Univerzitet u Beogradu
Matematički fakultet

Jozef J. Kratica

PARALELIZACIJA GENETSKIH ALGORITAMA ZA REŠAVANJE NEKIH NP - KOMPLETNIH PROBLEMA

Doktorska disertacija

B e o g r a d

2000.

Mentor: **Prof. dr Dušan Tošić**
Matematički fakultet u Beogradu

Članovi komisije: **Prof. dr Đorđe Dugošija**
Matematički fakultet u Beogradu

Prof. dr Vera Kovačević-Vujičić
Fakultet Organizacionih nauka u Beogradu

Datum odbrane: _____

Ovaj rad posvećujem mojim najdražim

ocu **Jozefu Kratici**

dedi **Jozefu Kratici**

koji na žalost više nisu među nama.

Paralelizacija genetskih algoritama za rešavanje nekih NP - kompletnih problema

Rezime

U ovom radu je opisana primena genetskih algoritama (GA) za rešavanje nekoliko problema kombinatorne optimizacije: prost lokacijski problem, problem dizajniranja mreže neograničenog kapaciteta i problem izbora indeksa. Dati NP-kompletni problemi ne samo što nalaze veliku primenu u praktičnim oblastima kao što su proizvodnja, transport, planiranje zaliha i trgovina, već i u specifičnim oblastima vezanim za računare, odnosno, programiranje kao što su telekomunikacije, lokalne računarske mreže, globalne računarske mreže, Internet i baze podataka.

Sekvencijalna GA implementacija sadrži fleksibilno realizovane razne varijante genetskih operatora selekcije, ukrštanja i mutacije. Osim njih dato je i nekoliko funkcija prilagođenosti, razni kriterijumi završetka rada GA i različite politike zamene generacija. Sve zajedničke GA promenljive su grupisane u strukturu, pa je laka dopuna funkcijama koje zavise od prirode samog problema, što obezbeđuje relativno jednostavno rešavanje novih problema.

Opisana aplikacija je zatim paralelizovana i implementirana na mreži računara. Za paralelizaciju je korišćen MPI standard (Message Passing Interface), koji postaje prihvaćeni standard za veliki broj višeprocesorskih arhitektura. Za razvoj, testiranje i izvršavanje programa je korišćena mreža PC kompatibilnih radnih stanica. Implementiran je distribuirani model paralelnog GA, gde svaki proces izvršava lokalni genetski algoritam na svojoj sopstvenoj potpopulaciji, uz povremenu razmenu najboljih jedinki, između susednih procesa.

Za dalje poboljšanje performansi sekvencijalne i paralelne implementacije razvijena je posebna tehnika, nazvana keširanje GA. Ona ne utiče na rezultate koje dobijamo, već samo smanjuje vreme izvršavanja. Keširanje GA je opšta tehnika, primenljiva na širokom spektru problema, a relativno dobre rezultate daje baš u primeni na NP-kompletne probleme.

Rezultati izvršavanja na datim problemima pokazuju da je sekvencijalna GA implementacija postigla rezultate koji su u svim slučajevima uporedivi sa heuristikama i ostalim metodama poznatim u literaturi, a u nekim slučajevima i bolji. Paralelizacija i implementacija korišćenjem MPI standarda je dalje poboljšala performanse GA, i učinila programski kod lako prenosivim na razne tipove paralelnih računara.

Ključne reči: *genetski algoritmi, paralelni algoritmi, NP-kompletni problemi, prost lokacijski problem, problem dizajna mreže neograničenog kapaciteta, problem selekcije indeksa, keširanje GA*.

Parallelization of the genetic algorithms for solving some NP - complete problems

Abstract

The genetic algorithm (GA) method for solving several complexes combinatorial optimization problems: simple plant location problem, uncapacitated network design problem and index selection problem is described. These NP-complete problems have various applications, not only in practical areas such as manufacturing, traffic, distribution planning or trading, but also in specific areas associated with computers and programming such as database design, telecommunications, local area networks, global area networks and Internet.

Sequential GA implementation contains various genetic operators of selection crossover and mutation realized on flexible way. Many variations of fitness function, GA finishing criteria and several generation replacement schemes are given too. All common GA variables are grouped into one structure, and adding the problem's depended functions is relatively easy. On this manner a fast way for solving new problems is provided.

This implementation is parallelized, and implemented on multiprocessor system, using the MPI standard (Message Passing Interface). MPI is fast and flexible practical interface for many multiprocessor architectures. For the program developing and performance testing the network of PC workstations is used. Distributed model for parallelizing GA is implemented. In this model every process executes GA on its own subpopulation with occasional exchange of good individuals between neighbor processes.

For further performance improvement of the sequential and parallel implementation a new technique, called caching GA is developed. Caching has not influence to the quality of solution, but only improve GA running time. Caching GA is a general technique, (applicable for solving many problems by genetic algorithms) giving, relatively, good results applying on NP-complete problems.

The computational results for given problems are presented to indicate that the sequential GA implementation is capable to produce high-quality solutions. In some cases this technique is superior to all known heuristic and other methods in the literature. Parallelization and implementation by MPI standard provides further improving performances. The compatible program code is obtained and could be executed on more powerful parallel platform.

Key words: *Genetic Algorithms, Parallel Algorithms, NP-complete problems, Simple Plant Location Problem, Uncapacitated Network Design Problem, Index Selection Problem, Caching GA*.

PREDGOVOR

Rad se sastoji od 8 poglavlja i 5 dodataka:

Uvodno poglavlje daje osnovne informacije o NP-kompletnim problemima, genetskim algoritmima, višeprocesorskim računarima i paralelnim algoritmima. Svi ovi pojmovi se suštinski koriste nadalje u ovom radu.

Opis sekvencijalne GA implementacije koja je realizovana na personalnim računarima je naveden u 2. poglavlju, a paralelne GA implementacije u 3. poglavlju.

Četvrto poglavlje sadrži detaljan opis keširanja genetskih algoritama, metode kojom se poboljšava vreme izvršavanja bez uticaja na ostale parametre GA. Ova metoda je samostalno razvijena i implementirana od strane autora ovog rada.

U 5., 6. i 7. poglavlju su opisani rezultati primene GA u rešavanju prostog lokacijskog problema, problema dizajna mreže neograničenog kapaciteta i problema izbora indeksa. Kratak pregled rezultata sa zaključkom je dat u 8. poglavlju.

U dodacima su redom dati: uputstvo za korišćenje date implementacije, detaljan opis sekvencijalnog i paralelnog GA, pojedinačni rezultati izvršavanja i osnovne karakteristike MPI standarda koje su korišćene u paralelizaciji.

Želeo bih da se zahvalim:

Sestri Lidi na pomoći u slaganju teksta i obradi crteža, a takođe i majci Lidi na pruženoj podršci.

Mentoru Prof. dr Dušanu Tošiću na izuzetnom strpljenju, veoma korisnim sugestijama i stručnim savetima.

Članovima komisije Prof. dr Đorđu Dugošiji i Prof. dr Veri Kovačević-Vujičić na vrlo pažljivom čitanju rukopisa i korisnim savetima.

Prof. Donald Erlenkotter-u (The Anderson School at UCLA) na obezbeđivanju programskog koda DUALOC iskorišćenog za poređenje performansi u odnosu na GA implementaciju za rešavanje prostog lokacijskog problema.

Autorima javno dostupnog programskog paketa WMPI (verzija 1.2) nastalog na Instituto Supererior de Engenharia de Coimbra (Portugal), koji je iskorišćen pri paralelizaciji genetskog algoritma i implementaciji pomoću MPI standarda.

Kolegama mr Vladimiru Filipoviću, mr Ivani Ljubić, Desimiru Pavloviću i dr Darku Vasiljeviću na korisnim savetima i sugestijama u vezi genetskih algoritama.

Profesoru Tatjani Jeremić na pomoći u lekturi radova na engleskom jeziku koji su objavljeni u inostranim časopisima.

Dekanu Matematičkog fakulteta Prof. dr Nedi Bokan, upravniku Računarske laboratorije dr Dušku Vitasu, mr Đuri Mišljenoviću i Saši Malkovu što mi je omogućeno da testiram performanse PGANP implementacije na računarskoj mreži Matematičkog fakulteta.

Bibliotekama Matematičkog fakulteta, Matematičkog instituta SANU, Saobraćajnog fakulteta, Elektrotehničkog fakulteta, Građevinskog fakulteta i Fakulteta organizacionih nauka u Beogradu, na pomoći u sakupljanju

neophodne literature. Posebno se zahvaljujem bibliotekarima Branki Bubonji (Matematički Institut) i Ljiljani Rokić (Saobraćajni Fakultet).

Dragim prijateljima mr Radošu Bakiću, Radetu Samardžiji, dr Zoranu Petroviću, dr Mirjani Borisavljević, mr Zdravku Stojanoviću, dr Slobodanu Radojeviću, mr Uglješi Bugariću, Vladimiru Nikiću, dr Vladimiru Dragoviću, Prof. dr Marijani Milačić, Prof. dr Slobodanu Nešiću, Prof. dr Velimiru Simonoviću, mr Stevi Steviću, Vojislavu Pantiću, dr Darku Milinkoviću, dr Slobodanu Simiću, dr Ratku Ivanovu, mr Radivoju Protiću i Vesni Grbić na podršci u izradi ovog rada.

Mojim nekadašnjim dragim profesorima:

- Mirjani Janković, Julijani Putnik, Anđelki Krunić, Roksandi Pajović i Ilonki Milošević;
- Slobodanu Tmušiću, Branki Đerasimović, Nenadu Lazareviću, dr Vesni Jevremović, Jasminki Nikolić, Vesni Jurašin, Aleksandru Cvetkoviću, Nevenki Spalević, Milanu Čabarkapi, Dušanu Veljkoviću i Životi Joksimoviću;
- Prof. dr Žarku Mijajloviću, Prof. dr Aleksandru Lipkovskom, Prof. dr Dragoslavu Ljubiću, Prof. dr Milošu Arsenoviću, Prof. dr Branki Alimpić, dr Žoltu Zolnai-u, Prof. dr Zoranu Ivkoviću, Prof. dr Slaviši Prešiću, Prof. dr Nedeljku Parezanoviću, Prof. dr Aleksandru Jovanoviću, Prof. dr Miodragu Potkonjaku, Prof. dr Mioljubu Nikiću, Prof. dr Branislavu Mirkoviću, Prof. dr Desanki Radunović, Prof. dr Gordani Pavlović-Lažetić, Prof. dr Ljubomiru Protiću, Prof. dr Lavu Ivanoviću, Akademiku Prof. dr Milosavu Marjanoviću i Prof. dr Savi Krstiću.

Kandidat
Beograd, februar 2000. mr Jozef Kratica

1. UVOD

Ekspanzija računarske industrije je doprinela napretku, ne samo programerskih, već i velikog broja srodnih matematičkih disciplina kao što su: kombinatorna optimizacija, numerička analiza, diskretna analiza, teorija algoritama, teorija konačnih automata. Iako većina ovih matematičkih disciplina nije direktno vezana za korišćenje računara, pravi smisao svog postojanja i primene su doživele tek u poslednjih pedesetak godina.

Kombinatorna optimizacija beleži posebno brz napredak, pa je u ovom trenutku jedna od oblasti u kojoj se intenzivno radi. Nagli razvoj računara je doneo mogućnosti testiranja programa na praktičnim primerima relativno velike dimenzije. Na taj način, ne samo što su rešavani problemi u praksi, već su dobijeni rezultati bili najčešće i predmet novog teorijskog razmatranja, koje je imalo za cilj da teorijski dokaže neke od praktičnih aspekata dobijenog rešenja. Dobijeni teorijski rezultati, potrebe industrije za sve boljim performansama pri rešavanju problema sve većih dimenzija i računarski resursi sposobni za skladištenje radikalno većeg broja podataka, generišu potrebu za potpuno novim metodama rešavanja, koje zatim ponovo generišu nove teorijske rezultate. Na taj način se zatvara krug, pri čemu se stalno poboljšava kako kvalitet rešenja, tako i teorijska zasnovanost metode.

Često se dešava da pojedini algoritmi budu tako dobro projektovani i optimizovani pri implementaciji, da u dužem vremenskom intervalu daju odlične rezultate, koji su zadovoljavajući čak i za najzahtevnije korisnike. Međutim, kasnije se dimenzija datog problema u praktičnim primenama toliko poveća, da rešavanje tom metodom postaje sporo ili čak vremenski nedostižno. Zbog toga se stalno vrši poboljšavanje već postojećih, kao i neprekidna potraga za potpuno novim metodama. Navedimo samo neke od najpoznatijih: genetski algoritmi (genetic algorithms - GA), simulirano kaljenje (simulated annealing), neuralne mreže (neural networks), tabu pretraživanje (tabu search) i Lagrange-ova relaksacija. Nešto detaljnije o ovim tehnikama će biti rečeno u narednim odeljcima.

U nekim slučajevima je takođe moguće poboljšavanje performansi algoritama, njegovom paralelizacijom i implementacijom na višeprocesorskom računaru. Neke metode kombinatorne optimizacije su vrlo pogodne za paralelizaciju, kao na primer genetski algoritmi, pa je ubrzanje u odnosu na odgovarajući sekvencijalni algoritam ponekad čak blisko broju upotrebljenih procesora, što je teorijski optimum. U nekim primenama je višeprocesorski računar sastavljen od više personalnih računara povezanih u mrežu, pokazao zavidne performanse uz nisku nabavnu cenu ([Kon98]).

Ovo uvodno poglavlje daje osnovne informacije o najvažnijim oblastima koje su zastupljene u radu, a to su:

- Složenost algoritama, NP-kompletni problemi i načini za njihovo rešavanje dati u odeljku 1.1;
- Kratak opis GA i njihova primena u rešavanju NP-kompletnih problema prezentirani u odeljku 1.2;
- Višeprocesorske arhitekture i paralelni algoritmi opisani u odeljku 1.3 .

1.1 Složenost algoritama i NP-kompletni problemi

Čest je slučaj sintaksno i semantički ispravnih programa koji korektno rade na demonstracionim test-primerima manjih dimenzija, ali izvršavanje na instancama veće dimenzije datog problema preuzetih iz prakse traje neprihvatljivo dugo ili je nedostižno. Tada se, prirodno, postavlja pitanje da li je moguće primeniti neki bolji (brži) algoritam.

Zbog toga se javila potreba za kvantifikovanjem vremena izvršavanja svakog složenijeg algoritma odnosno programa. Pri tome se nužno javlja ideja o klasifikaciji algoritama po brzini izvršavanja a takođe i klasifikacija problema koji se njima rešavaju. Vrlo važan aspekt svakog algoritma je pored vremena izvršavanja (vremenska složenost) takođe i njegova potrošnja memorijskog prostora (prostorna složenost). Za opšte infomacije o algoritmima i njihovim primenama u raznim oblastima, mogu se pogledati [Brs88], [Crm90] i [Man91] a od domaćih autora [Uro96] i [Pau97]. Osim njih mogu se koristiti i knjige [Grn72] i [Nem89] gde je dat sveobuhvatan pregled algoritama iz oblasti kombinatorne optimizacije i celobrojnog programiranja.

1.1.1 Vremenska složenost algoritama

Vremenska i prostorna složenost se ocenjuju asimptotski u terminima elementarnih operacija i podataka hipotetičkog računara. Postupa se na taj način da bi se izbegle razlike u konstrukciji i performansama konkretnih računara, a kao posledica pojavljuje se mogućnost da teorijski vrednujemo kvalitet datog algoritma, bez obzira na kojoj se platformi izvršava.

Definicija 1.1 Za dati problem, čiji su ulazni podaci dimenzije *n,* kažemo da je određeni algoritam vremenske složenosti O(g(n)), ako vreme izvršavanja algoritma u *najgorem slučaju* ne prelazi vrednost $c \cdot g(n)$, gde je *c* konstanta.

Definicija 1.2 Algoritam je *polinomske složenosti* po vremenu izvršavanja, ako je vremenske složenosti najviše $O(n^k)$, za neku konstantu *k*.

Na primer, polinomske složenosti su algoritmi za:

- pretragu uređenog niza, nalaženje Fibonacci-jevih brojeva, ... (složenosti O(log n));
- pretragu neuređenog niza, zbir elemenata niza, ... (složenosti O(n));
- sortiranje elemenata niza, brzu Furier-ovu transformaciju (FFT), ... (složenosti O(n log n));
- sabiranje matrica, množenje matrice vektorom, nalaženje najkraćeg puta (Dijkstra-in algoritam [Dij59]), nalaženje minimalnog drveta razapinjanja ... (složenosti $O(n^2)$) itd.

Definicija 1.3 Algoritam je *eksponencijalne složenosti* po vremenu izvršavanja, ako nije polinomske složenosti.

Na primer, eksponencijalne složenosti su algoritmi sa brojem operacija O(n!), $O(2^n)$, itd. Pošto eksponencijalna funkcija, sa porastom *n*, mnogo brže raste od polinomske, algoritmi polinomske složenosti su za veće dimenzije *n* jedini efikasni u praksi!

Definicija 1.4 Problem kod koga je rešenje oblika DA/NE, se naziva problem odlučivanja (decision problem).

Primetimo da se svaki problem kombinatorne optimizacije može preformulisati tako da bude problem odlučivanja. Na primer ako je rešenje nekog problema optimizacije *min*(f(x)) = min_1 tada on formulisan kao problem odlučivanja glasi: "Da li važi tvrđenje $((\forall x)\, f(x) \geq min_1) \wedge ((\exists x_1)\, f(x_1) = min_1)$ ".

Definicija 1.5 Problem pripada klasi složenosti **P**, i nazivamo ga problemom *polinomske složenosti*, ako se rešava, u opštem slučaju, nekim od poznatih algoritama polinomske složenosti.

Definicija 1.6 Problem pripada klasi **NP**, i nazivamo ga problemom *nedeterminističke polinomske složenosti*, ako se rešenje datog problema može verifikovati algoritmom polinomske složenosti. Preciznije, za unapred dato rešenje se utvrđuje se da li su ispunjeni svi uslovi problema. Pri tome je potrebno da on bude zadat kao problem odlučivanja, da bi odgovor bio i formalno (matematički) korektan.

Očigledno je da $\mathbf{P} \subseteq \mathbf{NP}$, jer za svaki problem polinomske složenosti postoji poznat algoritam polinomske složenosti koji ga rešava, pa time trivijalno takođe i verifikuje dato rešenje. Pitanje o tome da li je $\mathbf{P} \subset \mathbf{NP}$ ili **P=NP** nije ni do danas razrešeno, i pored mnogobrojnih pokušaja da se dokaže jedno od tih tvrđenja. Iako ne postoji matematički dokaz da je $\mathbf{P} \neq \mathbf{NP}$, višegodišnji rezultati istraživanja navode na hipotezu da **NP** klasa problema sadrži i neke probleme koji ne pripadaju klasi **P**. Detaljnije razmatranje svih aspekata vezanih za složenost algoritama daleko izlazi izvan okvira ovog rada, a osim već navedenih referenci može se naći i u [Pap82] i [Ynn97].

1.1.2 NP-kompletni problemi i njihovo optimalno rešavanje

Definicija 1.7 Za problem Q_1 kažemo da je svodiv u polinomskom vremenu na problem Q_2 ako postoji algoritam polinomske složenosti koji pretvara svaku interpretaciju problema Q_1 u interpretaciju problema Q_2 tako da imaju analogno zajedničko rešenje.

Svođenjem u polinomskom vremenu pokazujemo da složenost polaznog problema nije veća od složenosti drugog problema, ne uzimajući u obzir vreme izvršavanja algoritma za svođenje, koje je polinomske složenosti. Ovakav metod predstavlja vrlo moćan aparat za klasifikaciju složenosti NP problema za koje nije do sada poznat algoritam za rešavanje u polinomskom vremenu.

Definicija 1.7 Problem odlučivanja nazivamo *NP-kompletnim* ako:

- pripada klasi NP;
- svi ostali NP problemi se mogu algoritmom polinomske složenosti svesti na dati problem.

Pomoću prethodne definicije, unapred prihvatajući za neki polazni problem da je NP-kompletan, relativno lako nalazimo ostale NP-kompletne probleme, korišćenjem metoda svođenja u polinomskom vremenu. Obično se u literaturi za polazni NP-kompletan problem uzima problem zadovoljivosti Boole-ovog izraza (satisfiability problem).

Korišćenjem prethodnog načina, u poslednjih trideset godina je za veliki broj raznovrsnih problema, dokazano da pripadaju klasi NP-kompletnih problema. Za sve probleme kombinatorne optimizacije, koji su rešavani u ovom radu, je dokazano da su NP-kompletni. Odgovarajući dokazi se mogu naći za: prost lokacijski problem u [Krr83], problem dizajna mreža neograničenog kapaciteta u [Hlm86], problem selekcije indeksa u [Com78].

Teorija NP-kompletnih problema je praktično nastala radovima [Cook71] i [Krp72], a već tokom 70-tih godina je dokazano za više od 300 problema da pripadaju toj klasi. U [Gar79] je dat prvi pregled teorijskih rezultata kao i sveobuhvatan spisak do tada poznatih NP-kompletnih problema. Oni su podeljeni u 12 tematskih celina, a takva podela se održala sve do danas. Dati spisak, osim kratkog opisa svakog od problema, sadrži i značajne reference iz teorije NP-kompletnih problema. Kasnije je nastavljeno sa istraživanjima na tom polju, a trenutno aktuelni presek date oblasti sa više od 200 klasifikovanih problema se može naći u [Cre97]. Za detaljno upoznavanje teorije NP-kompletnih problema i raznih praktičnih aspekata, mogu poslužiti još i [Brs88], [Crm90], [Man91] i [Jun98].

