.75	354.40	358.78	353.01	351.24
150	111(1357)	20(1082)	18(1691)	19(1758)	24(1768)	2001.06	2010.64	2006.77	2009.77	2014.16
	1482.86	2010.64	908.19	1515.37	1796.38	2001.03	2010.63	2006.77	2009.76	2014.15
200	147(2115)	24(1638)	23(2138)	22(1956)	29(2085)	3004.60	3034.57	3004.90	3172.02	3020.32
	2691.55	2004.91	709.91	3172.02	418.13	3004.59	3034.57	3004.90	3172.02	3020.32
250	180(2574)	31(2518)	28(2537)	25(2534)	34(2578)	5001.58	5068.69	5068.50	5073.87	5100.00
	1002.93	4604.92	1491.87	4377.22	2579.32	5001.54	5068.67	5068.50	5073.86	5099.99
300	228(2981)	39(2863)	35(2969)	29(2958)	40(2963)	10013.05	10196.81	10128.94	10033.92	10060.18
	3917.71	7862.20	2222.36	6518.96	2714.68	10013.02	10196.81	10128.93	10033.91	10060.18
350	278(3314)	44(3283)	42(3350)	34(3335)	46(3399)	20024.47	20150.02	20069.90	20256.03	20145.82
	11498.98	17061.83	3606.95	9524.63	20145.82	20024.45	20150.00	20069.90	20256.01	20145.82
400	335(3795)	53(3729)	47(3774)	38(3751)	53(3817)	30023.93	30299.62	30375.40	30047.25	30177.49
	26891.39	30299.62	18351.62	28443.12	23288.13	30023.90	30299.62	30375.40	30047.25	30177.49
450	400(4234)	55(2837)	55(4180)	40(2948)	59(4215)	50087.39	50206.82	50454.61	50049.25	50786.15
	37366.61	49972.35	46611.17	48472.96	46250.49	50087.32	50206.81	50454.60	50049.24	50786.13
500	459(3230)	60(3168)	62(3462)	44(3409)	68(3323)	80012.57	80285.06	80030.68	80193.27	80903.42
	52427.13	73300.25	72295.55	70279.87	55172.04	80012.49	80285.04	80030.68	80193.26	80903.39
q =5	37.20				2302.10	20054.95	20163.67	20153.01	20121.90	20258.95
av. SL	2567.0	2302.1	2620.2	2469.6	2628.7	20054.92	20163.67	20153.01	20121.90	20258.95