Teorijsko proučavanje NP-kompletnih problema ima i veliki praktičan značaj, jer pokazuje da optimalno rešavanje takvih problema zahteva vreme izvršavanja koje raste eksponencijalno sa veličinom problema. To rezultuje dobrim rešavanjem "školskih" test-primera male dimenzije. Međutim, pri izvršavanju realnih instanci veće dimenzije, koji se pojavljuju u praksi, iako je algoritam matematički korektan, dobijanje rešenja je vremenski nedostižno, čak i na najmodernijim računarima visokih performansi.

I pored gorepomenutih karakteristika optimalnog rešavanja NP-kompletnih problema, složenost algoritama se ocenjuje asimptotski i računa za najnepovoljniju varijantu njegovog izvršavanja. To ipak ostavlja određene mogućnosti za relativno uspešnu primenu metoda koje daju optimalno rešenje NP-kompletnih problema, u sledećim slučajevima:

- Iako je u opštem slučaju NP-kompletan, mogu postojati određeni specijalni slučajevi kada je problem polinomske složenosti. Na primer, u opštem slučaju je problem bojenja grafa NP-kompletan, ali ako se posmatra bojenje samo sa 2 boje, on je polinomijalan.
- Neki algoritmi, iako im je složenost u najgorem slučaju eksponencijalna, u proseku mogu imati polinomijalan broj operacija. Na primer, poznati *simpleks metod* za rešavanje problema linearnog programiranja, iako eksponencijalne složenosti, najčešće ima polinomijalan broj koraka.
- U određenim slučajevima je moguće da konstanta kod eksponencijalnog člana bude ekstremno mala. Tada se rešenje može dobiti i za instance problema, čija je dimenzija relativno velika. To se može dobiti korišćenjem određenih tehnika koje smanjuju dimenziju problema, implicitnim

izbacivanjem "nepotrebnih" i "neperspektivnih" objekata, ili svođenjem na više problema manje dimenzije. Takođe je često moguće višestruko smanjivanje broja grana u drvetu pretrage, korišćenjem pogodnih ograničenja. To su metode: grananja i ograničavanja (Branch-and-Bound), grananja i sečenja (Branch-and-Cut), dinamičkog programiranja, itd. Njihov uspeh ili neuspeh direktno zavisi od postojanja pogodne strukture problema, koju dalje te metode intenzivno koriste.

1.1.3 Heuristike

I pored primene gorepomenutih metoda, često nije moguće optimalno rešiti neki NP-kompletan problem velike dimenzije u dostižnom vremenu izvršavanja. Zbog toga se pojavila potreba za metodama koje, u opštem slučaju, daju samo suboptimano odnosno, približno rešenje, ali je vreme izvršavanja kratko (polinomijane složenosti). Takve metode nazivamo *heuristikama*.
U početku takve metode nisu bile toliko popularne jer su:

- davale rešenje relativno slabijeg kvaliteta (sa relativno velikim odstupanjem od optimalnog rešenja);
- specijalno su konstruisane samo za dati problem, ili nekoliko sličnih problema;
- često su se zaustavljale u prvom lokalnom ekstremu, bez mogućnosti da iz njega izađu, i dostignu globalno opimalno rešenje.

Međutim, u poslednjih 10-20 godina se beleži spektakularan pomak u ovoj oblasti, tako da se heuristike sve šire primenjuju u praksi. Razlozi tako uspešne primene se mogu ukratko opisati na sledeći način:

- Kvalitet dobijenih rešenja je uglavnom zadovoljavajući, pa se često može dobiti čak i optimalno rešenje, iako ovakve metode, najčešće same ne mogu verifikovati da je dobijeno rešenje zaista optimalno;
- Poseduju veliku mogućnost prevazilaženja lokalnog ekstrema, i dostizanja globalnog optimalnog rešenja, čak i za probleme koji imaju veliki broj lokalnih ekstrema;
- Pored vrlo velikog broja heuristika vezanih za konkretan problem, pojavile su se i opšte heuristike, koje se mogu primeniti na široku klasu problema: genetski algoritmi (genetic algorithms), simulirano kaljenje (simulated annealing), tabu pretraživanje (tabu search), Lagrange-ova relaksacija (Lagrangean relaxation), itd;
- U velikom broju slučajeva je moguća hibridizacija nekoliko ovakvih heuristika, i korišćenje dobrih strana svake od njih;
- Heuristike se često uspešno mogu uklopiti u metode za tačno rešavanje problema, tako da se uspešno kombinuje brzina dobijanja rešenja sa verifikacijom optimalnosti.

Genetski algoritmi su nastali 70-tih godina uglavnom inspirisani Holland-ovim radom [Hll75], i imitiraju procese prirodne evolucije. Detaljnije će o njima biti reči u narednom odeljku.

Iako su prve ideje nastale još 50-tih godina u okviru poznatog Metropolis algoritma ([Met53]), simulirano kaljenje se u današnjem obliku zvanično pojavilo početkom 80-ih godina (jedan od prvih radova je [Kir83]), a zatim je doživelo naglu ekspanziju. Data metoda koristi znanja iz termodinamičkih procesa, i emulira ih u vidu algoritama za rešavanje problema kombinatorne optimizacije.

Detaljnije informacije se mogu videti u [Art97b], a opisi nekih primena na praktične probleme su dati u radovima [Lam88] i [Alv92].

Metod tabu pretraživanja je prvi uveo Glover 1986. u radu [Glo86], da bi ubrzo zatim postao standardno sredstvo za rešavanje NP-kompletnih problema, tako da su danas poznati brojni problemi, koji su njime uspešno rešeni. Bliži opis metode se može naći u [Glo90] i [Her97], a neke od primena u [Kno89].

Lagrange-ova relaksacija dodeljuje odgovarajuće faktore (Lagrange-ovi množioci) nekim od uslova zadatka, uključujući ih pri tome u vrednosnu funkciju ([Geo74]). Optimalno rešenje novodobijenog problema celobrojnog ili mešovitog programiranja predstavlja donju granicu za ocenu rešenja polaznog problema. Dobrim izborom uslova koje uključujemo u vrednosnu funkciju, može se dobiti jednostavniji novodobijeni problem, sa mnogo kraćim vremenom optimalnog rešavanja, a da im rešenja budu bliska. Detaljnije informacije o ovoj metodi se mogu naći u radu [BeJ95], a neke uspešne primene pri rešavanju NP-kompletnih problema u radovima: [BeJ88], [Gui88], [BeJ90b], [BeJ93] i [Glv93].

Detaljnije informacije o ovim tehnikama mogu se naći u [Zan89], [Ree95], [Osm96a], [Art97a] i [Vss99] uz opsežnu i klasifikovanu literaturu, uz napomenu da se neke od njih mogu i kombinovati ([Kid93]). Korisne informacije o temi heuristika se mogu naći i u domaćoj literaturi [Cve96] i [Uro96]. Od ostalih metoda pomenimo poliedralne tehnike (polyhedral techniques) koje se vrlo često koriste za rešavanje problema kombinatrone optimizacije ([Ard95] i [Ard96b]).

Pored gorepomenutih stalno se razvijaju nove heuristike od kojih većina vrlo brzo nalazi široku primenu. Neke od najznačajnijih takvih metoda su: neuralne mreže ([Fan90]), mravlji sistemi (ant systems) opisani u [Dor96], epsilon transformacija ([Zha96], [Pem96] i [Kra96b]). Posebno napomenimo metodu promenljivih okolina (Variable Neighborhood Search - VNS) koja je nastala vrlo skoro (1996.), a već je uspešno primenjena na veliki broj problema (videti [Han99]).

Pri rešavanju problema kombinatorne optimizacije takođe treba imati u vidu i činjenicu da ne postoji metoda za njihovo rešavanje koja je univerzalna i koja daje najbolje rezultate na svim problemima ([Wol95]).

1.2 Genetski algoritmi

Genetski algoritmi (GA) su zasnovani na ideji Darvinove teorije o postanku vrsta i prirodnoj evoluciji [Dar85], koja je nastala krajem 19. veka. Iako su prvi radovi koji se generalno mogu klasifikovati u ovu oblast nastali još 60-tih godina, kao idejni tvorac se zvanično uzima John Holland sa knjigom "Adaptation in natural and artificial systems" [Hll75]. Iako su tokom sledeće dve decenije postignuti zavidni rezultati na teorijskom i praktičnom planu, osnovne postavke GA date u tom radu, i danas važe.

Postoji veliki broj preglednih radova o genetskim algoritmima. Spomenimo samo neke od njih: [DJo75], [Bok87], [Gol89], [Dav91], [BeD93a], [BeD93b], [Yur94], [Mic96], [Mit96] i [Müh97]. Opšte informacije o GA se mogu naći i u domaćoj literaturi: [Čan96], [Fil97], [Kra97a], [Toš97] i [Fil98].

Evolucione strategije takođe pripadaju metodama za rešavanje problema optimizacije čije su ideje preuzete iz prirodne evolucije. One koriste mutaciju kao mehanizam pretrage i selekciju za usmeravanje prema perspektivnim regionima pretraživačkog prostora. Za razliku od GA, one ne sadrže operator ukrštanja, već je mutacija jedini mehanizam pretrage. U radovima [Bëc91a] i

[Shw95] se mogu naći detaljnije informacije o teorijskim i praktičnim aspektima, a u [Hof91] je data jedna od implementacija (Escapade).

1.2.1 Opis GA

Genetski algoritmi su robusne i adaptivne metode koje se mogu koristiti za rešavanje problema kombinatorne optimizacije. Osnovna konstrukcija je *populacija* jedinki, kojih je najčešće između 10 i 200. Svaka jedinka predstavlja moguće rešenje u pretraživačkom prostoru za dati probem (prostoru svih rešenja). Svaka jedinka je predstavljena genetskim kodom nad određenom konačnom azbukom. Najčešće se koristi binarno kodiranje, gde se genetski kod sastoji od niza bitova. U nekim slučajevima je pogodno koristiti i azbuke veće kardinalnosti, ali su mišljenja o njihovoj teorijskoj i praktičnoj efikasnosti podeljena ([Ant89] i [Gol89]).

Svakoj jedinki se dodeljuje funkcija prilagođenosti (fitness function) koja ocenjuje kvalitet date jedinke, predstavljene kao pojedinačno rešenje u pretraživačkom prostoru. GA mora da obezbedi način da stalno, iz generacije u generaciju, poboljšava apsolutnu prilagođenost svake jedinke u populaciji, kao i srednju prilagođenost cele populacije. To se postiže uzastopnom primenom genetskih operatora *selekcije*, *ukrštanja* i *mutacije*, čime se dobijaju sve bolja rešenja datog konkretnog problema.

Mehanizam selekcije favorizuje natprosečno prilagođene jedinke i njihove natprosečno prilagođene delove (gene), koji dobijaju veću šansu za sopstvenom reprodukcijom pri formiranju nove generacije. Slabije prilagođene jedinke i geni imaju smanjene šanse za reprodukciju pa postepeno izumiru.

Operator ukrštanja vrši rekombinaciju gena jedinki i time doprinosi raznovrsnosti genetskog materijala. Rezultat ukrštanja je strukturna, iako nedeterministička, razmena genetskog materijala između jedinki, sa mogućnošću da dobro prilagođene jedinke generišu još bolje jedinke. Ovim mehanizmom, i relativno slabije prilagođene jedinke, sa nekim dobro prilagođenim genima, dobijaju svoju šansu da rekombinacijom dobrih gena proizvedu dobro prilagođene jedinke.

Višestrukom primenom selekcije i ukrštanja moguće je gubljenje genetskog materijala, odnosno postaju nedostupni neki regioni pretraživačkog prostora. Mutacija vrši slučajnu promenu određenog gena, sa datom malom verovatnoćom p_{mut}, čime je moguće vraćanje izgubljenog genetskog materijala u populaciju. To je osnovni mehanizam za sprečavanje preuranjene konvergencije GA u lokalnom ekstremu.

Početna populacija se često generiše na slučajan način, što doprinosti raznovrsnosti genetskog materijala. U nekim slučajevima se povoljnije pokazalo generisanje cele početne populacije ili dela populacije nekom drugom pogodno izabranom heuristikom. Jedini preduslov je da vreme izvršavanja date heuristike bude relativno kratko.

Na slici 1.1 je dat shematski zapis osnovnih elemenata GA:

```
Učitavanje_Ulaznih_Podataka();
Inicijalizacija_Populacije();
while (! Kriterijum_Završetka_GA() )
 {
     for (i=0; i< Npop; i++) pi = Vrednosna_Funkcija();
     Funkcija_Prilagođenosti();
     Selekcija();
     Ukrštanje();
     Mutacija();
 }
Štampanje_Izlaznih_Podataka();
```

Slika 1.1 Shematski zapis GA

1.2.2 Prost GA i njegovi nedostaci

Najprostija varijanta genetskog algoritma je opisana u Holland-ovoj knjizi [Hll75] i sastoji se od proste rulet selekcije, jednopozicionog ukrštanja i proste mutacije i ona se u literaturi obično naziva prost genetski algoritam (simple genetic algorithm - SGA). Detaljan opis datih genetskih operatora sa primerima se može naći u 3. poglavlju.

U teorijskim rezultatima prost GA najbolje odgovara hipotezi o gradivnim blokovima i teoremi o shemi, ali ima neke nedostatke u praktičnoj primeni. U teorijskim razmatranjima se veličina populacije ne uzima u obzir, jer se smatra implicitno da populacija ima beskonačno mnogo članova. Tada se lako primenjuju matematičke tehnike iz teorije verovatnoće. Međutim, u realnim uslovima populacija ipak sadrži konačno mnogo jedinki pa primenom genetskih operatora selekcije, ukrštanja i mutacije nastaje greška uzorkovanja (sampling error), koja može značajno uticati na izvršavanje GA. Iako je, u proseku, na velikom obimu uzorka, broj potomaka svake jedinke približno jednak očekivanom broju, pojedinačno je moguće prilično veliko odstupanje. Ukoliko takvo odstupanje nastupi u nekoliko početnih generacija, ono može da utiče presudno na kasnije performanse GA: kvalitet dobijenog rešenja i brzinu konvergencije. U tom slučaju genetski operatori gube svoju prvobitnu funkciju i postaju nedelotvorni ili destruktivni. Dati problemi se najčešće manifestuju kroz gubitak genetskog materijala, preuranjenu konvergenciju i sporu konvergenciju GA.

1.2.2.1 Preuranjena konvergencija i gubitak genetskog materijala

Najčešći problem u primeni prostog genetskog algotima je tzv. *preuranjena konvergencija*. Data pojava se dešava ukoliko jedna ili nekoliko relativno dobrih jedinki, ali ne i optimalnih, postepeno preovlada u populaciji i proces konvergira u lokalnom ekstremu. Pri tome, u nastavku izvršavanja, mogućnosti genetskog algoritma za poboljšanje datog rešenja su jako male. Osnovni razlozi za to su:

- Selekcija i ukrštanje u populaciji sa istim jedinkama nema nikakvog efekta.
- Jedino mutacija, teorijski, može da doprinese izlazu iz date situacije, međutim zbog potpuno uništenog genetskog materijala, u praksi je često bez efekata. Ukoliko je nivo mutacije relativno mali, promene genetskog

materijala su neznatne, i dominantne jedinke ponovo, vrlo brzo, eliminišu sve ostale jedinke iz populacije. Ukoliko je nivo mutacije relativno veliki, GA se pretvara u pretragu na slučajan način.

Preuranjena konvergencija najčešće nastaje kao posledica primene proste rulet selekcije. Ukoliko u populaciji postoji jedinka sa relativno velikom funkcijom prilagođenosti, ona će najverovatnije istisnuti sve ostale jedinke iz populacije. Pri tome ostale jedinke, iako sa malom funkcijom prilagođenosti, sadrže raznovrsne gene od kojih su neki bolji u poređenju sa genima favorizovane jedinke. Međutim, zbog lošeg rasporeda dobrih gena po lošim jedinkama i konačne veličine populacije, u procesu proste selekcije, dolazi do eliminacije jedinki (a samim tim i dobrih gena) iz populacije. Pri tome dobri geni gube šansu, da se operatorima ukrštanja i selekcije rekombinuju u dobre jedinke i time poboljšaju dobijeno rešenje.

1.2.2.2 Spora konvergencija

Ovaj problem je suprotan preuranjenoj konvergenciji i dešava se po pravilu u kasnijoj fazi izvršavanja prostog genetskog algoritma. Ukoliko je populacija postala dovoljno slična, ali nije dostignuto optimalno rešenje, srednja prilagođenost svih jedinki u populaciji je velika, a razlike između najbolje jedinke i ostalih jedinki u populaciji su male. Zbog toga postoji nedovoljni gradijent u funkciji prilagođenosti koji bi pomogao genetskom algoritmu da dostigne optimalnu vrednost.

1.2.3 Napredne tehnike GA

Za prevazilaženje gorepomenutih anomalija prostog GA i za uspešno izvršavanje genetskog algoritma koriste se razne tehnike prilagođene prirodi problema koji rešavamo. To su: razne vrste kodiranja i odgovarajućih vrednosnih funkcija, prilagođeni složeniji genetski operatori, više vrsta funkcija prilagođenosti, politika zamene generacija, fiksna ili adaptivna promena parametara tokom izvršavanja GA.

1.2.3.1 Kodiranje i vrednosna funkcija

Najznačajniji faktori za uspešnu primenu GA u rešavanju određenog problema su kodiranje i vrednosna funkcija. Međutim, oni direktno zavise od prirode problema koji rešavamo pa neke metode koje su uspešne u nekim primenama, mogu dati vrlo loše rezultate u primeni na drugoj vrsti problema.

Idealno je da za dato kodiranje, funkcija prilagođenosti bude neprekidna i glatka, pri čemu jedinke sa sličnim genetskim kodom imaju sličnu vrednost (prilagođenost). Dobre performanse, takođe, obezbeđujemo ako funkcija prilagođenosti nema suviše mnogo lokalnih ekstrema ili suviše izolovan globalni ekstrem ([BeD93a]). Međutim, u tim povoljnim slučajevima i druge metode često daju približne ili čak i bolje rezultate. Zbog toga se GA primenjuju i u nepovoljnim slučajevima, kada druge metode ili nije moguće primeniti, ili daju izuzetno loše rezultate, a genetski algoritmi ipak uspevaju da nađu nekakvo rešenje.

Generalno je pravilo da vrednosna funkcija treba da preslikava genetski kod jedinke u prostor pretrage na neki "realan" način. Često se, međutim, dešava da takav način ne postoji ili zbog nekih razloga nije pogodan (na primer zbog velikog učešća nekorektnih jedinki).

Genetski algoritam je generalno najpogodniji za probleme koji se mogu binarno kodirati na prirodan način. Mogući su i slučajevi korišćenja alfabeta veće kardinalnosti od binarne. Goldberg u [Gol89] dokazuje da teorijski binarna reprezentacija daje veći broj shema, čija je posledica veći stepen implicitnog paralelizma. Međutim, Antonisse u radu [Ant89], drugačije interpretira pojam sheme i potpuno suprotno zaključuje da GA nad alfabetima veće kardinalnosti sadrže više shema od binarnih.

U primerima GA nad alfabetima veće kardinalnosti, simboli (geni) su najčešće predstavljeni celim ili realnim brojevima. U [Dav91] se zaključuje da su vrednosti u programu često numeričkog tipa, i da njihova direktna reprezentacija kao brojeva, umesto kao niza bitova, poseduje niz prednosti. Na primer, mnogo je lakše i prirodnije definisanje operatora ukrštanja i mutacije zavisnih od prirode problema. O nebinarnim reprezentacijama detaljnije se može videti u [Ant89], [BeD93b] i [Gol89], a jedna uporedna analiza performansi binarnih i reprezentacija pomoću realnih brojeva je data u [Jan91].

1.2.3.2 Postupak u slučaju nekoreknih jedinki

U praktičnim primenama je najpogodnije ako se pri kodiranju bijektivno preslikavaju parametri problema u genetske kodove. Međutim, za neke probleme je nemoguće naći takvo kodiranje, ili ono daje rezultate koji nisu zadovoljavajući. U tom slučaju smo prinuđeni na kodiranje koje nije bijekcija, ili čak nije preslikavanje. Mogući su sledeći slučajevi:

- Za određenu jedinku ne postoji genetski kod. U tom slučaju kodiranje moramo odbaciti, jer ne postoji nikakav način za prevazilaženje datog nedostatka;
- Za neke jedinke postoji više genetskih kodova. Iako to sa teorijskog stanovišta nije pogodno, ne postoji nikakva praktična prepreka za primenu datog kodiranja;
- Određeni genetski kod predstavlja više jedinki. Kodiranje moramo odbaciti, kao i u prvom slučaju;
- Određeni genetski kod ne odgovara nijednoj jedinki. Takav genetski kod nazivamo *nekorektnim*. Najčešće dati genetski kod ima interpretaciju u prostoru gde se problem rešava, ali ne pripada prostoru rešenja. Odnosno, nije ispunjen neki od uslova, koji definišu rešenje problema.

Pošto su prvi i treći slučaj teorijski neprihvatljivi, a drugi možemo ignorisati, zanimljiv za rešavanje je jedino poslednji slučaj. Postoji nekoliko načina za rešavanje problema sa nekorektnim genetskim kodovima.

Nekorektne genetske kodove možemo da ignorišemo, i dodelimo im nulu za vrednost funkcije prilagođenosti. U sledećoj generaciji će oni biti automatski izbačeni iz populacije. Na taj način u daljem postupku ostaju samo korektni genetski kodovi, a nekorektni ostaju najduže u tekućoj generaciji, a zatim nestaju. Iako teorijski dobar, dati postupak se u praksi može primeniti samo na slučajeve, gde u prostoru pretrage nekorektnih jedinki (kodova) ima najviše 50-70%. U praksi takvih problema ima zanemarljivo malo, jer su obično uslovi koji definišu rešenje problema jako restriktivni. Pri tome je korektnih jedinki u prostoru pretrage najčešće nekoliko redova veličine manje od nekorektnih. U tom slučaju, genetski algoritam uopšte ne može da se izvršava, jer na početku ili u toku izvršavanja, u datoj generaciji, populacija sadrži samo nekorektne jedinke.

Moguće je i uključivanje nekorektnih kodova u genetski algoritam. Pri tome za svaku jedinku u populaciji (korektnu ili nekorektnu) računamo, kao i u klasičnom genetskom algoritmu, vrednost date jedinke. Svaku nekorektnu jedinku "kažnjavamo" računajući kaznenu vrednosnu funkciju (penalty value function). Na osnovu datih vrednosti, računamo prilagođenost svake jedinke u populaciji, i dalje nastavljamo klasični GA. Pri definisanju kaznene funkcije je potreban kompromis, jer preblaga kaznena funkcija može dovesti do toga da konačno rešenje ne ispunjava uslove zadatka. Nasuprot tome, preoštra kaznena funkcija može dovesti do rešenja čija je vrednost daleko od optimalne. Jedno od iskustvenih pravila kaže da kaznena funkcija mora biti nešto strožija od cene pretvaranja nekorektne u korektnu jedinku (videti [Ric89]). O različitim aspektima kaznenih funkcija se takođe može videti u [Lvi93a] i [SmA93].

Nekorektne jedinke "popravkom" možemo da transformišemo u korektne. Pri tome za vrednost nekorektne jedinke preuzimamo vrednost korektne "popravke". Tada je moguće da popravljenu jedinku vratimo u populaciju na mesto stare nekorektne (videti [Nak91]), ili ne (videti [Orv93]). Podeljena su mišljenja oko toga koji od ova dva načina daje bolje rezultate.

1.2.3.3 Selekcija

Selekcija zasnovana na rangu, turnirska selekcija i fino gradirana turnirska selekcija su postigle najbolje rezultate u praksi na prevazilaženju anomalija proste, rulet selekcije. Izvršavanjem GA na problemima koji su rešavani u ovom radu (SPLP, UNDP i ISP), korišćenjem ovih operatora selekcije višestruko su poboljšane njegove performanse: kvalitet dobijenog rešenja, vreme izvršavanja, broj generacija. Detaljnije informacije o datim operatorima selekcije, kao i o rezultatima izvršavanja GA na datim problemima se može videti u nekoliko narednih poglavlja. Uporedna analiza performansi nekoliko operatora selekcije se može naći i u radu [Gol91].

1.2.3.4 Politika zamene generacija

Prost GA predviđa da se u svakoj generaciji promeni čitava populacija novim jedinkama (potomcima), što se u literaturi često naziva i *generacijski GA* (generational GA). Iako teorema o gradivnim blokovima teorijski dokazuje da će kvalitetni geni i dobre jedinke sa velikom verovatnoćom proći u narednu generaciju, to ne mora uvek i da se ostvari. Tako se dešava da neki potomci mogu biti i lošiji od roditelja, pa čak i da najbolja jedinka ne prođe u narednu generaciju. To je posebno loše ako u nekom trenutku postignemo rešenje dobrog kvaliteta, a da ga zatim izgubimo probabilističkom primenom genetskih operatora selekcije, ukrštanja ili mutacije.

Jedno od rešenja je u tom slučaju smanjivanje selekcionog pritiska uvođenjem *stacionarnog GA* (steady-state GA), eventualno u kombinaciji sa elitističkom strategijom (elitist strategy) zamene generacija. U stacionarnom GA samo određeni deo populacije prolazi selekciju, a ostatak populacije direktno prolazi u narednu generaciju. Tada se u svakoj generaciji novim jedinkama (potomcima) ne zamenjuje celokupna populacija, već samo jedan njen deo. U nekim radovima se primenjuju elitističke strategije, gde jedna ili nekoliko najboljih jedinki može preći direktno u narednu generaciju, bez primene genetskih operatora selekcije, ukrštanja i mutacije.

Iako se u tim slučajevima smanjuje moć pretrage GA, dobre jedinke sa sigurnošću prolaze u narednu generaciju. Time se ukida mogućnost gubljenja

dobijenih dobrih jedinki "nesretnom" primenom nekog genetskog operatora (selekcije, ukrštanja ili mutacije). Rezultat toga su obično slabije performanse GA u nekoliko početnih generacija, ali su konačni rezultati stacionarnog GA uz elitne strategije u najvećem broju primena bolji u odnosu na rezultate generacijskog GA. Detaljnije oko ovog aspekta GA se može naći u radovima [Sys91] i [SmR93].

1.2.3.5 Funkcija prilagođenosti

Način računanja funkcije prilagođenosti takođe može da utiče, u velikoj meri, na performanse GA. Osim običnog (linearnog) skaliranja, direktnog ili inverznog skaliranja u neki interval (najčešće u interval [0,1]), razvijena je i posebna tehnika sigma odsecanja (sigma truncation scheme). One su detaljno opisane u odeljku 2.3.1, a takođe i u radovima [Sri94] i [BeD93a]. Primenjuju se još i tehnike rangiranja prilagođenosti jedinki u populaciji (fitness ranking) i implicitna funkcija prilagođenosti (implicit fitness remapping).

Jedan od pokušaja da se obezbedi bolja funkcija prilagođenosti je korišćenje tzv. Gray kodova, čija je osobina da se susedni genetski kodovi razlikuju samo u jednom bitu. Time se u nekim pogodnim slučajevima mogu poboljšati performanse genetskih algoritama.

1.2.3.6 Ukrštanje i mutacija

Izbor operatora ukrštanja takođe utiče na performanse GA, ali u nešto manjoj meri od prethodnih tehnika. Generalno, izbor operatora ukrštanja zavisi i od međuzavisnosti gena u genetskom kodu. U problemima sa velikom međuzavisnošću bitova, najpogodnije je jednopoziciono ukrštanje, jer se pomoću njega najmanje razbija celina jedinke. Za probleme sa manjom međuzavisnošću bitova je najpogodnije dvopoziciono ili višepoziciono ukrštanje. U problemima gde su bitovi potpuno nezavisni, ili je zavisnost vrlo mala, najpogodnije je uniformno ukrštanje, iako teorijski za njega ne važi hipoteza o gradivnim blokovima. Detaljnije o raznim vrstama ukrštanja se može videti u 2. poglavlju kao i u radovima [BeD93b] i [Sri94].

Generalno se u slučaju mutacije, za razliku od ukrštanja, uglavnom koristi samo prosta mutacija. Ona može biti realizovana na neki brži način (na primer korišćenjem normalne raspodele). Izuzetak su jedino operatori mutacije zavisni od prirode problema.

Postoji priličan broj genetskih operatora (ukrštanja i mutacije) zavisnih od prirode problema. To se dešava najčešće u slučajevima kada se jedinke kodirane na poseban način, pa postoji određena struktura koja mora biti očuvana primenom genetskih operatora. Drugi način korišćenja specifičnih genetskih operatora, je pri rešavanju problema sa mnogo uslova (constrained problems), gde postoje nekorektni genetski kodovi. Ukrštanje i mutacija se konstruišu tako da čuvaju korektnost genetskih kodova jedinki na koje se primenjuju. Pri tome se teži očuvanju osobina kanonskog GA i za operatore zavisne od prirode problema.

Jedan od prvih primera ovakvih operatora je dao Goldberg, razvijajući PMX (partially matched crossover) za korišćenje u problemima koji zavise isključivo od poretka, a ne i od vrednosti (order-based problems). Operator je konkretno primenjen na problemu trgovačkog putnika, i može se videti u [Gol85]. Posle toga se pojavilo više takvih genetskih operatora koji se mogu primeniti samo na

jedan ili nekoliko sličnih problema, od kojih ćemo pomenuti samo neke: [Gre85], [Mos89], [Uck93],

1.2.3.7 Izbor parametara GA

Iako je izbor parametara GA, kao što su nivo ukrštanja, nivo mutacije, broj jedinki u populaciji, i sličnih, vrlo važan za primenu genetskog algoritma, ipak se pokazalo da nije presudan. Mnogi pokušaji poboljšanja performansi genetskih algoritama zasnovani na određivanju optimalnih parametara GA su dali relativno skromne rezultate. Grefenstette u radu [Gre86] proučava parametre GA i daje konkretna uputstva za određivanje njihove vrednosti. Međutim, u istom radu zaključuje da je mehanizam GA toliko robustan i fleksibilan, da daje dobre rezultate u širokom opsegu vrednosti parametara. Zaključak je da su kritični momenti kodiranje i funkcija prilagođenosti, a da ostali parametri manje utiču na performanse GA. Ukoliko određeni genetski algoritam daje loše rezultate, on će biti isto toliko loš, i pri proizvoljnom izboru parametara GA. I pored toga dobrim izborom parametara GA mogu se propraviti performanse. Tada genetski algoritam koji je davao solidne rezultate biće još bolji.

Pri izvršavanju GA je primećeno da fiksni izbor parametara, a naročito nivo mutacije, nije uvek najpogodniji ([Bëc93]). Raznovrsnost genetskog materijala nije uvek jednaka u svim fazama izvršavanja genetskog algoritma, pa su česti slučajevi da se optimalne vrednosti nivoa ukrštanja, odnosno mutacije, menjaju tokom izvršavanja GA. Načini promene parametara GA u toku izvršavanja su grupisani u dve kategorije:

- Fiksna promena parametara, pri čemu se unapred zadaje smer promene (povećavanje ili smanjivanje vrednosti tokom generacija), i formula po kojoj se promena vrši. U nekim slučajevima je postepeno povećavanje nivoa mutacije i istovremeno smanjivanje nivoa ukrštanja davalo dobre rezultate [Sys89]. Nasuprot tome, u drugim okolnostima se efikasnijim pokazalo eksponencijalno smanjivanje nivoa mutacije [Ack87], [Fog89] i [Brml91].
- Adaptivno određivanje parametara, na osnovu toga kako se dati operator ponašao i kakve je, do tada, rezultate postigao. Operator koji će biti primenjen bira se na slučajan način na osnovu svoje težine, odnosno, dotadašnje uspešnosti. Za detaljnije informacije o adaptivnom određivanju parametara videti [Dav89], [Dav91], [Bëc92a], [BeD93b] i [Sri94].

1.2.3.8 Obmanjivački problemi

Uspešna primena GA se zasniva na činjenici da se korišćenjem selekcije tokom generacija u populaciji stalno povećava učestanost natprosečnih shema (koje učestvuju u dobrim jedinkama populacije) u odnosu na ostale sheme. Ovo se posebno odnosi na sheme malog reda (low-order shemata), poznate i pod imenom gradivni blokovi (building blocks). Primenom operatora ukrštanja rekombinacijom natprosečnih shema (gradivnih blokova) se u sledećoj generaciji formiraju jedinke koje su još bolje od onih u tekućoj generaciji. Teorijski, u najvećem broju slučajeva, ovaj proces u konačnom broju generacija konvergira optimalnom rešenju.

Međutim, u nekim slučajevima sheme, koje treba da generišu optimalno rešenje, trenutno pripadaju lošim jedinkama u populaciji, pa bivaju istisnute. Na taj način se povećava učestanost shema koje ne pripadaju optimalnom rešenju, na račun onih koje pripadaju, čime se GA u pretrazi udaljava od optimalnog rešenja, umesto da mu se približava. Ova pojava je poznata kao ”obmanjivanje”

(deception) i dešava se kao specijalan slučaj epistaze, kada promena nekog gena u genetskom kodu potpuno različito utiče na promenu prilagođenosti (vrednosti) jedinke, u zavisnosti od ostalih gena. Međutim, epistaza je samo potreban, ali ne i dovoljan uslov za pojavu ”obmanjivanja”, a nihov međusobni odnos je detaljno opisan u [Gol87] i [Deb91].

Problem nazivamo ”obmanjivačkim” (deceptive problem), ako je srednja prilagođenost shema u populaciji, koje ne pripadaju optimalnom rešenju, veći od onih koje pripadaju. Problem je ”potpuno obmanjivački” (fully deceptive problem), ako su sve sheme malog reda, sadržane u nekom suboptimalnom rešenju, bolje od svih ostalih shema.

U radovima [Vse91] i [Bat93] su definisani izomorfizmi GA, a pomoću njih je teorijski pokazano da se svaki ”obmanjivački” problem može preslikati u ”neobmanjivački”. Međutim, ovo tvrđenje ima samo teorijski značaj, pošto je praktično konstruisanje takvog izomorfizma složenije od polaznog problema. U praksi su se ”obmanjivački” problemi pokazali relativno teškim za rešavanje pomoću GA, a još težim za klasične metode, pošto obično ne ispunjavaju mnoge uslove (neprekidnost, diferencijabilnost, itd). Zbog toga su osim opštih metoda razvijene i specijalne metode za prevazilaženje ove anomalije (videti [Gre93]). Treba imati u vidu da postoje problemi koji se relativno lako rešavaju pomoću GA, ali ne i klasičnim metodama (opširnije videti u [Wls91]).

Napomenimo da se u poslednje vreme pojavio veći broj uglavnom praktičnih problema, koji nisu obmanjivačkog tipa, ali su teški za rešavanje (kako pomoću GA tako i pomoću drugih metoda).

1.2.4 Primena GA u rešavanju problema kombinatorne optimizacije

Jedna od najznačajnijih primena GA je rešavanje NP-kompletnih problema, kao i ostalih problema kombinatorne optimizacije. Pregled svih relevantnih informacija iz ove oblasti daleko prevazilazi obim ovog rada pa ćemo prikazati samo nekoliko značajnih opštih radova i primenu GA na neke od poznatijih NP-kompletnih problema. {XE"genetski algoritmi:primena"}

De Jong i Spears u radu [DJo89] navode da iako su svi NP-kompletni problemi teorijski svodivi algoritmom polinomske složenosti, neki od njih se mogu mnogo bolje kodirati pomoću GA od ostalih. Zbog toga su rezultati genetskog algoritma veoma različiti ukoliko ga primenjujemo na razne NP-kompletne probleme. Dalje je u tom radu prikazana primena GA za rešavanje problema logičke zadovoljivosti (SAT - Boolean satisfiability problem), koji je uspešno rešen. Analiza opštih karakteristika genetskih algoritama kao metoda za rešavanje problema kombinatorne optimizacije je data i u radu [Dow96].

U radu [Khu94] su dati rezultati primene GA na nekoliko specifičnih NP-kompletnih problema: maksimalno presecanje (maximum cut problem), nalaženje plana sa minimalnim zakašnjenima (minimum tardy task problem) i zbir podskupova (subset sum problem). Za poslednja dva rešavana problema, zbog velikog broja ograničenja (constraints), razvijene su posebne vrste kaznenih funkcija primenjenih na funkciju prilagođenosti. Još neki pristupi u primeni GA za rešavanje problema sa velikim brojem ograničenja su dati u radovima [Mic91], [Shn93], [Hmf94] i [Mic94].

Jedan pristup rešavanju problema kombinatorne optimizacije pomoću genetskih algoritama je dat i u radovima [Chu97a], [Chu97b] i [BeJ96b]. Primenjena je hibridizacija stacionarnog GA sa nekim heuristikama za poboljšavanje rešenja koje zavise od prirode problema. Dati pristup je

primenjen za rešavanje nekoliko poznatih problema: pokrivanje skupa (set covering problem), disjunktno pokrivanje skupa (set partitioning problem), uopšteni problem dodele (generalized assignment problem) i problem ranca sa višestrukim ograničenjima (multiconstraint knapsack problem). Vršeni su eksperimenti i sa različitim vrstama kodiranja datih problema i korišćenjem kaznenih funkcija.

Jedan pokušaj korišćenja optimalnih drveta zavisnosti za rešavanje problema kombinatorne optimizacije je prikazan u radu [Blu97]. U datom pristupu su primenjene konstrukcijame koje opisuju pretraživački prostor na pogodan način, da bi na njima mogle biti primenjene metode učenja.

Od pojedinačnih NP-kompletnih problema koji su rešavani pomoću GA najviše primena je zabeleženo u slučaju problema trgovačkog putnika (TSP) . Iako su se prvi radovi pojavili relativno kasno, tek sredinom 80-tih godina ([Gol85] i [Gre85]), kasnije se vrlo aktivno nastavilo u ovom pravcu, sve do danas. Pomenimo samo nekoliko zanimljivih pristupa datih u [Jog89], [Jog90], [Hmf93] i [Dzu94], a opsežniji pregled nekih od metoda se može naći u [Joh97] i [Kra98a].

Osim navedenih pomenimo i primene GA za rešavanje problema pokrivanja skupa ([AlS96]) , disjunktnog pokrivanja skupa ([Lvi93a], [Lvi93b] i [Lvi96]) , lokacijsko-alokacijskog problema ([Hou95]) i Štajnerovog problema ([Lju98]) . Veliki broj radova koji opisuju primenu GA za rešavanje problema kombinatorne optimizacije je klasifikovano, i mogu se naći u bibliografijama [Hei93], [Aln95] i [Osm96b].

Osim za rešavanje NP-kompletnih problema i problema kombinatorne optimizacije, GA i ostale tehnike bazirane na evolucionoj paradigmi se primenjuju i na širokom spektru drugih problema. Neke od najznačajnijih primena su: mašinsko učenje ([Vug96]), neuralne mreže ([Shf92]), optimizacija funkcija ([Fil96a] i [Fil96b]), numerički problemi, adaptivno generisanje programa ([Koz93]), višekriterijumska optimizacija ([Fon93]), dizajn ([Lou91]) kao i rešavanje ostalih praktičnih problema.

Napomenimo da postoje i neki pokušaji uopštenja GA radi poboljšavanja efikasnosti njihove pretrage. Detaljni opisi nekih takvih pristupa se mogu naći u [Lve91], [Müh94] i [Müh98].

1.3 Višeprocesorske arhitekture i paralelni računari

Programiranje višeprocesorskih računara je višestruko složenije od programiranja jednoprocesorskih računara. Mnogo je više faktora koji bitno utiču na ukupno vreme izvršavanja (detaljnije videti u [Ber89] i [Lew92]). I analiza efikasnosti algoritama i vremena izvršavanja programa na višeprocesorskim računarima je mnogo složenija u odnosu na sekvencijalne algoritme i programe.

U programiranju višeprocesorskih računara cilj je minimizacija vremena izvršavanja. Preduslov za to je maksimalna iskorišćenost procesora (da pojedini procesori čekaju što je moguće manje). Te vrednosti zavise direktno od prirode problema, i to od toga da li se i kako problem može razložiti na potprobleme, koji ne zavise jedni od drugih. Tada se svi potproblemi mogu istovremeno izvršavati na različitim procesorima.

1.3.1 Modeli računara

Postoji više podela računarskih sistema. Jedna od najpoznatijih je Flynn-ova podela (videti [Fly66] i [Fly72]), pri čemu se računari po arhitekturi dele na:

- SISD (Single Instruction Single Data) u kojima postoji samo jedan procesor koji prima jedan tok instrukcija i operiše nad jednim tokom podataka. Najveći broj današnjih računara spada u ovaj model koji je projektovao John von Neumann sa saradnicima krajem 40-tih. Algoritmi za ovaj model su sekvencijalni, jer postoji samo jedan procesor.
- MISD (Multiple Instruction Single Data) gde postoji N procesora, pri čemu svaki poseduje po jednu kontrolnu jedinicu, ali svi dele isti tok podataka. U svakom koraku, svi procesori obrađuju istovremeno jedan podatak prihvaćen iz zajedničke memorije. Svaki procesor obrađuje podatak prema instrukciji koju prima sa sopstvene upravljačke jedinice. Ovaj model je pogodan ukoliko nad istim podatkom treba izvršiti više operacija u isto vreme.
- SIMD (Single Instruction Multiple Data) koji karakteriše N identičnih procesora. Svaki procesor poseduje sopstvenu lokalnu memoriju. Svi procesori rade pod kontrolom jedne kontrolne jedinice, ili svaki procesor ima svoju kopiju istog programa u lokalnoj memoriji. Izvršavanje je sinhrono, odnosno, svaki procesor u isto vreme izvršava istu instrukciju, i ima svoj tok podataka.
- MIMD (Multiple Instruction Multiple Data) je najsloženiji model, u kome svaki procesor poseduje kontrolnu jedinicu, aritmetičko logičku jedinicu i lokalnu memoriju. Svaki procesor ima svoj tok instrukcija i podataka, tako da procesori mogu izvršavati različite programe nad različitim podacima, što znači da tipično rade asinhrono. Iako je najmoćniji model, sa višeprocesorskim računarima ovog modela je najteže raditi, jer postoje problemi koji se ne javljaju kod ostalih modela: dodela procesa procesoru, čekanje procesa na slobodan procesor i razmena podataka između više procesora (procesa).

Druga podela paralelnih računarskih sistema je po načinu komunikacije između procesora:

- Višeprocesorski računari sa komunikacijom preko deljene memorije (shared memory multiprocessors).
- Višeprocesorski računari sa komunikacijom prosleđivanjem poruka (message passing multiprocessors).

1.3.1.1 Komunikacija preko deljene memorije

U ovom modelu je celokupni memorijski prostor zajednički i dostupan za sve procesore. Ukoliko procesori žele da razmene neke podatke, to se vrši prostim upisom podatka od strane jednog i čitanja podatka od strane drugog procesora.

Međutim, ovaj paralelni model, iako teorijski moćniji i fleksibilniji, nije u praksi dao rezultate koji se u teoriji predvidjaju. Razlozi su u težoj implementaciji dela za paralelno čitanje odnosno upis podataka i u njegovom sporijem izvršavanju, ukoliko je broj procesora veliki (videti [Akl89]). Programiranje na takvim sistemima je olakšano, jer operativni sistem sam vodi računa o paralelnom čitanju i upisu.

Jedna od najpoznatijih biblioteka, koja je i IEEE standard, za programiranje paralelnih računara sa međuprocesorskom komunikacijom preko deljene memorije je Pthread detaljno opisan u [Nic96].

1.3.1.2 Komunikacija prosleđivanjem poruka

U poslednjoj deceniji su naročito postali popularni paralelni računari gde se međuprocesorska komunikacija ostvaruje prosleđivanjem poruka (message passing). Oni su lakši za projektovanje i implementaciju, ali je na njima teže programirati, jer sam programer mora da vodi računa o sinhronizaciji procesa i razmeni podataka izmedju njih, za razliku od sistema sa zajedničkom memorijom gde o tome vodi računa operativni sistem.

Međuprocesorska komunikacija, kod sistema sa prosledjivanjem poruka, takodje, bitno utiče na vreme izvršavanja. Vreme potrebno za prenos odredjenog podatka, izmedju različitih procesora, je ponekad, nekoliko puta veće od vremena potrebnog za aritmetičku operaciju nad njim (videti rad [Kra94]). Povećanje iskorišćenosti procesora zahteva i veću komunikaciju, pa se moraju odrediti takve vrednosti (komunikacije izmedju procesora i iskorišćenosti procesora) koje obezbedjuju minimalno vreme izvršavanja.

U prošlosti je svaki paralelni računar zasnovan na datom modelu posedovao sopstveni programski sistem za razvoj paralelnih aplikacija, gde su različiti sistemi bili nekompatibilni a samim tim i aplikacije neprenosive između raznih sistema. Najpoznatiji su bili sistemi razvijeni od velikih firmi Cray Research, IBM, Intel, Meiko i Ncube, kao podrška njihovim paralelnim računarima visokih performansi. Osim njih, poznati su bili i programski sistemi za transpjutere (opisani u [Moc89] i [Tra89b]).

Kasnije je pokušavano sa standardizacijom ovakvih sistema i razvijeni su sledeći sistemi:

- CHIMP ([CHI91] i [CHI92]);
- p4 ([But94]);
- PVM ([Gei94] i [Sev98]);
- Zipcode ([Skj90] i [Skj92]);
- neki drugi sistemi ([Fry92]).

Međutim, svi ovi sistemi su posedovali određene nedostatke, pa se pristupilo razvoju standarda, koji bi bio podržan na što većem broju platformi, efikasan i sa programskim konstrukcijama relativno visokog nivoa. Tako je nastao MPI standard detaljnije opisan u odeljku 1.3.3 .

1.3.2 Paralelne platforme

Računarske platforme za razvoj, testiranje i izvršavanje paralelnih programa se mogu podeliti u nekoliko globalnih kategorija:

- Simulatori i emulatori višeprocesorskih računara na jednoprocesorskim računarima;
- Multiprocesorski sistemi nižih performansi;
- Mreža radnih stanica povezanih u paralelni računar;
- Superračunari i multiprocesorski sistemi visokih performansi.

1.3.2.1 Simulatori paralelnih računara

Najprostije razvojne platforme uz minimalne troškove za paralelizaciju su simulatori (emulatori) višeprocesorskih računara. Njihova namena može biti:

- Modeliranje nekog realnog višeprocesorskog sistema u cilju bližeg upoznavanja sa datim sistemom i eventualno razvijanje konkretnih aplikacija. U tom slučaju bi sve pripremne radnje u razvoju date aplikacije bile urađene na simulatoru, a samo konačno izvršavanje aplikacije bi bilo izvršeno na konkretnom paralelnom računaru. Jedan od primera simulatora MPI standarda za PC računare pod operativnim sistemom Windows 3.1 se može videti u radu [Mey94].

- Simulatori nekog imaginarnog višeprocesorskog sistema, čija je najvažnija namena što lakše upoznavanje šireg auditorijuma sa paralelnim konstrukcijama. Zbog toga su relativno zanemareni ostali važni segmenti kao što su kompatibilnost sa ostalim sistemima i performanse datog simulatora. Oni su često zamišljeni kao podrška odgovarajućim knjigama ([Les93]), ili kursevima paralelnog programiranja.

Ovakvi sistemi mogu poslužiti za upoznavanje sa osnovnim karakteristikama paralelnih računara, ali zbog niza nedostataka nisu pogodni za širu primenu. Iako je teorijski moguće simulirati sve elemente paralelnog računara, u praksi rezultati u velikoj meri odstupaju od očekivanih. To je posledica činjenice da se simulacija vrši na jednoprocesorskom računaru, gde je nemoguće simulirati neke vrlo važne fizičke karakteristike višeprocesorskog računara: problemi sa paralelnim pristupom memoriji, relativno spora međuprocesorska komunikacija, različite karakteristike pojedinačnih procesora unutar paralelnog računara i neke od ostalih efekata realnog paralelnog računara. Zbog toga je moguće da rezultati dobijeni na simulatoru budu potpuno nereprezentativni, pa čak i u kontradikciji sa rezultatima iste aplikacije na realnom paralelnom računaru. Iako dati pristup ima vrlo velike nedostatke, ranije je relativno često korišćen, zbog relativno visoke cene višeprocesorskih računara u prošlosti.

1.3.2.2 Multiprocesorski sistemi nižih performansi

Prvi šire dostupni i relativno jevtini višeprocesorski sistemi su se pojavili 80-tih godina (transpjuteri, ICUBE, itd). Transpjuterski sistemi su bili najpoznatiji predstavnici ove klase paralelnih računara, zasnovanih na transpjuterima kao osnovnim procesorima, gde je međuprocesorska komunikacija vršena prosleđivanjem poruka. Ploče sa transpjuterima su se ugrađivale u PC računare ili Sun radne stanice, koji su obezbeđivali ulazno/izlazne operacije. Nisu razvijani novi operativni sistemi, već su korišćeni postojeći operativni sistemi domaćinskih računara (host computers). Paralelizacija je ostvarivana paralelnim konstrukcijama u okviru konkretnih programskih jezika, koji su nastajali nadogradnjom postojećih (paralelni C) ili razvojem potpuno novih programskih jezika prilagođenih paralelnom izvršavanju (Occam). Više informacija o transpjuterskim sistemima se može naći u [Moc89] i [Tra89a].

I pored brojnih tehničkih ograničenja, ovakvi višeprocesorski sistemi su u vreme pojavljivanja doprineli širokoj popularizaciji paralelnih računara i upoznavanju šireg kruga ljudi sa paralelnim konstrukcijama na realnom višeprocesorskom računaru. Međutim u poslednje vreme su u većini slučajeva praktično napušteni kao paralelna platforma zbog nedovoljne sistemske podrške, relativno malog broja paralelnih konstrukcija i njihove ograničene izražajnosti, kao i zbog nedostatka efikasnih alata za razvoj i testiranje aplikacija.

1.3.2.3 Mreža radnih stanica

Sekvencijalni model izvršavanja je bio široko rasprostranjen pri rešavanju raznih problema. Najčešće platforme za razvoj, testiranje i izvršavanje programa su bile radna stanica (workstation) i PC računar zbog svoje niske cene. U poslednje vreme su performanse PC računara toliko napredovale, tako da danas praktično ne postoje razlike u performansama PC računara i radnih stanica. Uporedo sa razvojem pojedinačnih PC računara pristupilo se i njihovom povezivanju i umrežavanju u lokalne i globalne računarske mreže. Kvalitetni pomak je učinjen i u korišćenju sigurnijih i fleksibilnijih operativnih sistema za personalne računare (UNIX, Windows NT) što doprinosi pouzdanijem radu i korišćenju svih prednosti umrežavanja.

Sve te osobine su dovele do situacije da je postalo moguće, a i poželjno sa stanovišta performansi, korišćenje mreže radnih stanica kao paralelnog računara. Korišćenje prednosti više procesora i paralelnog izvršavanja instrukcija se kod PC računara, odnosno radnih stanica, odvijalo u dva smera:

- Ugradnja matičnih ploča koje podržavaju više procesora (2-8), koji komuniciraju preko zajedničke (deljene) memorije;
- Korišćenje lokalnih računarskih mreža kao paralelnog računara, gde procesori komuniciraju prosleđivanjem poruka preko date mreže. Jedan uspešan primer takve primene naveden je u [Hil98].

Detaljniji opis raznih aspekata mreže radnih stanica kao paralelnog računara možemo videti u [Wlk98], a specifične informacije vezane za paralelno izvršavanje na mreži radnih stanica pod Linux operativnim sistemom se mogu naći u [Kon98]. Zbog svih navedenih prednosti ova platforma je, kao najpogodnija, izabrana za paralelizaciju genetskog algoritma.

1.3.2.4 Superračunari i paralelni računari visokih performansi

Zajedničke karakteristike superračunara i višeprocesorskih sistema visokih performansi su:

- Kontrolisan i vrlo komplikovan pristup takvim sistemima, zbog njihove vrlo visoke cene;
- Vrlo skupo računarsko vreme i visoki troškovi razvoja, testiranja i izvršavanja aplikacija;
- Vrlo velika pažnja se posvećuje raznim optimizacijama i efikasnom paralelnom izvršavanju. Deo tih optimizacija vrše specijalizovani prevodioci datog programskog jezika, ali veliki deo mora obaviti sam programer. U suprotnom dolazi do osetne degradacije performasi celog sistema;
- Tehnike optimizacije i paralelizacije su ograničene samo na datu klasu višeprocesorskih računara. One su obično neprimenljive na ostale klase paralelnih računara.

Superračunare, zbog izuzetno visoke cene, poseduju samo neke velike kompanije i bogate institucije. Zbog izuzetno visokih troškova nabavke i održavanja takvih sistema, oni su praktično nedostupni za širu klasu istraživača i programera.

Iako je cena paralelnih računara visokih performansi manja nekoliko redova veličine u odnosu na superračunare, i u poslednje vreme prilično opada, za njih takođe važe gorepomenute karakteristike. Njih najčešće poseduju samo veći

univerziteti i istraživački centri u SAD, Japanu i zapadnoj Evropi. Pristup im je takođe vrlo ograničen.

Neki od poznatijih paralelnih sistema su: ASCI Red, Convex SPP1000, Cray T3D, Fujitsu VPP300, IBM SP-2, Intel Delta, Intel iPSC/2, Intel Paragon, Meiko CS2, SGI Origin 2000 i TMC CM-5. Detaljniji opis većine od njih, kao i njihovih performansi u praksi je dat u radovima [Hoc91], [Hoc94] i [Don97].

1.3.3 MPI standard

MPI (Message Passing Interface) je specifikacija standardne biblioteke funkcija za paralelni model prosleđivanja poruka. Definisanje standarda je trajalo preko 2 godine u organizaciji MPI foruma koji je sadržao više od 80 istraživača iz oko 40 institucija. U grupi koja je definisala standard su bili najzastupljeniji: proizvođači paralelnih računara, osobe zadužene za razvojne programske pakete i specijalisti za paralelne aplikacije.

MPI standard je preuzeo konstrukcije raznih postojećih sistema koji su se dobro pokazale u praksi, i inkorporirao ih u jedan efikasan sistem visokog nivoa. To se pokazalo kao bolje rešenje od direktnog preuzimanja nekog od postojećih sistema, od kojih je svaki imao određene nedostatke. Korišćene su konstrukcije sledećih sistema: CHIMP ([CHI92]), Express ([Exp92]), p4 ([But94]), Parmacs ([Cal94]), PVM ([Gei94]), Zipcode ([Skj92]), kao i sistema za podršku višeprocesorskih računara firmi Cray Research, IBM, Intel, Meiko i Ncube. Bez zahtevanja kompatibilnosti po svaku cenu, bilo je moguće projektovanje standarda koji zadovoljava i najzahtevnije primene, uz očuvanje efikasnosti izvršavanja. Većina koncepata koja je primenjena pri projektovanju MPI standarda nije nova, već je standardizovano višegodišnje iskustvo eksperata.

Početni dogovor oko MPI standarda je nastao aprila 1992 na međunarodnoj konferenciji Supercomputing ’92 i može se videti u [Don93]. Posle brojnih usaglašavanja konačna verzija MPI standarda 1.0 nastala je u maju 1994, i detaljno je opisana u knjigama [Gro94], [MPI94], [MPI95] i [Sni95]. On sadrži više od 100 funkcija visokog nivoa, što donekle produžava vreme učenja sintakse celog standarda, ali su zbog toga izražajne mogućnosti mnogo veće. Iako je za veliki broj primena dovoljno koristiti samo deo mogućnosti koje MPI podržava, za razumevanje njegovih karakteristika , je potrebno detaljno proučavanje. U suprotnom, pri nedovoljno pažljivoj primeni, moguć je veliki gubitak performansi korisničke aplikacije. Zbog toga je potrebno, osim same sintakse, voditi računa i o globalnoj koncepciji koju su imali autori MPI standarda, kao i o njihovim preporukama i savetima.

MPI standard sadrži sledeće konstrukcije:

- Pojedinačne međuprocesorske komunikacije (point-to-point communications) raznih tipova. Uz veći broj sistemskih tipova, moguće je korišćenje i korisnički definisanih tipova;
- Kolektivne operacije za međuprocesorsku komunikaciju nad sistemskim i korisnički definisanim tipovima;
- Grupe procesa i manipulacija kontekstima, koji čuvaju zajedničke osobine grupa procesa;
- Korisnički definisane topologije koje mogu biti realne (odgovaraju fizičkim vezama) ili virtuelne;
- Mogućnost svakog procesa da inicijalizuje, očitava ili menja komunikacioni kontekst svoje grupe procesa u toku izvršavanja.

Pošto je MPI standard nezavisan od platforme i operativnog sistema na kome se izvršava, svaka MPI implementacija mora dodatno da obezbedi ulazno/izlazne operacije i način izvršavanja aplikacije na datom konkretnom operativnom sistemu. Za prevođenje izvornog koda i ispravljanje grešaka se koriste standardni prevodioci i ostali alati posebno konfigurisani i podešeni za MPI aplikacije.

U verziji 1.0 nisu eksplicitno podržane operacije paralelnog modela sa deljenom memorijom, podrška nitima (threads), dinamičko alociranje i dealociranje procesa u toku izvršavanja programa, ali je u kasnijim verzijama otklonjen deo takvih nedostataka.

Pošto je u stvaranju MPI standarda učestvovao veći broj iskusnih sistemskih i aplikativnih programera specijalizovanih za paralelizaciju, prve implementacije su nastale vrlo brzo nakon usvajanja standarda. Zbog toga je MPI standard postao prihvaćen od mnogih, i pojavio se veći broj raznih MPI implementacija. Dati preduslovi su pogodno uticali na pojavu velikog broja paralelnih MPI aplikacija, a informacije o nekima od njih se mogu naći u [MPIb].

Kratak opis MPI standarda se može naći u radu [Don95] i dodatku D, a detaljnije informacije u knjigama [Gro94], [MPI94], [MPI95] i [Sni95]. Analiza performansi MPI programa je data u radu [Krl94].

1.3.4 MPI implementacije

Od nastanka 1994. godine, MPI standard je implementiran na velikom broju paralelnih platformi raznih osobina, od mreže radnih stanica do superračunara. Za neke od platformi postoji čak više različitih MPI implementacija. Neke od javno dostupnih implementacija su: MPICH, WMPI, MPI-FM, LAM, CHIMP/MPI, CRI/EPCC MPI for Cray T3D, MPIAP, RACE-MPI. Postoji i veliki broj komercijalnih MPI implementacija kao što su proizvodi sledećih kompanija: MPI Software Technology - MPI/PRO, Genias - PaTENT WMPI, Alpha - Data MPI, HP - MPI, IBM Parallel Environment for AIX - MPI Library, Hitachi - MPI, Silicon Graphics - MPI, Nec - MPI/DE, Intel - Paragon OS R1.4, Scali Computer - ScaMPI, Telmat Multinode - T.MPI, Periastron - TransMPI. Detaljnije informacije o ovim implementacijama se mogu naći u [MPIa].

Za nas su najvažnije one implementacije koje su primenljive na mreži radnih stanica, što je paralelna platforma iskorišćena u ovom radu. Poređenje performansi nekih od njih pod Windows NT operativnim sistemom je dato u [Bak98].

1.3.4.1 MPICH

MPICH ([Gro96c]) je implementacija nastala u Argonne National Laboratory (Ilinois - USA), i ona u potpunosti podržava MPI 1.1 standard. Javno je dostupna verzija za UNIX operativne sisteme gde su podržane sledeće platforme: IBM SP1 i SP2, TMC CM-5, Intel Paragon, IPSC860, Touchstone Delta, Ncube2, Meiko CS-2, Kendall Square KSR-1 i KSR-2, Convex Exemplar, IBM, SGI i Sun višeprocesorski sistemi. Takođe su podržane i mreže radnih stanica Sun4, Hewlett-Packard, DEC 3000 i Alpha, IBM RS/6000 i SGI familije, kao i mreže Intel x86 PC računara pod LINUX ili FreeBSD operativnim sistemom. Opis procedure za instalaciju ovog programskog paketa, kao i detaljno uputstvo za rad je dato u [Gro96a], [Gro96b] i [Gro97].

Ova implementacija je prenesena i na Intel x86 platformu pod Windows NT operativnim sistemom i nazvana je MPICH/NT. Međutim, javno su dostupne

samo verzije do 0.92 BETA ([MPICH96]), koje su potpuno funkcionalne, i sadrže sve što je neophodno za razvoj MPI paralelnih aplikacija, ali nemaju sve dodatne pogodnosti za komforan razvoj aplikacija. Kasnije verzije su opsežno modifikovane i dopunjene dodatnim mogućnostima za razvoj i ispravljanje paralelnih aplikacija, ali su komercijalizovane pod nazivom MPI/PRO ([MPI98a]).

1.3.4.2 WMPI

WMPI ([WMPI98a]) je implementacija za Intel x86 personalne računare pod 32-bitnim Windows operativnim sistemima (prvenstveno Windows NT). U potpunosti je kompatibilna sa MPICH implementacijom, a razvijena je u Instituto Supererior de Engenharia de Coimbra, Portugal. Implementacija MPI standarda je izvršena preko *p4* sistema komunikacije niskog nivoa. Čak je moguće izvršavanje pod mešovitom arhitekturom, gde se na nekim radnim stanicama izvršava WMPI pod Windows, a na drugim MPICH pod UNIX operativnim sistemom.

Takođe se može nabaviti i komercijalna verzija ovog paketa pod nazivom PaTENT WMPI ([WMPI98b]) kao deo WinPar evropskog projekta (WINdows based PARallel computing).

1.3.4.3 MPI-FM

MPI-FM ([MPIFM98]) je proizvod nastao u University of Illinois, Concurrent Systems Architecture Group i nastao je implementacijom MPI standarda preko vrlo efikasnog FM (Fast Messages) sistema za komunikaciju niskog nivoa. Iako se na većini verzija može izvršavati i sa Winsock drajverima na Ethernet mrežama, prvenstveno je predviđen za specijalne *Myrinet* mreže, i na njima su ostvarene vrhunske performanse.

1.3.4.4 LAM

Ohio Supercomputer Center je razvio paralelni sistem LAM 6.1 koji je opisan u [Bur94], i koji predstavlja MPI programsko okruženje i razvojni sistem prvenstveno za mreže radnih stanica sa raznorodnim čvorovima (heterogeneous computers on a network). LAM sadrži alate za razvoj i ispravljanje programa, mogućnost dinamičkog konfigurisanja i dinamičke manipulacije MPI procesima, tolerantnost na greške i efikasno implementiran sistem za međuprocesorsku komunikaciju. Podržani su sledeći sistemi: Sun (Solaris 5.4-5), SGI (IRIX 6.2), IBM RS/6000 (AIX V3R2), DEC Alpha (OSF/1 V4.0), HP PA-RISC (HP-UX 10.01), Intel X86 (LINUX v2.0.24).

1.3.4.5 Poređenje performansi

U [Bak98] su dati rezultati MPICH/NT, WMPI i MPI-FM implementacija pod Windows NT operativnim sistemom. Najbolje rezultate su pokazali WMPI i MPI-FM, dok MPICH/NT veoma zaostaje u performansama. WMPI se pokazao kao najefikasniji na većini, a MPI-FM na nešto manje testova, ali su razlike u performansama male.

Po mišljenju autora ovog rada WMPI implementacija se pokazala kao najjednostavnija za upotrebu, što je u skladu sa rezultatima ispitivanja datim u [Bak98]. Pošto je u slučaju WMPI zabeležen relativno najmanji broj bagova i nedostataka, a imajući u vidu prethodne napomene, ona je primenjena za paralelizaciju GA u ovom radu.

2. SEKVENCIJALNA IMPLEMENTACIJA (GANP)

Za rešavanje nekih praktičnih problema i ispitivanje teorijskih aspekata GA u ovom radu razvijena je GANP implementacija (Genetski Algoritam za rešavanje NP-kompletnih problema - Genetic Algorithm for solving the NP-complete problems).

Već je rečeno da se genetski algoritmi primenjuju za rešavanje širokog spektra optimizacionih problema. Iako se najveći broj implementacija oslanja na Holland-ov model prostog GA, zbog različitih svojstava datih problema, javila se potreba za velikim brojem tehnika koje dodatno poboljšavaju primenu GA. U odeljku 2.1 data je klasifikacija i napravljen je pregled nekoliko poznatijih GA implementacija, od kojih su neke dostupne preko Interneta.

U odeljcima 2.2 - 2.6 će biti opisana GANP implementacija, koja sadrži nekoliko varijanti genetskih operatora i funkcija prilagođenosti, uz brojne dodatne funkcije i fleksibilan način čitanja podataka iz konfiguracione datoteke. Takođe je implementirano nekoliko politika zamene generacija, fiksna ili adaptivna promena vrednosti parametara genetskog algoritma, kao i još neki važni aspekti za uspešano izvršavanje GA. Kao dodatni metod poboljšavanja performansi date implementacije, samostalno je osmišljena i po prvi put od strane autora ovog rada primenjena metoda keširanja GA. Data metoda za keširanje GA je detaljno opisana u poglavlju 4.

Napomenimo i to da je GANP implementacija poslužila kao osnova za rešavanje još nekih problema:

- U radovima [Šeš99a], [Šeš99b] i [Šeš00] je opisana primena GA (korišćenjem GANP) za rešavanje inverznog geofizičkog problema;
- Osnovni elementi GANP su iskorišćeni i pri rešavanju problema 2-povezanosti (biconnectivity problem) u grafovima, koji je takođe NP-kompletan. Detaljan opis date metode je dat u [Trm00].

2.1 Pregled nekih postojećih GA implementacija

Po načinu korišćenja, GA implementacije se mogu podeliti na: aplikativno orjentisane-sisteme, algoritamski orjentisane sisteme i razvojne alate.

2.1.1 Aplikativno orjentisani sistemi

Postoji veliki broj korisnika koji nisu zainteresovani za proučavanje GA, već samo za korišćenje gotovih programskih paketa koji rešavaju zadate konkretne probleme. Takav pristup generiše potrebu za aplikativno orjentisanim sistemima, gde se primenjeni GA tretira samo kao "crna kutija", a akcenat je na zadovoljavanju specifičnih zahteva korisnika. Date sisteme karakteriše vizuelno

prepoznatljiv sistem menija, alfa-numerički i grafički sistem praćenja rada i prezentacije konačnih rezultata, obiman i profesionalno urađena uputstva za korišćenje i sistemi za interaktivnu pomoć (online help) u toku izvršavanja programskog paketa. Najčešće su realizovani kao ekspertni sistemi sa primenom u poslovnom okruženju, pa su po pravilu komercijalni proizvodi. Neki od najpoznatijih programskih paketa ove klase su: PC/Beagle, XpertRule, GenAsys, Evolver, Omega, ... U radu [Rib94a] je dat njihov kratak pregled, ali i neke informacije o ostalim vrstama programskih paketa povezanim sa GA.

2.1.2 Algoritamski orjentisani sistemi

U ovu grupu spadaju sistemi koji sadrže pojedinačne GA implementacije ili skup programskih biblioteka za razvoj samostalnih genetskih algoritama. Algoritamski orjentisani sistemi su obično:

- razvijani na univerzitetima ili istraživačkim razvojnim centrima, u javnom su vlasništvu i dostupni preko Interneta;
- dati u izvornom kodu, pa je moguće prilagođavanje specifičnim zahtevima pri rešavanju konkretnih problema optimizacije;
- modularne strukture koja garantuje mogućnost zamene ili nadogradnje određenih delova;
- sa korisničkim interfejsom niskog nivoa, a najčešće bez sistema menija i grafičkog prikaza, već se samo koristi komandna linija;
- korišćeni kao opšte GA implementacije za rešavanje konkretnih problema, ali i za testiranje i analizu performansi specifičnih genetskih operatora ili algoritama.

Jedna od prvih i najpoznatijih GA implementacija opšte namene je bio Genesis, detaljno opisan u [Gre84], koji je vrlo dugo uspešno korišćen za realizaciju većeg broja genetskih operatora i njihovo testiranje u praksi (videti [Gre85] i [Gre87]). Implementirana je u C programskom jeziku, a verzija 5.0 se uspešno izvršava na personalnim računarima i SUN radnim stanicama. Programski kod je razvijan portabilno, pa uz manje izmene može da se izvršava i na ostalim sistemima. Koristi se gotova biblioteka genetskih operatora selekcije, ukrštanja i mutacije, a korisnik samo definiše vrednosnu funkciju, koja zavisi od prirode konkretnog problema. Date su tri mogućnosti za kodiranje jedinki: binarna reprezentacija, alfanumerička i reprezentacija pomoću realnih brojeva. Na osnovu kodiranja izabranog od strane korisnika, sistem datu strukturu automatski pakuje tako da prostorna i vremenska efikasnost njene manipulacije bude što veća. Genesis programski paket se sastoji od tri programske celine: *set-up*, *report* i *ga*. Program *set-up* učitava parametre genetskog algoritma koje zadaje korisnik, smešta ih u konfiguracionu datoteku, i koristi ih pri izvršavanju genetskog algoritma u programskoj celini *ga.* Na taj način se mogu zadati: tip kodiranja, broj gena, broj eksperimenata, broj izvršavanja u svakom eksperimentu, veličina populacije, broj bitova u genetskom kodu jedinke, nivo ukrštanja, nivo mutacije i ostali parametri važni za primenu GA. Za svaki od parametara je unapred predefinisana vrednost, koja se koristi ukoliko korisnik zbog nekih razloga propusti da definiše određenu vrednost.

GAGA (The Genetic ALgorithms for General Application) je bila jedna od prvih GA implementacija u Evropi (Pascal verzija: Hillary Adams, University of York; C verzija: John Crowcroft, University College London). Korisnik mora da zada ciljnu funkciju koja se optimizuje, vrstu optimizacije (minimum ili

maksimum) i parametre potrebne za izvršavanje GA. Program izvršava GA u skladu sa izabranim parametrima, i štampa odgovarajući izveštaj. Uz pomoć date implementacije su dobijeni dobri rezultati, čak i pri izvršavanju na relativno teškim problemima.

Genitor (Genetic Implementator) je modularno implementiran programski paket u programskom jeziku C, koji dopušta binarno, celobrojno i kodiranje pomoću realnih brojeva. Iako dopušta i drugačiji pristup, on ipak forsira korišćenje selekcije zasnovane na rangu ([Whi89]) i stacionarnog GA. Pri datoj politici zamene generacija, generiše se samo jedna nova jedinka (potomak), a ona zamenjuje najlošiju jedinku u populaciji. Pošto se u svakoj generaciji vrši zamena samo jedne jedinke, njeno rangiranje u populaciji je vrlo jednostavno, a time i računanje prilagođenosti svih jedinki u populaciji. Detaljan opis implementacije se može videti u [Whi88]. Kasnije je implementirana i distribuirana verzija date implementacije pod nazivom Genitor II, čiji se opis može naći u [Whi90].

GENEsYs implementacija detaljno je opisana u [Bëc92b] i predstavlja proširenje Genesis 4.5 implementacije. Nadogradnja je izvršena u sledećim elementima:

- m-poziciono [DJo75], uniformno ukrštanje [Sys89], diskretna rekombinacija i rekombinacija u srednjem [Bëc91a];
- Adaptivna promena nivoa mutacije [Bëc92a];
- (μ, λ) - selekcija [Bëc91b] i Boltzmann-ova selekcija [Gol90];
- Veliki broj test funkcija obmanjivačkog tipa (deceptive functions), relativno težih za optimizaciju.

Programski kod je takođe napisan na C programskom jeziku, i implementiran za UNIX operativne sisteme.

GAGS je jedna od prvih objektno orjentisanih GA implementacija u javnom vlasništvu i dostupnih preko Interneta [Mer94a]. Realizovane su standardne varijante genetskih operatora: jednopoziciono, dvopoziciono i uniformno ukrštanje; prosta mutacija i transpozicija. Međutim, realizovani su i genetski operatori promenljive dužine: dodavanje gena na slučajan način, brisanje nekog gena pojedinačno ili kopiranje uz primenu mutacije određenog gena. Konfiguracija datog programskog paketa se može izvršiti preko sistema menija, a rezultati izvršavanja se mogu prikazati i grafički. Detaljnije informacije o svim aspektima ove implementacije se mogu videti u uputstvu za korisnike osnovnog paketa [Mer94a] i uputstva za programere datog u [Mer94b], gde su opisane napredne tehnike korišćenja date GA programske biblioteke.

EM (Evolutionary Machine) je programska biblioteka koja, osim više varijanti GA, sadrži i implementaciju nekoliko evolucionih strategija, a razvijena je na Institute for Informatics and Computing Techniques, Germany. Ovakva kombinacija genetskih algoritama i evolucionih strategija obezbeđuje efikasno izvršavanje na širokom spektru optimizacionih problema, koji mogu imati veoma različite numeričke karakteristike. Korisnik samo zadaje funkciju prilagođenosti u programskom jeziku C. Predefinisane (default) vrednosti su tako izabrane, da odgovaraju većini primena, pa je intervencija korisnika svedena na minimum. Podaci se prikazuju u toku i na kraju izvršavanja GA, a pored toga moguć je i grafički prikaz u jednoj, dve ili tri dimenzije. Program je razvijan za personalne računare, a za prevođenje izvornog koda se koristi Turbo C ili Turbo C++ prevodilac pod DOS operativnim sistemom. U [Voi91] se može videti detaljniji opis date implementacije.

OOGA (The Object-Oriented Genetic Algorithm) je još jedna objektno orjentisana GA implementacija, čiji je autor Lawrence Davis, a prvobitna namena je bila podrška primerima iz njegove knjige [Dav91]. Njen cilj je razvoj i testiranje novih genetskih algoritama i genetskih operatora, kao i prilagođavanje već postojećih raznim primenama. Svaka korišćena GA tehnika je implementirana kao poseban objekat pa je pregled i eventualna promena nekog dela GA jednostavna zahvaljujući fleksibilnosti objektno-orjentisane programske paradigme. Program sadrži tri glavna modula koji su zaduženi za sledeće namene:

- *Evaluacijski* računa vrednost i prilagođenost svake od jedinki u populaciji;
- *Populacijski* je zadužen za smeštanje i manipulaciju jedinkama, kao što su kodiranje i inicijalizacija populacije;
- *Reprodukcioni* koji konfiguriše, a zatim primenjuje izabrane genetske operatore. Sistemski je podržan veći broj genetskih operatora, a njihova primena i raspored nisu fiksirani, već se mogu međusobno kombinovati.

2.1.3 Razvojni alati

Razvojni alati se mogu generalno podeliti na dve velike grupe:

- Školski GA sistemi za obučavanje korisnika
- Sistemi GA opšte namene.

Školski GA sistemi su razvijeni za pomoć i upoznavanje novih korisnika sa raznim delovima genetskog algoritma, i o njegovim praktičnim aspektima. Da bi bili što jednostavniji za korišćenje, konfigurisanje parametara GA je maksimalno olakšano i realizuje se preko sistema menija. Primer ovakvog programskog paketa je GA Workbench, opisan detaljno u [Hug89].

Sistemi GA opšte namene su razvijeni za zahtevne korisnike, koji ne samo što žele da razviju sopstvene aplikacije i algoritme, već žele da na lak način prilagode ceo sistem svojim potrebama. Oni moraju da zadovolje nekoliko kriterijuma, među kojima su najvažniji:

- Dobro osmišljeno grafičko okruženje i pogodan sistem menija;
- Parametrizovana biblioteka algoritama;
- Jezik visokog nivoa za programiranje genetskih algoritama;
- Otvorena arhitektura fleksibilna za dalju nadogradnju.

Broj ovakvih sistema je u stalnom porastu, zbog rastućeg interesovanja za primene GA u širokom opsegu optimizacionih problema. Neki od najpoznatijih sistema GA opšte namene, koji pripadaju ovoj kategoriji, su:

- Splicer [Bay91] koji uvodi pojam razmenljivih biblioteka;
- MicroGA [MGA92] koji sadrži objektno orjentisanu okolinu za razvoj, jednostavnu za korišćenje;
- Paralelni programski sistemi Engineer [Rob92], GAME [Rib94b] i Pegasus [M**ü**h95].

2.2 Opis podataka

Praktična osnova ovog rada je GANP implementacija i njena kasnija paralelizacija. Ona je nastala kao pokušaj primene generalnih principa GA za rešavanje više NP-kompletnih problema. Primenjeni su neki standardni principi koji su se dokazali u praksi, uz istovremeno poboljšavanje korišćenjem nekih

novih, prvi put primenjenih, metoda. Svi elementi su ukomponovani u jednu celinu i maksimalno je olakšano programiranje pri rešavanju novih NP-kompletnih problema, uz očuvanje efikasnosti cele implementacije. Pošto genetski algoritam sadrži veliki broj parametara koji su bitni za izvršavanje, kao i veći broj varijanti genetskih operatora, njihovo izbor i zadavanje je izvršeno na vrlo fleksibilan način. Pri rešavanju problema opisanih u ovom radu, dobijeni su veoma dobri rezultati, u nekim slučajevima čak bolji od svih dosadašnjih primenjivanih metoda.

Neke važne karakteristike GANP implementacije su:

- Formiranje strukture koja sadrži sve informacije o GA, zajedničke za svaki problem koji se rešava;
- Formiranje strukture koja sadrži informacije specifične za problem koji se rešava;
- Izbor raznih varijanti operatora selekcije, ukrštanja i mutacije, kao i izbor funkcije prilagođenosti i kriterijuma za kraj izvršavanja GA se vrši pomoću konfiguracione datoteke. Pošto su korišćeni funkcijski pokazivači, očuvana je brzina izvršavanja, uz mogućnost variranja raznih operatora promenom u konfiguracionoj datoteci, bez ponovnog prevođenja programa.
- Fleksibilan način čitanja podataka iz konfiguracione datoteke;
- Mogućnost nadogradnje nekim specifičnijim potrebama (meta GA, paralelni GA, itd).
- Korišćenje generatora slučajnih brojeva datog u izvornom kodu, što doprinosi nezavisnosti koda od platforme na kojoj se izvršava, mogućnost determinisanog testiranja programa i jednake rezultate izvršavanja na svim platformama;
- Laka dopuna funkcijama koje zavise od prirode problema.

Svi podaci specifični za GANP implementaciju su grupisani u jednu strukturu C programskog jezika. Ona je podeljena po načinu korišćenja na podstrukture koje su zadužene za sledeće elemente GA:

- Populacija sastavljena od jedinki;
- Funkcija prilagođenosti;
- Politika zamene generacija;
- Selekcija;
- Ukrštanje;
- Mutacija;
- Zajedničke funkcije;
- Keširanje GA;
- Kriterijum završetka.

U ovom odeljku je dat samo pregled promenljivih u odgovarajućoj podstrukturi, a pregled funkcija, koje mogu biti dodeljene odgovarajućem funkcijskom pokazivaču, se može videti u narednom odeljku. Detaljne pojedinačne informacije o parametrima, koje sadrže tip i opseg odgovarajućeg numeričkog parametra, se mogu videti u dodatku B.

2.2.1 Osnovni podaci

2.2.1.1 Globalni deo

Neke promenljive su od većeg značaja ili se ne mogu klasifikovati u neku od postojećih podstruktura pa su zadate u globalnoj GA strukturi. U ovoj kategoriji su sledeći podaci:

- Redni broj tekuće generacije GA koja se izvršava (N_{gener});
- Ukupan broj jedinki u populaciji (N_{pop});
- Indeks tekuće jedinke koju obrađujemo u datom trenutku;
- Tip optimizacije problema koji rešavamo (minimum ili maksimum);
- Fiksno ili varijabilno dodeljivanje početnog elementa u generatoru slučajnih brojeva (random seed). Ukoliko je fiksno dodeljivanje tada se zadaje i vrednost datog početnog elementa.

2.2.1.2 Jedinka

Ova podstruktura sadrži pokazivače na jedinke u populaciji. Za svaku jedinku su definisani sledeći podaci:

- Dužina genetskog koda jedinke;
- Genetski kod jedinke;
- Dužina privremenog genetskog koda jedinke;
- Privremeni genetski kod jedinke;
- Vrednost date jedinke;
- Prilagođenost date jedinke;
- Indikator korektnosti jedinke.

Osim prostora za genetski kod jedinke rezervisan je memorijski prostor i za privremeni genetski kod jedinke. On služi kao pomoćni niz za primenu genetskih operatora selekcije i ukrštanja. Iako se time u nekoj meri povećava potrošnja memoriskog prostora, izvršavanje je brže, što doprinosi i efikasnosti cele implementacije.

U primeni GANP na NP-kompletne probleme, opisane u ovom radu, svi genetski kodovi su bili iste dužine, ali u nekim primenama to nije optimalno ili čak i nije moguće. Zbog toga se za svaku jedinku posebno zadaje dužina genetskog koda, pa je moguća primena i genetskog algoritma sa jedinkama nejednake dužine.

Korektnost jedinke je vrlo važna osobina. Pošto pri rešavanju NP-kompletnih problema, opisanih u ovom radu, nema mnogo nekorektnih jedinki, one se ne uzimaju u obzir pri računanju funkcije prilagođenosti, već im se odmah dodeljuje nulta vrednost prilagođenosti. Ukoliko bi veliki broj jedinki bio nekorektan, one bi se morale uzimati u obzir pri računanju funkcije prilagođenosti i na njih bi se morali primenjivati genetski operatori. Tada se primenjuje neka od strategija detaljnije opisanih u odeljku 1.2.3.1 .

2.2.1.3 Funkcija prilagođenosti

Podaci vezani za funkciju prilagođenosti su dati u ovom delu, a oni se sastoje od sledećih promenljivih:

- Pokazivač na funkciju prilagođenosti;
- Konstante c_A i c_B u slučaju da je izabrano linearno skaliranje;
- Konstanta c_C ako se primenjuje skaliranje pomoću sigma odsecanja;

- Eksponent ako se vrši stepenovanje kao deo računanja funkcije prilagođenosti.

2.2.1.4 Politika zamene generacija

Primenjeno je nekoliko politika zamene generacija, a odgovarajuće promenljive su:

- Pokazivači na funkcije za politiku zamene generacija pre i posle primene genetskih operatora;
- Broj elitnih jedinki na koje se ne primenjuju genetski operatori, već direktno prelaze u narednu generaciju (obeležimo ga sa N_{elite});
- Pokazivač na funkciju za računanje indeksa sličnosti jedinki u populaciji;
- Sličnost jedinki u tekućoj generaciji;
- Redni broj generacije u kojoj je dobijena tekuća najbolja jedinka.

2.2.2 Genetski operatori

2.2.2.1 Selekcija

Operator selekcije, takođe, sadrži nekoliko važnih parametara, koji se koriste ukoliko je izabran neki od modela opisanih u odeljku 2.3.2 . Ova podstruktura sadrži sledeće parametre:

- Pokazivač na funkciju koja realizuje operator selekcije;
- Broj jedinki koje su direktno izabrane, u slučaju stacionarnog GA, pa na njih ne primenjujemo operator selekcije (obeležimo ga sa N_{pass});
- Pokazivač na niz rangova u slučaju ako je izabrana selekcija zasnovana na rangu;
- Broj jedinki koje učestvuju na turniru ukoliko se primenjuje turnirska selekcija;
- Prosečna veličina turnira za fino gradiranu turnirsku selekciju.

2.2.2.2 Ukrštanje

U slučaju operatora ukrštanja mogu se pojaviti sledeći parametri:

- Pokazivač na funkciju za realizaciju operatora ukrštanja;
- Nivo (učestanost) ukrštanja p_{cross};
- Indikator da li je nivo ukrštanja stalan ili se menja tokom generacija;
- Koeficijent koji koristi ako se primenjuje promenljiv nivo ukrštanja;
- Verovatnoća sa kojom se ukršta neki gen ako se primenjuje uniformno ukrštanje p_{unif};
- Broj pozicija za višepoziciono ukrštanje;
- Veličina svakog gena, ukoliko gen sadrži više bitova u genetskom kodu;
- Ukupan broj korisnih bitova u genetskom kodu jedinke.

U većini slučajeva se primenjuje konstantan nivo ukrštanja tokom generacija GA. Međutim, u nekim slučajevima, kao što smo napomenuli u odeljku 1.2.3.6, pogodnije je da se nivo ukrštanja menja tokom generacija. Koeficijent koji se pri tome primenjuje može se memorisati u jednoj promenljivoj, iako on ima različito značenje, u zavisnosti od toga da li je promena nivoa ukrštanja statička ili dinamička.

Pošto je, zbog efikasnijeg izvršavanja, genetski kod podeljen na 32-bitne reči, može se dogoditi da neki bitovi u poslednjoj reči genetskog koda budu

neiskorišćeni. U nekim slučajevima je pogodnije ako genetske operatore, ili neke druge funkcije, primenjujemo samo na korisne bitove u genetskom kodu, pa je zbog toga potrebno memoristi njihov ukupan broj. Ovaj podatak zavisi od prirode samog problema i veličine instance koji se izvršava, pa vrednost date promenljive postavlja funkcija za inicijalizaciju konkretnog problema. Međutim, kada je on memorisan u ovoj promenljivoj, on postaje opšti pa se može koristiti nezavisno od prirode samog problema.

2.2.2.3 Mutacija

U podstukturi zaduženoj za mutaciju date su sledeće promenljive:

- Pokazivač na funkciju za inicijalizaciju operatora mutacije;
- Pokazivač na funkciju za realizaciju operatora mutacije;
- Maska koja služi za postavljanje svih neiskorišćenih bitova u genetskom kodu na nultu vrednost;
- Nivo (učestanost) mutacije p_{mut};
- Indikator da li je nivo mutacije stalan ili se menja tokom generacija;
- Parametar koji se koristi pri statičkoj eksponencijalnoj ili dinamičkoj promeni nivoa mutacije;
- Koeficijent eksponenta pri statičkoj eksponencijalnoj promeni mutacije;
- Pokazivač na niz inverznih vrednosti normalne raspodele.

U ovoj implementaciji postoji samo jedan generalni operator mutacije, samo je on realizovan na različite načine, od kojih su neki efikasniji od drugih. Kod realizacije preko normalne raspodele se unapred na slučajan način određuje broj mutacija, a zatim mesta mutacije. Na taj način se obrađuju samo reči koje imaju gene za mutiranje što je, u najvećem broju slučajeva, efikasnije od direktnog pristupa gde se obrađuje ceo genetski kod jedinke. Pri tome se na početku iz datoteke učitava niz koji sadrži inverzne vrednosti normalne raspodele, koji se kasnije koristi. I u slučaju operatora mutacije, isto kao i kod ukrštanja, je moguće statički ili dinamički menjati nivo mutacije tokom generacija.

2.2.3 Ostali podaci

2.2.3.1 Zajedničke funkcije

U ovom delu su dati pokazivači na zajedničke funkcije GANP implementacije. Iako su skoro sve ove funkcije definisane u delu koji zavisi od prirode problema, pokazivač je zadat u ovom zajedničkom delu pa je moguće pozivanje date funkcije nezavisno od prirode samog problema. Funkcijski pokazivači dati u ovoj podstrukturi su:

- Inicijalizacija parametara za dati problem koji rešavamo;
- Ulaz podataka za konkretni problem;
- Konverzija genetskog koda jedinke u argumente datog konkretnog problema;
- Konverzija argumenata datog problema u genetski kod jedinke;
- Na osnovu argumenata problema računanje vrednosti (jedinke);
- Eventualna heuristika za poboljšavanje vrednosti neke jedinke u populaciji;
- Inicijalizacija generisanja izveštaja za GA;
- Generisanje izveštaja za GA;

- Generisanje izveštaja o datom problemu;
- Inicijalizacija generisanja završnog izveštaja za GA;
- Generisanje završnog izveštaja za GA;
- Generisanje završnog izveštaja o datom problemu.

2.2.3.2 Keširanje GA

Keširanje GA je originalni doprinos autora ovog rada i ima veliki značaj u poboljšavanju performansi GA pri rešavanju NP-kompletnih problema. Zbog toga je ova podstruktura odvojena za parametre keširanja, i sastoji se od sledećih podataka:

- Pokazivači na funkcije za: inicijalizaciju keširanja, pretraživanje keš memorije za zadatom jedinkom, ažuriranje bloka keš memorije koji sadrži pronađenu jedinku i dodavanje nove jedinke;
- Maksimalan i tekući broj zauzetih blokova keš memorije;
- CRC vrednost tekuće jedinke (CRC - Cyclic Redundancy Code);
- Pokazivač na keš memoriju;
- Koeficijenti pri generisanju CRC vrednosti;
- Tekuće vrednosti heš-funkcije u obe heš-tabele ako se primenjuje keširanje pomoću dvostruke heš-tabele;
- Minimalna, maksimalna i tekuća vrednost funkcije za oslobađanje blokova ukoliko je keš memorija puna, u slučaju keširanja pomoću dvostruke heš-tabele.

Svaki blok keš memorije služi za memorisanje jedne jedinke i (kasnije) eventualno direktno očitavanje njene vrednosti, umesto ponovnog izračunavanja vrednosne funkcije nad datom jedinkom. U slučaju linearne organizacije keš memorija je organizovana kao niz pokazivača na blokove keš memorije, dok u slučaju dvostruke heš-tabele postoje dva niza pokazivača, svaki za po jednu heš-tabelu. Maksimalni broj blokova keš memorije je unapred zadat, a svaki od njih ima sledeću organizaciju:

- Dužina genetskog koda memorisane jedinke;
- Genetski kod memorisane jedinke;
- Indikator da li je već izračunata vrednost jedinke:
- Vrednost jedinke;
- Indikator, da li je jedinka korektna;
- Izračunata CRC vrednost jedinke;
- Vrednosti heš-funkcije jedinke u heš-tabeli;
- Pokazivači na sledeće i prethodne blokove keš memorije u dvostruko povezanim listama koje se formiraju za red i svaku poziciju heš-tabele.

Indikator da li je već (ranije) izračunata vrednost jedinke datog bloka keš memorije koristi se samo u paralelnoj GA implementaciji. U sekvencijalnom modelu izračunavanja za njegovim korišćenjem nema potrebe, pošto je vrednost svake jedinke već izračunata u momentu pretraživanja druge jedinke u keš memoriji. Detaljnije o ovom indikatoru se može videti u narednom poglavlju gde je opisana PGANP implementacija.

Poslednje četiri stavke se koriste samo u slučaju kada je za keširanje GA primenjena heš-red struktura i detalji o tome se mogu videti u poglavlju 4.

2.2.3.3 Kriterijum završetka

Primenjeno je više kriterijuma završetka izvršavanja GA i njihovih kombinacija, čiji su podaci grupisani u ovoj podstrukturi. Ona sadrži sledeće informacije:

- Pokazivač na funkciju koja određuje da li je nastupio kraj izvršavanja GA;
- Maksimalan broj generacija N_{gener};
- Maksimalan (N_{rep}) i tekući broj generacija tokom kojih se najbolja jedinka nije promenila;
- Indikator da li je dozvoljeno da korisnik prekine izvršavanje GA;
- Kod koji označava razlog prekida izvršavanja GA;
- Maksimalna dozvoljena sličnost jedinki u populaciji;
- Pokazivač na funkciju koja dokazuje optimalnost najbolje jedinke, ukoliko je takva funkcija moguća i implementirana u delu koji zavisi od prirode samog problema;
- Početna generacija za dokazivanje optimalnosti, ako je realizovana funkcija za dokazivanje optimalnosti;
- Indikator da li je dokazana optimalnost tekuće jedinke.

2.2.4 Druge strukture

Osim prethodne GA strukture, koja je najznačajnija, postoji još nekoliko drugih struktura koje sadrže sledeće podatke:

- Podaci vezani za konkretan problem;
- Opis podataka u konfiguracionim datotekama;
- Nizovi funkcijskih pokazivača, koji služe za dodelu odgovarajuće funkcije.

2.2.4.1 Problem struktura

Svaki problem koji se rešava ima i svoje specifične podatke, koji se ne mogu primeniti za rešavanje nekih drugih problema. Oni su zadati u ovoj strukturi, a mogu se u najvećem broju slučajeva podeliti u sledeće kategorije:

- Ulazni podaci konkretne instance problema koji rešavamo;
- Izlazni podaci koji sadrže opis rešenja;
- Međupodaci dobijeni u toku izračunavanja;
- Promenljive vezane za, eventualne, heuristike za dobijanje početne populacije;
- Promenljive koje koriste, eventualne, heuristike za poboljšavanje rešenja;
- Podaci o optimalnom rešenju, ako je ono unapred poznato.

2.2.4.2 Opis konfiguracionih datoteka

Postoje i dve strukture koje služe za opis podataka u konfiguracionim datotekama. Prva je potrebna za učitavanje GA konfiguracione datoteke u datu strukturu, a druga za učitavanje konfiguracione datoteke vezane za konkretan problem u odgovarajuću strukturu.

2.2.4.3 Nizovi funkcijskih pokazivača

Za svaki funkcijski pokazivač u GA i problem strukturi, postoji i odgovarajući niz funkcijskih pokazivača, koji sadrži skup dopustivih funkcije koje mu se mogu dodeliti. Dodela neke od dopustivih funkcija funkcijskom pokazivaču se vrši pri učitavanju odgovarajuće konfiguracione datoteke.

Ovakav pristup u nekoj meri komplikuje samu implementaciju, ali je moguće vršiti konfiguraciju celog sistema bez potrebe za prevođenjem programa, već se samo promeni odgovarajući slog u konfiguracionoj datoteci. Na taj način je moguće distribuirati samo izvršnu verziju programa, bez potrebe za izvornim kodom i njegovim prevođenjem.

2.3 Osnovni deo implementacije

Jedan od najvažnijih aspekata svake GA implementacije su kodiranje, funkcija prilagođenosti, politika zamene generacija i genetski operatori. Zbog toga je posebna pažnja posvećena ovim aspektima, prilagođena osnovnoj nameni ove implementacije za rešavanjem NP-kompletnih problema.

2.3.1 Kodiranje

Osnovni način kodiranja u GANP je binarno kodiranje, u kome je svaki gen dužine jednog ili više bitova. Zbog toga je posebna pažnja posvećena efikasnosti manipulacije nad tako definisanim genetskim kodovima jedinki. Genetski kod je, radi efikasnosti izvršavanja, podeljen na 32-bitne reči. Pri takvoj organizaciji se može desiti da određen broj bitova u poslednjoj reči bude neiskorišćen. Gubitak memorijskog prostora može biti značajan za instance male dimenzije, međutim to postaje praktično beznačajno kada rešavamo instance datog problema koje su velike dimenzije. Pošto su memorijski kapaciteti savremenih računara vrlo veliki, primarni faktor postaje brzina izvršavanja, a ona je ovakvim pristupom maksimalno očuvana.

Implementirani genetski operatori se mogu direktno primeniti i u slučaju ako su geni nekog drugog tipa (recimo realni brojevi), pa se pomoću GANP implementacije mogu rešavati i takvi problemi. Međutim, primarni cilj ove implementacije (do sada) nije bio rešavanje numeričkih problema, gde se geni realnog tipa najčešće javljavljaju, već rešavanje problema kombinatorne optimizacije gde je takvo kodiranje relativno ređe. I pored toga, ova implementacija može uspešno rešavati numeričke probleme male ili srednje dimenzije, a za rešavanje problema velike dimenzije bilo bi potrebno primeniti i neke specijalne metode GA koje su prilagođene ovim problemima. Jedna od uspešnih primena GANP za projektovanje optimalne trase cevovoda se može naći u [Kra97c].

2.3.2 Funkcija prilagođenosti

Kao što je već rečeno postoji veći broj načina za računanje funkcije prilagođenosti. Oni odgovaraju različitim vrstama problema koji se rešavaju. Najčešće se primenjuju različite vrste skaliranja, koje doprinose izbegavanju preuranjene konvergencije i gubljenja genetskog materijala.

Neki od metoda koji su poznati u literaturi kao rangiranje prilagođenosti jedinki u populaciji (fitness ranking) i implicitna funkcija prilagođenosti (implicit fitness remapping) nisu samostalno implementirani, već su realizovani kroz odgovarajuće operatore selekcije. Tako je rangiranje prilagođenosti jedinki u populaciji izvedeno kroz selekciju zasnovanu na rangu, a implicitna funkcija prilagođenosti preko turnirske selekcije.

2.3.2.1 Direktno preuzimanje

Ovo je najprostiji slučaj kada se za prilagođenost jedinke direktno preuzima njena vrednost, kao što se može videti iz formule (2.1).

$$f(x) = x \tag{2.1}$$

Funkcija prilagođenosti data u (2.1) se može primeniti samo u slučaju problema maksimizacije, što je još jedan njen nedostatak. Ovaj nedostatak je moguće relativno lako ispraviti ako za minimizacione probleme primenimo sledeću funkciju datu u (2.2).

$$f(x) = M - x \tag{2.2}$$

Potrebno je da vrednost M bude dovoljno velika da bi funkcija prilagođenosti f bila nenegativna. Ovo je neophodno da bi se mogao primeniti operator selekcije. Svi operatori selekcije zahtevaju da sve jedinke imaju nenegativnu prilagođenost. Ni ovo nije najpogodnije za primenu jer vrednost M nije uvek lako unapred odrediti.

Međutim, čak i da to ispravimo, ovakav pristup nije u skladu sa praktičnim aspektima GA. Pošto u praksi daje loše rezultate čak i na "školskim' test-primerima relativno manje dimenzije, dati pristup se primenjuje samo u slučaju prostog genetskog algoritma, i to samo za rešavanje relativno prostijih problema.

2.3.2.2 Linearno skaliranje

U ovom slučaju se primenjuje funkcija prilagođenosti oblika:

$$f(x) = c_A \cdot x + c_B \tag{2.3}$$

Koeficijenti c_A i c_B su konstantni i unose se iz konfiguracione datoteke. Iako je nešto bolji od prethodnog pristupa, ovakav način računanja prilagođenosti ima slične nedostatke. Zbog toga se situacija može prilično popraviti ako koeficijenti A i B nisu konstantni, već se menjaju tokom generacija.

Jedan od takvih pristupa određuje koeficijente A i B tako da je sredina intervala u koji skaliramo približno jednaka srednjoj vrednosti jedinki u populaciji. Funkcija koristi parametar C čija se vrednost bira eksperimentalno, a obično je u intervalu [1, 3]. U slučaju da je interval u koji skaliramo već nenegativan, odgovarajuća formula je (2.4), a u suprotnom se koristi formula (2.5).

$$\begin{aligned} \delta &= max - \bar{x} \\ A &= (C - 1) * \bar{x} / \delta \\ B &= \bar{x} * (max - C * \bar{x}) / \delta \end{aligned} \tag{2.4}$$

$$\begin{aligned} \delta &= \bar{x} - min \\ A &= \bar{x} / \delta \\ B &= - min * \bar{x} / \delta \end{aligned} \tag{2.5}$$

Vrednosti min, max i $\bar{x}$ su redom minimalna, maksimalna i srednja vrednost jedinki u populaciji. Primetimo da je ovo skaliranje takvo da rezultujuća funkcija prilagođenosti vraća isključivo nenegativne vrednosti.

2.3.2.3 Skaliranje u jedinični interval

Pri ovakvom pristupu vrednosti jedinki se skaliraju u interval [0, 1] tako da prilagođenost najbolje jedinke bude 1, a najlošije 0. Ukoliko je u pitanju problem maksimizacije najbolja jedinka je ona sa najvećom vrednošću, pa se za takvo skaliranje primenjuje formula (2.6). U suprotnom, za probleme minimizacije primenjuje se formula (2.7) koja vrednosti jedinki inverzno skalira u jedinični interval.

$$f(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{2.6}$$

$$f(x) = \frac{x_{\max} - x}{x_{\max} - x_{\min}} \tag{2.7}$$

U ovom slučaju su x, x_{min} i x_{max} redom tekuća, minimalna i maksimalna vrednost jedinki u populaciji. Ovako zadata funkcija prilagođenosti f nije definisana ako sve jedinke u populaciji imaju istu vrednost. Međutim, taj slučaj se lako razrešava, jer se to dešava po pravilu samo kada su sve jedinke iste, što znači da je genetski algoritam konvergirao u toj vrednosti, pa dalje izvršavanje nema smisla. Zbog toga se u tom slučaju odmah prekida izvršavanje GA i štampa odgovarajući izveštaj.

Iako relativno jednostavan, ovaj način skaliranja u velikoj meri prati samu prirodu genetskog algoritma, jer se postepeno tokom generacija pojačava selekcioni pritisak na populaciju. Zbog toga se u praksi dobijaju dobri rezultati, čak i u slučaju primene prostog genetskog algoritma, kao što se može videti i u radovima [Kra96a], [Kra98b] i [Kra99a].

2.3.2.4 Sigma odsecanje

Funkcija prilagođenosti po shemi sigma-odsecanja (sigma-truncation) je data formulom:

$$f(x) = \begin{cases} x - (\bar{x} - C * \sigma), & x \geq \bar{x} - C * \sigma \\ 0 \quad , & x < \bar{x} - C * \sigma \end{cases} \tag{2.8}$$

Promenljive $\bar{x}$ i σ redom označavaju srednju vrednost i standardnu devijaciju vrednosti jedinki u populaciji. Detaljnije o ovom skaliranju se može videti u [Sri94].

2.3.2.5 Uklanjanje višestruke pojave jedinki u populaciji

Cilj genetskog algoritma je da dobre jedinke i dobri geni češće prolaze u sledeću generaciju, da bi njihovim ukrštanjem dobijali još bolje jedinke. Međutim, širenjem dobrih jedinki u populaciji može da izostane i širenje dobrih gena. Zbog toga se može destiti da pojedine dobre jedinke počinju da dominiraju populacijom, izbacujući druge jedinke koje imaju možda nešto lošiju vrednost, ali koje mogu da sadrže dobre gene. Na taj način se vrši samo kopiranje dobrih jedinki u više primeraka za narednu generaciju, bez mogućnosti da se one ukrštanjem poboljšaju, jer je genetski materijal nepovratno izgubljen.

Jedan od načina za sprečavanje gubitka genetskog materijala je i uklanjanje višestruke pojave jedinki u populaciji. Pošto bi fizičko uklanjanje bilo relativno sporo, a pojavili bi se i dodatni problemi pošto je veličina populacije N_{pop} u ovoj implementaciji konstantna tokom generacija, jedinke se markiraju u tekućoj a implicitno uklanjaju u sledećoj generaciji. To se vrši dodeljivanjem nulte vrednosti za prilagođenost takvih jedinki, pa ih operator selekcije uopšte ne uzima u obzir pri izboru jedinki za narednu generaciju, što možemo videti i u primeru 2.1.

Primer 2.1. Neka su jedinke predstavljene binarnim nizom dužine 2, i neka populacija u nekoj generaciji sadrži sledeće jedinke: 00, 01, 01, 00, 11, a vrednost jedinki u populaciji data sledećom tabelom:

00	0.2
01	0.3
10	0.5
11	0.7

Pretpostavimo da je primenjena najprostija funkcija prilagođenosti koja direktno preuzima vrednosti jedinki i da se vrši uklanjanje višestruke pojave jedinki u populaciji. Pošto su se 3. i 4. jedinka već pojavile u populaciji (kopije 2. i 1. jedinke respektivno) njihove prilagođenosti će biti 0. Zbog toga će prilagođenost jedinki u populaciji biti redom 0.2, 0.3, 0, 0, 0.7 .

Nema nikavog praktičnog razloga koji bi išao u prilog ostavljanja višestruke pojave jedinki u populaciji, jer pri dobrom izboru primenjenih genetskih operatora i ostalih važnih aspekata za primenu GA, dobre jedinke svoju namenu uspešno vrše i ako se u populaciji pojavljuju pojedinačno. Pošto se višestruka pojava jedinki implicitno uklanja vrlo efikasno i brzo, može se primenjivati u skoro svim slučajevima. Primenom ove tehnike se u praktičnim primenama dobijaju znatno bolji rezultati u odnosu na osnovnu varijantu, što u potpunosti odgovara gore pomenutim očekivanim efektima. Pošto ovaj pristup ne govori ništa o jedinkama čiji se genetski kodovi u populaciji ne ponavljaju, nije predviđen za samostalnu primenu u računanju funkcije prilagođenosti, već isključivo u kombinaciji sa nekom od ostalih tehnika.

2.3.3 Politika zamene generacija

U GANP implementaciji je dato nekoliko politika zamene generacija, od kojih se neke mogu i kombinovati:

- Generacijski (generational) GA;
- Stacionarni (steady-state) GA;
- Elitistička strategija (elitist strategy).

Odgovarajuće politike zamene generacija nisu implementirane konkretnim funkcijama, već od vrednosti datih parametara zavisi koja je varijanta izabrana. Parametri od važnosti za politiku zamene generacija pripadaju GA strukturi, i to:

- Podstruktura za selekciju: broj jedinki koje su direktno izabrane, pa se na njih ne primenjuje operator selekcije (N_{pass});
- Podstruktura za politiku zamene generacija: broj elitnih jedinki, na koje se ne primenjuju genetski operatori, već direktno prelaze u narednu generaciju (N_{elite}).

Pri ovakvom pristupu se u narednu generaciju direktno bira najboljih N_{elite} + N_{pass} jedinki. Pošto je jedna kopija svake od tih jedinki direktno prošla u narednu generaciju, a zatim još i ravnopravno učestvuju u selekciji preostalog dela populacije u narednoj generaciji, one bi bile neopravdano privilegovane u odnosu na ostale jedinke u populaciji. Šanse ostalih jedinki da budu izabrane u narednu generaciju bi time bile svedene na minimum, i one bi izgubile svaki značaj tokom izvršavanja GA.

Da bi se izbegla ova anomalija, posle direknog izbora ”privilegovanih” jedinki u narednu generaciju, vrši se popravka (smanjivanje) njihovih prilagođenosti pre izbora jedinki za preostali deo populacije u sledećoj generaciji. Popravka prilagođenosti "privilegovanih" jedinki se vrši po formuli (2.9).

$$\text{(novo) } f_i = \begin{cases} f_i - \bar{f}, & f_i > \bar{f} \\ 0, & f_i \le \bar{f} \end{cases} \qquad 1 \le i \le N_{elite} + N_{pass} \tag{2.9}$$

$$\bar{f} = \frac{1}{N_{pop}} \cdot \sum_{i=1}^{N_{pop}} f_i \tag{2.10}$$

Vrednost $\bar{f}$ data u (2.10) je srednja vrednost prilagođenosti svih jedinki u populaciji pre popravke. Ovakav pristup obezbeđuje ravnopravniji tretman svih jedinki u populaciji, jer se prilagođenost ”privilegovanih” jedinki smanjuje za onu vrednost kojom su ”nagrađene” direktnim izborom u narednu generaciju. Time i nešto slabije jedinke, koje mogu sadržati dobre gene, čuvaju svoje šanse pri primeni operatora selekcije.

2.3.3.1 Generacijski GA

Generacijski GA je model u kome se na sve jedinke u populaciji primenjuju genetski operatori, što predstavlja slučaj ako se oba gore pomenuta parametra postave na nultu vrednost (N_{elite} = 0, N_{pass} = 0).

Ovaj model najviše odgovara teorijskim aspektima GA (teorema o shemama i hipoteza o gradivnim blokovima). I u praktičnim aspektima GA, ovakav pristup ima nekih prednosti. Na primer, moć pretrage i selekcioni pritisak su veći u odnosu na ostale pristupe jer sve jedinke aktivno učestvuju pri izvršavanju genetskog algoritma i potpuno su ravnopravne.

Međutim, najveći nedostatak ovakvog pristupa je mogućnost gubljenja dobrih gena ili jedinki nekom ”nesretnom” primenom genetskih operatora. Zbog toga veliki selekcioni pritisak može da anulira dobru moć pretrage jer se stalno mogu naizmenično dobijati i gubiti ista rešenja. Dakle, ovakva politika zamene generacija ne mora da predstavlja i najbolji pristup u primeni GA.

Pri rešavanju nekih problema prethodno opisana anomalija ne dolazi do izražaja, pa se mogu dobiti i rešenja odličnog kvaliteta. Međutim, takvih problema je relativno malo, pa se sugeriše primena ovakvog modela samo kao pomoćnog u izuzetnim slučajevima, a da se kao osnovni model primeni neka od ostalih politika zamene generacija.

2.3.3.2 Stacionarni GA

Ovakav pristup favorizuje jedan deo generacije, pa određeni broj najboljih jedinki (N_{pass}) ne prolazi kroz selekciju, već je direktno izabran. Imajući u vidu

vrednost ovog parametra, operator selekcije automatski vrši izbor jedinki za narednu generaciju samo na preostala slobodna mesta (N_{pop} - N_{pass}).

Ovakav pristup smanjuje moć pretrage, ali se istovremeno smanjuje i selekcioni pritisak, čime se čuva dobar genetski materijal. Time ne samo da se u velikoj meri izbegava mogućnost gubljenja dobrih jedinki, već se čuvaju i dobri geni, koji zatim uz pomoć operatora ukrštanja mogu dati još bolje jedinke.

2.3.3.3 Elitistička strategija

Korišćenjem ove politike zamene generacija najboljih N_{elite} jedinki direktno prolazi u narednu generaciju bez primene genetskih operatora selekcije, ukrštanja i mutacije. Pošto se na date jedinke ne primenjuju genetski operatori, njihove vrednosti se prenose iz tekuće u narednu generaciju, bez potrebe za ponovnim računanjem. Ovakav pristup čuva najbolje jedinke u potpunosti, ali se još više smanjuje moć pretrage GA.

Prednost ovakvog pristupa je u tome što je vrednosti elitnih jedinki potrebno izračunati samo u prvoj generaciji GA, a zatim se one prenose. Pošto u većini primena, a pogotovo u rešavanju NP-kompletnih problema, vrednosna funkcija zahteva najveći deo procesorskog vremena u odnosu na sve ostale delove GA, dodavanje većeg broja elitnih jedinki praktično vrlo malo utiče na povećanje vremena izvršavanja. Često je moguće da zbog keširanja vreme izvršavanja bude čak i kraće, kao što je moguće videti u primeru 5.3 .

Preporučuje se korišćenje većeg broja elitnih jedinki u populaciji pošto njihovo dodavanje nije vremenski zahtevno, a one doprinose čuvanju dobrih rešenja i njihovoj rekombinaciji za dobijanje još boljih rešenja od strane ostalog ”neprivilegovanog” dela populacije. Zbog toga ne samo što GA primenom elitističkih strategija dobija bolje rešenje, već je disperzija u kvalitetu rešenja mnogo manja, ako se genetski algoritam izvršava više puta. Time je i samo izvršavanje stabilnije, a kvalitet rešenja se može bolje predvideti.

2.3.3.4 Stacionarni GA sa elitističkom strategijom

Moguća je i kombinacija stacionarnog GA sa elitističkom strategijom, gde se oba parametra N_{elite} i N_{pass} postavljaju na celobrojne pozitivne vrednosti. U tom slučaju se prvo razmatra N_{elite} najboljih jedinki koje kao ”elitne” prolaze bez primene genetskih operatora. Zatim od preostalih N_{pop} - N_{elite} jedinki u populaciji najboljih N_{pass} prolazi direktno samo selekciju, ali se na njima primenjuju operatori ukrštanja i mutacije. Na konačno preostala mesta u populaciji, kojih ima N_{pop} - N_{elite} - N_{pass} , se primenjuju svi genetski operatori. Ovakav pristup predstavlja kompromis između prethodnih strategija, što u nekim slučajevima može dati dobre rezultate.

2.4 Genetski operatori

Osim prethodno opisanih važnih aspekata za uspešnu primenu GA, kao što su kodiranje, funkcija prilagođenosti i politika zamene generacija, vrlo su značajni i genetski operatori. Pri rešavanju praktičnih problema veće dimenzije, dobro podešavanje genetskih operatora je presudan faktor za kvalitet dobijenih rešenja.

U ovoj implementaciji su direktno realizovani osnovni genetski operatori: selekcija, ukrštanje i mutacija. Pošto je pojava ostalih genetskih operatora rudimentarna, njihova primena vrlo retka, a i najčešće se mogu prikazati kao

specijalan slučaj osnovnih genetskih operatora, oni nisu direktno realizovani, ali je njihova primena moguća.

2.4.1 Selekcija

Podržani su sledeći operatori selekcije opšte namene:

- Prosta rulet selekcija;
- Selekcija zasnovana na rangu;
- Turnirska selekcija i fino gradirana turnirska selekcija;
- Selekcija primenom ostataka.

Za neke od tih operatora selekcije realizovane su i razne podvarijante. Za rešavanje problema u ovom radu je korišćena selekcija zasnovana na rangu, jer je ona dala najbolje rezultate na svakom od njih (SPLP, UNDP i ISP), a eksperimentisano je i sa ostalim operatorima selekcije.

2.4.1.1 Prosta rulet selekcija

Primer 2.2. Neka su jedinke predstavljene binarnim nizom dužine 2, i neka su vrednosti jedinki date sa

00 0.2
01 0.3
10 0.5
11 0.7

Neka populacija u nekoj generaciji sadrži sledeće jedinke: **00, 01, 01, 10, 11**. Njihova ukupna vrednost je 0.2 + 0.3 + 0.3 + 0.5 + 0.7 = 2.0, pa ako primenimo najprostiju šemu preuzimanja vrednosti, to su ujedno i prilagođenosti jedinki u populaciji. Prema tome očekivane učestanosti datih jedinki u sledećoj generaciji su redom jednake 5 * 0.2 / 2.0 = 0.5; 5 * 0.3 / 2.0 = 0.75; 5 * 0.3 / 2.0 = 0.75; 5 * 0.5 / 2.0 = 1.25; 5 * 0.7 / 2.0 = 1.75 . Ove učestanosti možemo nazvati i *normalizovanim* prilagođenostima jedinki u populaciji jer sa takvim šansama (širinama polja) one učestvuju na ruletu, gde se bira sledeća generacija.

Primer 2.3. Neka su jedinke kao i u primeru 2.2 predstavljene binarnim nizom dužine 2, ali sa drugačijim vrednostima jedinki:

00 0.2
01 0.3
10 8.0
11 9.0

Ako su u populaciji tokom neke generacije date jedinke **00, 00, 01, 01, 10**; tada je ukupna vrednost jedinki jednaka 0.2 + 0.2 + 0.3 + 0.3 + 8.0 = 9.0 . Normalizovana prilagođenost jedinki u populaciji je tada jednaka nizu 5 * 0.2 / 9.0 = 0.11; 5 * 0.2 / 9.0 = 0.11; 5 * 0.3 / 9.0 = 0.17; 5 * 0.3 / 9.0 = 0.17; 5 * 8.0 / 9.0 = 4.44, što su i širine odgovarajućih polja na ruletu. Primetimo da u ovom slučaju jedinka čiji je genetski kod **10**, iako nije globalno optimalna, dominira populacijom, jer ima značajno veću prilagođenost od zbira prilagođenosti svih ostalih jedinki u populaciji. Tada se u narednoj generaciji ostale jedinke izbacuju iz populacije, što dovodi do praktične nemogućnosti dostizanja globalnog optimuma (jedinka sa kodom 11), odnosno do pojave tzv. preuranjene konvergencije.

2.4.1.2 Selekcija zasnovana na rangu

Za prevazilaženje anomalija proste selekcije je vrlo pogodna selekcija zasnovana na rangu (rank based selection). Unapred formiramo niz rangova, za odgovarajuće pozicije jedinki u populaciji. Funkcija prilagođenosti svake jedinke je vrednost ranga date jedinke. Pri tome funkcija prilagođenosti jedinki u populaciji zavisi samo od poretka jedinki u populaciji, kao što se može videti u primeru 2.3, a ne i od konkretnih vrednosti jedinki, pa se preuranjena konvergencija retko javlja.

Primer 2.4. Neka su date jedinke iz primera 2.3, pri čemu je niz rangova redom 2.5, 2, 1.8, 1.5, 1.2 . Tada će normalizovane funkcije prilagođenosti jedinki 00, 00, 01, 01, 10 biti 1.5, 1.2, 2, 1.8, 2.5, pa će u narednoj generaciji bar 3 jedinke (očekivani broj jedinki je 5*(1.5+1.2+2+1.8) / (1.5+1.2+2+1.8+2.5) = 5*6.5/9 =3.6) biti iz skupa {00,01}. To ostavlja realnu mogućnost da se ukrštanjem 10 i 01 na prvoj poziciji dobije optimalno rešenje (jedinka 11). Ovim načinom selekcije jedinka čiji je genetski kod 10, iako po funkciji prilagođenosti trenutno daleko najbolja u populaciji, ipak daje šansu i ostalim jedinkama sa dobrim genima (jedinka čiji je kod 01 sa genom 1 na drugoj poziciji).

U primeni genetskih algoritama na problemima opisanim u ovom radu, eksperimentisanjem su najbolji rezultati dobijeni primenom selekcije zasnovane na rangu. Ovim načinom selekcije se može zadati proizvoljni nenegativni niz vrednosti rangova. Za svaki pojedinačni problem, se eksperimentalno određuje niz rangova, koji predstavlja dobar kompromis između elemenata istraživanja i iskorišćenja u primeni GA.

2.4.1.3 Turnirska selekcija i fino gradirana turnirska selekcija

U slučaju turnirske selekcije (tournament selection) za unapred dati broj N_{tourn}, na slučajan način se generiše N_{sel} skupova od po N_{tourn} jedinki. U svakom od skupova jedinki se simulira "turnir", i u narednu generaciju prolazi "pobednik turnira", odnosno jedinka sa najboljom funkcijom prilagođenosti u datom skupu.

Primer 2.5. Za jedinke u primeru 2.2 primenimo turnirsku selekciju sa N_{tourn} = 2 i N_{sel} = 5. Ako su slučajno izabrani skupovi {00,01}; {00,00}; {01,10}; {01,00}; {10,10} tada je populacija u narednoj generaciji 01, 00, 10, 01, 10, što je znatno bolje u odnosu na prostu (rulet) selekciju.

Fino gradirana turnirska selekcija je unapređenje osnovne turnirske selekcije, u slučajevima kada nije pogodno da broj N_{tourn} bude ceo broj. Na primer ako je N_{tourn} = 2 previše mala, a N_{tourn} = 3 prevelika vrednost, tada je pogodno izabrati $N_{tourn} \in [2, 3]$. Detaljnije informacije o fino gradiranoj turnirskoj selekciji mogu se videti u [Fil98].

2.4.1.4 Selekcija primenom ostataka

U selekciji primenom ostataka (stohastic remainder selection) se celobrojni deo očekivanog broja potomaka date jedinke (broj jedinki u populaciji * prilagođenost jedinke / zbir prilagođenosti svih jedinki) direktno izabere u narednu generaciju, a razlomljeni deo se bira prostom rulet selekcijom. Time se smanjuje nedeterminizam u selekciji, ali ovaj metod selekcije i dalje poseduje nedostatke proste selekcije. U nekim aspektima su nedostaci čak i povećani u odnosu na prostu selekciju.

Primer 2.6. U populaciji iz primera 2.2, u novoj generaciji je direktno izabrano četiri kopije jedinke sa genetskim kodom 10 (prilagođenost te jedinke je 5 * 8.0 / 9.0 = 4.444). Preostalo jedno mesto se bira prostom selekcijom sa verovatnoćama jednakim redom 5 * 0.2 / 9.0 = 0.11; 5 * 0.2 / 9.0 = 0.11; 5 * 0.3 / 9.0 = 0.17; 5 * 0.3 / 9.0 = 0.17; 5 * 8.0 / 9.0 - 4.0 = 0.444 .

Detaljnije o datim operatorima selekcije se može videti u [BeD93b], [Sri94] i [Kra97a].

2.4.2 Ukrštanje

Implementirani su operatori za jednopoziciono (one-point crossover), dvopoziciono (two-point crossover) i uniformno ukrštanje (uniform crossover). Osim njih, eksperimentisano je i sa nekim drugim operatorima ukrštanja, ali oni nisu pokazali očekivane rezultate, pa nisu uvršteni u GANP implementaciju.

Izbor odgovarajućeg operatora ukrštanja zavisi od prirode problema koji rešavamo, i od načina kodiranja datog problema. U slučajevima sa velikom međuzavisnošću gena u genetskom kodu obično se koristi jednopoziciono ukrštanje, jer se njime vrši najmanje razbijanje celine jedinke. Uniformno ukrštanje je, nasuprot tome, najčešće pogodno primeniti kada je međuzavisnost gena u genetskom kodu mala, pa nešto veće razbijanje celine jedinke nema mnogo uticaja na performanse GA.

Takođe je podržana i mogućnost primene specijalnih načina ukrštanja koji su projektovani samo za određenu vrstu problema, ili čak i samo za neki konkretan problem. Međutim takvi operatori nisu opšteg tipa, pa nisu smešteni u zajednički deo GANP implementacije, već u onaj koji zavisi od prirode konkretnog problema.

2.4.2.1 Jednopoziciono ukrštanje

Pri jednopozicionom ukrštanju na početku se bira $N_{pop}/2$ parova jedinki (N_{pop} je veličina populacije). Zatim se za svaki par jedinki, sa unapred zadatom verovatnoćom (nivoom) ukrštanja, na slučajan način bira pozicija za ukrštanje i vrši razmena bitova genetskih kodova datih parova jedinki posle date pozicije. Shematski ga možemo prikazati:

pre ukrštanja	***posle ukrštanja***
XXX | XXXXX	XXX | **YYYYY**
YYY | **YYYYY**	**YYY** | XXXXX

Primer 2.7. Neka je dat nivo ukrštanja p_{cross} = 0.85, i populacija od 5 jedinki 01011, 10001, 00100, 10100, 11001. Neka su izabrani parovi jedinki (prva, peta) i (druga, četvrta), i neka se u oba para vrši ukrštanje. Ako je pozicija za ukrštanje prvog para 2 a drugog para 4, pa se tada ukrštanjem dobijaju sledeće nove jedinke:

prvi par jedinki (prva i peta)		***drugi par jedinki (druga i četvrta)***	
pre ukrštanja	*------> posle ukrštanja*	*pre ukrštanja*	*------> posle ukrštanja*
0 1|0 1 1	**0 1 0 0 1**	**1 0 0 0|1**	**1 0 0 0 0**
1 1|0 0 1	**1 1 0 1 1**	**1 0 1 0|0**	**1 0 1 0 1**

Posle ukrštanja populacija će izgledati: **01001, 10000, 00100, 10101, 11011**

Iako je teorijski najbolje, odnosno najbolje odgovara hipotezi o gradivnim blokovima, jednopoziciono ukrštanje u praksi ima neke manje nedostatke. Na primer, prvi bit u genetskom kodu jedinke neće biti razmenjen skoro nikad, a poslednji će biti razmenjen u skoro svakom ukrštanju. To nije pogodno ako su prvi i poslednji bit jedinke u uzajamnoj vezi, jer će tada data veza biti prekinuta, pa prvi bit ostaje, a poslednji se razmenjuje.

2.4.2.2 Dvopoziciono ukrštanje

Kao i pri jednopozicionom ukrštanju na početku se bira $N_{pop}/2$ parova jedinki. Za razliku od njega se zatim za svaki par jedinki, sa unapred zadatim nivoom ukrštanja, na slučajan način biraju 2 pozicije i vrši razmena bitova genetskih kodova datih parova jedinki između datih pozicija. Shematski ga možemo prikazati:

pre ukrštanja	***posle ukrštanja***
XXX | XXXXX | XXXX	XXX | **YYYYY** | XXXX
YYY | **YYYYY** | **YYYY**	**YYY** | XXXXX | **YYYY**

Primer 2.8. Neka je dat nivo ukrštanja p_{cross} = 0.85, i populacija od 5 jedinki 01011, 10001, 00100, 10100, 11001. Neka su izabrani parovi jedinki (prva, peta) i (druga, četvrta), i neka se u oba para vrši ukrštanje. Neka su pozicije redom 2,4 i 0,3 pa je tada:

prvi par jedinki (prva i peta)		***drugi par jedinki (druga i četvrta)***	
pre ukrštanja	*------> posle ukrštanja*	*pre ukrštanja*	*------> posle ukrštanja*
0 1|0 1|1	**0 1 0 0 1**	**|1 0 0|0 1**	**1 0 1 0 1**
1 1|0 0|1	**1 1 0 1 1**	**|1 0 1|0 0**	**1 0 0 0 0**

Posle ukrštanja populacija će izgledati: **01001, 10101, 00100, 11011, 10000**

2.4.2.3 Uniformno ukrštanje

Pri uniformnom ukrštanju (videti [Sys89]), za svaki par jedinki koje se ukrštaju, se na slučajan način generiše maska. Maska je binarni niz dužine N_{bits}, gde je N_{bits} dužina genetskog koda jedinki. Ukoliko je na nekoj poziciji vrednost maske 1, prvi potomak uzima dati bit (gen) iz prve, a drugi potomak iz druge jedinke. Ako je vrednost maske 0, suprotno, prvi potomak preuzima bit (gen) iz druge, a drugi potomak iz prve jedinke. Na primer:

maska
101101001001
pre ukrštanja
XXXXXXXXXXXX
YYYYYYYYYYYY
posle ukrštanja
X**Y**XX**Y**X**YY**X**YY**X
YX**YY**X**Y**XX**Y**XX**Y**

Primer 2.9. Neka je dat nivo ukrštanja p_{cross} = 0.45, a učestanost razmene bitova uniformnog ukrštanja p_{unif} = 0.6 i populacija od 5 jedinki 01011, 10001, 10100, 10100, 11001. Neka su izabrani parovi jedinki (prva, treća) i (četvrta,

peta), i neka se u prvom paru vrši ukrštanje, a u drugom ne. Pretpostavimo da je za prvi par maska 11001, tada je:

prvi par jedinki (prva i treća)
maska
1 1 0 0 1

pre ukrštanja	------>	*posle ukrštanja*
0 1 0 1 1		**0 1 1 0 1**
1 0 1 0 0		**1 0 0 1 0**

Posle ukrštanja populacija će izgledati: **01101, 10001, 10010, 10100, 10100** . U [Spe91] se mogu naći još neke dodatne informacije oko uniformnog ukrštanja.

2.4.3 Mutacija

U ovoj impelementaciji je primenjen jedinstven koncept operatora mutacije, samo je on realizovan na različite načine. U zavisnosti od oblasti primene, neke realizacije su efikasniji od drugih u određenim slučajevima. Realizovani su sledeći operatori mutacije: prosta mutacija, mutacija pomoću binomne raspodele i mutacija korišćenjem normalne raspodele. Kao i u slučaju operatora ukrštanja, ostavljena je mogućnost primene i operatora mutacije zavisnih od prirode samog problema.

2.4.3.1 Prosta mutacija

Ova realizacija doslovno prati definiciju operatora mutacije, gde se genetski kod procesira bit po bit, i za svaki bit se određuje da li je došlo do mutacije ili ne. Pri realizaciji se može proces nešto ubrzati, ako se za svaku reč u genetskom kodu formira maska, pa se mutacija vrši preko nje. Međutim, i pri takvom pristupu generisanje maske zahteva procesiranje bit po bit.

Primer 2.10. Neka je dat nivo mutacije p_{mut} = 0.2 i jedinka 010111000101. Ako je slučajnim izborom generisana maska 001000000100, tada je:

maska
0 0 1 0 0 0 0 0 0 1 0 0
jedinka pre mutacije
0 1 0 1 1 1 0 0 0 1 0 1
jedinka posle mutacije
0 1 1 1 1 1 0 0 0 0 0 1

Pošto je nivo mutacije uglavnom mali, i obično se kreće u opsegu [0.001, 0.01], broj mutiranih gena (bitova), u genetskom kodu jedinke, je obično mali, pa nije svrsishodno procesiranje celokupnog genetskog koda. Zbog toga su realizovane neke od ostalih metoda koje teže da procesiraju samo reči koje imaju mutirane gene, što je u većini slučajeva mnogo efikasnije.

Ovakav način ipak ima i svojih prednosti, jer su ostali pristupi realizaciji operatora mutacije zasnovani na određenim aproksimativnim svojstvima teorije verovatnoće, koja važe samo u slučaju velikog broja bitova. Zbog toga je ovaj pristup koristan i može se primenjivati u slučajevima kada su instance rešavanog problema male dimenzije, tako da je genetski kod jedinke kratak.

2.4.3.2 Mutacija pomoću binomne raspodele

Neka slučajna promenljiva X_{gene} uzima vrednost 1 ako je dati gen mutiran, a u suprotnom uzima vrednost 0. Tada odgovarajuća slučajna promenljiva X_{ind} odgovara celom genetskom kodu jedinke i označava broj mutiranih gena u njoj. Pošto je X_{gene} prosta slučajna promenljiva, tada slučajna promenljiva X_{ind} ima binomnu raspodelu $B(N_{bits}, p_{mut})$.

Koristeći datu osobinu operator mutacije možemo implementirati relativno efikasno. Na slučajan način izaberemo broj $s \in [0,1]$, a zatim nađemo broj N_{mut} takav da važi formula (2.11).

$$F(N_{mut}) \leq s < F(N_{mut} + 1) \qquad (2.11)$$

$$F(n) = \sum_{k=1}^{n} \binom{n}{k} \cdot p^k \cdot q^{n-k}$$

$F(n)$ je kumulativna funkcija raspodele za datu binomnu raspodelu, a dobijena vrednost N_{mut} predstavlja broj gena koje treba mutirati u datom genetskom kodu jedinke. Zatim se na slučajan način biraju pozicije gena za mutiranje i nad tim genima vrše mutacije.

Za realizaciju mutacije na ovakav način, potrebno je N_{mut} koraka za računanje prve aproksimacije za $F^{-1}(s)$, i još N_{mut} koraka za mutiranje odgovarajućih gena. Dakle potrebno je $O(N_{mut})$ koraka, za razliku od $O(N_{bits})$ koraka kod prethodnog načina. Ovakav pristup je i u praksi dosta brži od prethodnog, u slučajevima kada je N_{bits} dovoljno veliko ($N_{bits} \geq 20$) a p_{mut} dovoljno malo ($p_{mut} \leq 0.05$), što se dešava u velikom broju slučajeva primene GA.

Pokazalo se da data realizacija operatora mutacije nije pogodna ako je broj gena u genetskom kodu jedinke veliki ($N_{bits} > 1000$), pa se tada preporučuje primena nekog drugog pristupa. Data anomalija se javlja kao posledica računske greške kod izračunavanja sabiraka kumulativne funkcije raspodele, pa broj N_{mut} nije tačno izračunat.

2.4.3.3 Mutacija korišćenjem normalne raspodele

U slučaju kada N_{bits} --> ∞ slučajna promenljiva X_{ind} teži normalnoj raspodeli $N(N_{bits} * p_{mut} ; N_{bits} * p_{mut} * (1-p_{mut}))$. Zbog toga, kada je N_{bits} dovoljno veliko a p_{mut} dovoljno malo, slučajnu promenljivu X_{ind} možemo aproksimirati datom normalnom raspodelom.

Kao i kod prethodnog načina izračunavanja, prvo se računa broj gena koje treba mutirati N_{mut}, a zatim se biraju na slučajan način pozicije i dati geni se mutiraju. Za izračunavanje broja N_{mut} se koristi aproksimacija funkcije $\Phi^{-1}(x)$, gde funkcija $\Phi(x)$ definiše normalnu raspodelu $N(0, 1)$. Pri tome se u datoteci tablično memorišu vrednosti funkcije $\Phi^{-1}(x)$, koje se učitavaju u fazi inicijalizacije programa, a na osnovu njih se broj N_{mut} direktno nalazi.

Aproksimacija slučajne promenljive X_{ind} normalnom raspodelom se pokazala uspešnom u praksi ukoliko je $N_{bits} * p_{mut} \geq 5$, inače se ne preporučuje, jer rezultati aproksimacije suviše odstupaju od originalnih vrednosti.

Pošto izračunavanje broja N_{mut} na ovaj način zahteva konstantno O(1) vreme izvršavanja, i pošto ne postoji gornje ograničenje za broj bitova N_{bits} (kao kod prethodnog načina), realizovana je i varijanta ovog operatora mutacije koji

operiše nad celom populacijom, umesto nad pojedinačnim jedinkama. Tada se računa ukupan broj gena koje treba mutirati u celoj populaciji, zatim se na slučajan način biraju globalne pozicije mutiranih gena, na osnovu kojih se izračunavaju redni brojevi jedinki i lokalne pozicije mutiranih gena. Pošto je u tom slučaju uslov $N_{pop} * N_{bits} * p_{mut} \geq 5$ gotovo uvek ispunjen, ovakav pristup je generalno primenjiv.

2.5 Ostale funkcije

Osim kodiranja, politike zamene generacija i genetskih operatora, postoje još neki aspekti koji mogu uticati na performanse genetskog algoritma ili doprineti na neki drugi način. To su: kriterijum završetka izvršavanja GA, štampanje izveštaja u toku i na kraju izvršavanja GA, parametri GA, keširanje GA i generator slučajnih brojeva dat u izvornom kodu. Keširanje GA kao tehnika za poboljšavanje vremena izvršavanja GA biće detaljno opisano u poglavlju 4, dok su ostale funkcije opisane u ovom odeljku.

2.5.1 Kriterijumi završetka izvršavanja GA

GANP implementacija sadrži nekoliko kriterijuma završetka izvršavanja GA, koji se mogu i proizvoljno kombinovati:

- Dostignut je maksimalan broj generacija N_{gener};
- Najbolja jedinka je ponovljena N_{rep} puta;
- Sličnost jedinki u generaciji je veća od zadate granice;
- Dokazana je optimalnost najbolje jedinke, ukoliko je funkcija za dokazivanje optimalnosti moguća i realizovana u delu koji zavisi od prirode samog problema;
- Najbolja jedinka je dostigla optimalno rešenje, ukoliko je ono unapred poznato;
- Izvršavanje genetskog algoritma je prekinuto od strane korisnika.

Ukoliko je ispunjen izabrani kriterijum za završetak, izvršavanje genetskog algoritma se regularno prekida i štampa se odgovarajući izveštaj, koji sadrži najbolje dobijeno rešenje datog problema (najbolja jedinka u poslednjoj generaciji).

2.5.1.1 Maksimalan broj generacija

Ovo je najprostiji kriterijum za završetak izvršavanja genetskog algoritma, jer se GA prekida kada broj generacija dostigne N_{gener} koji je unapred zadat. Pošto je predviđeno da izvršavanje GA bude završeno u nekom konačnom (realno dostižnom) vremenu. Ovaj kriterijum se koristi skoro u svakoj primeni GA, ređe samostalno, najčešće u kombinaciji sa drugim kriterijumima.

Parametar N_{gener} treba da obezbedi dovoljan broj generacija za izvršavanje GA tako da rešenje bude kvalitetno, a da vreme izvršavanja ne bude nepotrebno dugo. Najnepovoljnija mogućnost je ako genetski algoritam dostigne optimalno rešenje, ali zbog lošeg izbora parametra N_{gener} koji ima suviše veliku vrednost, nepotrebno nastavi sa radom i izvrši još nekoliko stotina ili hiljada generacija. U tom slučaju nije dobro ni ishitreno smanjiti broj N_{gener} jer se može dobiti rešenje lošeg kvaliteta pri izvršavanju na nekoj drugoj instanci datog problema, slične dimenzije, ali različitih karakteristika. Obično nije lako unapred odrediti pogodnu vrednost za N_{gener} pogotovo u slučaju kada se dati

kriterijum samostalno primenjuje, jer se ovaj nedostatak najčešće i javlja baš u tom slučaju. Zbog toga je najbolje ovaj kriterijum primeniti, ali u kombinaciji sa nekim drugim kriterijumima, koji na neki način ocenjuju kvalitet dobijenog rešenja.

2.5.1.2 Ponavljanje najbolje jedinke

Ponekad može biti korisno ako se ograniči broj ponavljanja najbolje jedinke tokom generacija, jer veliki broj ponavljanja najbolje jedinke obično znači da je genetski algoritam konvergirao, pa tada dalje izvršavanje nema mnogo smisla. Takav pristup je pogodan u slučaju kada smo dostigli globalno optimalno rešenje. Međutim moguć je i slučaj dostizanja lokalnog optimuma, koji nije lako napustiti, ali bi GA možda u daljem izvršavanju dostigao i globalni optimum.

Ovaj kriterijum, ipak, može da oceni ponašanje populacije tokom generacija, pa time i da grubo proceni kvalitet dobijenog rešenja. Iako je takva procena često loša, primenljiva je, za razliku od prethodnog kriterijuma. Da bi se smanjio uticaj eventualne loše procene kvaliteta dobijenog rešenja, preporučuje se kombinovanje ovog kriterijuma sa nekim od ostalih.

2.5.1.3 Sličnost jedinki u populaciji

Ovaj kriterijum prekida izvršavanje GA ukoliko je sličnost jedinki u populaciji veća od unapred dozvoljene. Kada jedinke u populaciji postanu suviše slične, to je dobar pokazatelj da je konvergencija GA završena. On tada više nije sposoban da poboljšava dobijeno rešenje u narednim generacijama, jer je raznovrsnost genetskog materijala vrlo mala, pa genetski operatori nemaju efekta. Ako je dostignuto optimalno rešenje, tada poboljšavanje rešenja više ni teorijski nije moguće, pa je ovakav ishod očekivan. Međutim, i kada nije dostignuto optimalno rešenje, i teorijski je moguće poboljšavanje, genetski algoritam više nema praktičnih mogućnosti za takvu alternativu, pa više nema nikakvih razloga za njegovo izvršavanje.

Sličnost jedinki u populaciji je generalno vrlo dobar kriterijum za završetak izvršavanja GA, ako je dobro projektovana funkcija za merenje sličnosti. U ovoj implementaciji je funkcija za merenje sličnosti jedinki u populaciji zasnovana na broju istih jedinki, a detaljnije se može videti u dodatku B. Na žalost ovako realizovana funkcija nema smisla ukoliko je primenjeno implicitno uklanjanje jedinki koje se višestruko pojavljuju u populaciji.

2.5.1.4 Prekid od strane korisnika

Operativni sistem dozvoljava prekid bilo kog programa u proizvoljnom trenutku, ali u tom slučaju nije moguće dobiti rezultate dotadašnjeg izvršavanja. Zbog toga je projektovana jednostavna funkcija koja na osnovu zahteva korisnika prekida izvršavanje GA po završetku tekuće generacije, i štampa sve dobijene rezultate na standardan način. Ovaj kriterijum nije mnogo pogodan za opštu primenu, jer zahteva intervenciju korisnika, ali ponekad može biti koristan pri testiranju nekih karakteristika GA.

2.5.1.5 Kombinovanje više kriterijuma

Kao što je napomenuto svaki od prethodnih kriterijuma ima neke prednosti i određene nedostatke, pa u većini slučajeva nisu pogodni za pojedinačnu primenu. Zbog toga se najčešće primenjuju kombinacije više kriterijuma pa se na taj način smanjuje mogućnost loše procene o prekidu izvršavanju GA.

2.5.1.6 Neregularan završetak izvršavanja

Osim gore navedenih kriterijuma moguće je i da se izvršavanje GA završi na neki neregularan način, kada je ispunjen neki od sledećih uslova:

- Neke od izabranih funkcija nisu međusobno kompatibilne;
- Utvrđeno je da neka od kontrolisanih promenljivih u datom momentu ima vrednost izvan dozvoljenog opsega;
- Sve jedinke u populaciji su nekorektne, a nije definisano njihovo uključivanje i korišćenje u GA.

U slučaju neregularnog prekida izvršavanja GA, štampa se izveštaj o razlogu prekida, a najbolje dobijeno rešenje se štampa samo u onim slučajevima kada to ima smisla.

2.5.2 Promena parametara tokom izvršavanja

U ovoj implementaciji je dozvoljeno da nivo ukrštanja, odnosno mutacije, može da:

- bude konstantan tokom izvršavanja GA;
- se menja statički tokom generacija GA po unapred zadatoj formuli;
- se menja dinamički za svaku generaciju genetskog algoritma posebno, u zavisnosti od zadatih karakteristika populacije u datoj generaciji.

U dosta primena GA za rešavanje praktičnih NP-kompletnih problema, testiranjem se ne mogu uočiti razlike između datih pristupa, pa se primenjuje konstantan nivo ukrštanja ili mutacije tokom generacija GA, zato što je on najjednostavniji. Takođe je najlakše praktično odrediti, na osnovu eksperimenata, optimalan nivo datih genetskih operatora, ako je on na konstantnom nivou. Dati proces bi se dodatno otežao ukoliko nivo nije konstantan, već se menja tokom generacija.

Nasuprot tome, u nekim slučajevima se može odmah primetiti razlika u kvalitetu dobijenog rešenja i performansama GA, pri poređenju konstantnog i promenljivog nivoa datog genetskog operatora. Tada ima smisla eksperimentalno određivanje načina promene i ostalih karakteristika datog operatora, jer je dobitak značajan i možda se takav pristup može primeniti i na rešavanje nekih sličnih problema.

Od načina statičke promene nivoa mutacije (ukrštanja) načešće se koristi eksponencijalna promena tokom generacija. Jedino što se u nekim primenama bolje pokazao eksponencijalni porast, a u drugim eksponencijalno opadanje nivoa mutacije.

Za korišćenje dinamičke promene nivoa mutacije, u praksi je pokazala dobre rezultate funkcija koja izračunava sličnost jedinki u populaciji, definisana u prethodnom odeljku. Nivo mutacije u svakoj generaciji je jednak datoj sličnosti jedinki u populaciji koja je pomnožena konstantnim koeficijentom zadatim unapred iz konfiguracione datoteke. Napomenimo da ovakav pristup ne daje rezultate, ukoliko se primeni na operator ukrštanja, već se kod njega mora primeniti drugačije definisana dinamička promena tokom generacija.

Promena ostalih parametara ne utiče toliko na performanse GA, pa nije eksplicitno predviđeno da se oni mogu menjati tokom generacija. Međutim, ukoliko bi to bilo potrebno, lako bi se mogla realizovati takva mogućnost.

2.5.3 Štampanje izveštaja

Realizovane funkcije za štampanje izveštaja u toku i na kraju izvršavanja GA se razlikuju samo po broju odštampanih podataka. Tako se dobijaju izveštaji raznih formi, od najsažetijih koji se koriste u slučajevima kada je potreban samo rezultat i najosnovniji prateći podaci, do najdetaljnijih kada se štampaju svi relevantni podaci o izvršavanju GA.

U svakoj od generacija mogu biti prikazati sledeći podaci:

- Redni broj tekuće generacije;
- Genetski kod najbolje jedinke ili njegov deo;
- Vrednosti nekih jedinki (obično najbolje jedinke) ili vrednosti svih jedinki u populaciji;
- U koliko poslednjih generacija je ponovljena najbolja jedinka;
- Sličnost jedinki u populaciji tokom tekuće generacije;
- Koliko je odstupanje najbolje jedinke od optimalnog rešenja, ukoliko je ono unapred poznato.

Izveštaj na kraju izvršavanja GA može sadržati sledeće informacije:

- Argumente komandne linije;
- Datum i vreme kada je izvršavanje obavljeno;
- Ukupan broj generacija i vreme izvršavanja;
- Razlog završetka GA;
- Koliko je puta, brojčano i procentualno, bilo uspešno keširanje GA za izračunavanje vrednosti jedinki;
- Dobijeno rešenje, odnosno vrednost najbolje jedinke u poslednjoj generaciji;
- Optimalno rešenje za datu instancu, kao i odstupanje dobijenog rešenja od optimalnog, ako je optimalno rešenje unapred poznato;
- Ukupan broj jedinki u populaciji, kao i broj ”elitnih” i povlašćenih jedinki u populaciji;
- Način na koji je inicijalizovan generator slučajnih brojeva, kao i odgovarajuća vrednost;
- Veličina keš memorije i tip keširanja GA koji je primenjen;
- Operator selekcije koji je primenjen, kao i njegovi prateći parametri;
- Vrsta i nivo ukrštanja, kao i ostali podaci o njemu;
- Primenjeni operator mutacije, njen nivo, i način promene tokom generacija;

Napomenimo da ove funkcije daju izveštaje samo o karakteristikama GA koji je primenjen, dok odgovarajuće izveštaje o detaljnijim karakteristikama rešenja konkretnog problema (vrednosti svih njegovih relevantnih promenljivih) štampaju odgovarajuće funkcije koje zavise od prirode samog problema.

2.5.4 Generator slučajnih brojeva

Predviđeno je da se ova implementacija izvršava pod različitim operativnim sistemima i da se prevodi različitim prevodiocima. Iako svaki C prevodilac ima ugrađen generator slučajnih brojeva, oni daju međusobno različite nizove slučajnih (u stvari pseudoslučajnih) brojeva. U nekim specijalnim slučajevima je radi potrebe testiranja određenih karakteristika GA, neophodno da ova implementacija daje uvek potpuno isti niz slučajnih brojeva za sve platforme. Pošto postojeće sistemski ugrađene funkcije nisu zadovoljavale taj kriterijum,

pristupilo se realizaciji generatora slučajnih brojeva. Realizovane su sledeće funkcije:

- Dobijanje slučajnih neoznačenih celih brojeva iz 16-bitnog i 32-bitnog opsega;
- Inicijalizacija generatora slučajnih brojeva nekom konstantom ili uz pomoć vremenske funkcije;
- Generisanje slučajnog celog označenog broja iz zadatog intervala;
- Simulacija prostog slučajnog događaja;
- Zaokruživanje realnog broja dvostruke preciznosti celim brojem.

Generisanje slučajnih brojeva iz 32-bitnog opsega je bilo potrebno zbog dobijanja početne populacije koje se najčešće vrši na slučajan način. Direktno se na slučajan način generiše svaka reč u genetskom kodu jedinki početne populacije. Ukoliko u budućnosti genetski kod bude podeljen na reči dužine veće od 32 bita, vrlo je lako promeniti odgovarajuću funkciju, tako da direktno inicijalizuje date reči na slučajan način u početnoj populaciji.

Generator slučajnih brojeva je moguće inicijalizovati uz pomoć sistemske funkcije koja generiše tačno vreme. Takav način je pogodan upotrebi GA, kada je bitno da podaci generisani na slučajan način budu raznovrsni što je moguće više. Na taj način se postiže maksimalna verodostojnost dobijenih rezultata. Ovakav pristup se ne preporučuje u toku pisanja programskog koda koji je specifičan za rešavani problem, njegovog ispravljanja i testiranja. U toj fazi je najbolje da se program ponaša deterministički, da bi što efikasnije mogli da kontrolišemo sva njegova stanja. Zbog toga se u datim slučajevima generator slučajnih brojeva inicijalizuje nekom konstantom, tako da izvršavanje programa bude determinističko. Dati pristup se koristi i za merenje performansi keširanja GA.

U nekim delovima GANP implementacije javlja se potreba za efikasnom realizacijom izbora kod prostog slučajnog događaja. Prost slučajni događaj ima samo dve mogu]nosti čije su verovatnoće p i $q = 1-p$ unapred poznate, a koristi se pri realizaciji genetskih operatora ukrštanja i mutacije kao i u nekim drugim aspektima GA.

2.6 Funkcije koje zavise od prirode problema

Osnovne funkcije zavisne od prirode samog problema koji se rešava su: inicijalizacija problema, učitavanje i štampanje podataka o samom problemu i vrednosna funkcija. Bez njih nije moguće izvršavanje genetskog algoritma, a dodeljuju se odgovarajućim funkcijskim pokazivačima u GA strukturi.

2.6.1 Učitavanje i štampanje podataka

Deo za učitavanje podataka uključuje:

- Učitavanje konfiguracione datoteke za dati problem (u slučajevima kada ona postoji);
- Ulaz podataka iz datoteke sa konkretnom instancom problema, i njihovo smeštanje u odgovarajuću (problem) strukturu;
- Alociranje dinamičkih struktura koje se koriste za smeštanje ulaznih podataka;
- Učitavanje podataka o optimalnom rešenju, ako je ono unapred dato.

Podaci koje sadrži izveštaj na kraju izvršavanja GA (detaljno opisani u odeljku 2.5.3), često nisu dovoljni za upoznavanje i detaljniju analizu dobijenog rešenja. To je omogućeno funkcijom za štampanje detaljnih informacija o dobijenom rešenju, koje dopunjuje izveštaj GA.

2.6.2 Inicijalizacija problema

Ova funkcija je zadužena za inicijalizaciju svih promenljivih čija početna vrednost nije već ranije postavljena od samog GA, jer zavise od prirode problema i konkretne njegove instance. To u opštem slučaju obuhvata sledeće elemente:

- Generisanje početne populacije, na slučajan način ili pomoću neke heuristike;
- Dinamičko alociranje preostalih pokazivačkih promenljivih u problem strukturi;
- Inicijalizacija podataka u GA strukturi koje zavise od prirode problema i preostalih parametara problema;
- Eventualno, uređivanje nekih podataka kako bi njihova kasnija obrada bila efikasnija.

2.6.3 Vrednosna funkcija

Vrednosna funkcija je u ovoj implementaciji podeljena na dva dela:

- Dekodiranje genetskog koda jedinke;
- Izračunavanje samog rešenja koje predstavlja vrednost date jedinke, na osnovu argumenata problema.

Ovakvim pristupom se različite realizacije datih delova (funkcija) mogu međusobno kombinovati, što doprinosi fleksibilnosti implementacije. Pošto u većini slučajeva izračunavanje vrednosti jedinke troši većinu procesorskog vremena, ovaj pristup može doprineti i boljoj efikasnosti cele implementacije. Zbog toga je potrebno posebnu pažnju posvetiti efikasnoj realizaciji ovih funkcija, kao i njihovom kombinovanju.

2.6.4 Dodatne funkcije

Pored prethodno opisanih funkcija koje su obavezne pri rešavanju konkretnih problema pomoću GA, može se javiti potreba i za još nekim dodatnim funkcijama. To mogu biti: heuristike za dobijanje jedinki početne populacije, heuristike za poboljšavanje jedinki u toku izvršavanja GA, konverzija argumenata problema u genetski kod jedinke, konfigurisanje parametara vezanih za sam problem i operatori ukrštanja i mutacije zavisni od prirode problema.

2.6.4.1 Heuristike za dobijanje jedinki početne populacije

U ovoj implementaciji je dozvoljeno korišćenje nekih heuristika za inicijalizaciju jedinki ili cele populacije u početnoj generaciji. Moguće je korišćenje i više različitih heuristika, od kojih svaka može imati više podvarijanti. Inicijalizacija pomoću heuristika obezbeđuje bolje početne performanse GA od inicijalizacije na slučajan način. Međutim, pri rešavanju NP kompletnih problema, opisanih u ovom radu, prednosti ovakve inicijalizacije se mogu primetiti samo u prvih nekoliko generacija, a kasnije se mogu dobiti čak i nešto lošiji rezultati od klasičnog pristupa. To se objašnjava gubitkom genetskog

materijala pri inicijalizaciji heuristikama, koji se kasnije može vrlo teško nadoknaditi.

2.6.4.2 Heuristike za poboljšavanje jedinki u toku izvršavanja GA

Osim korišćenja za inicijalizaciju početne populacije, heuristike se mogu koristiti i za poboljšavanje rešenja GA u svakoj generaciji. Moguće ih je primeniti na neku pojedinačnu jedinku (najčešće je to najbolja jedinka), na nekoliko jedinki ili na celu populaciju. U ovom radu je težište bilo na fleksibilnoj i efikasnoj implementaciji osnovnog genetskog algoritma, dok je hibridizacija GA sa nekim drugim metodama bila u drugom planu. Dati pristup je već pokazao neke dobre karakteristike i može se očekivati da hibridizacijom GA pomoću nekih boljih heuristika dobijemo rešenja koji prevazilaze rezultate osnovnog genetskog algoritma. Detaljnije informacije o jednom pokušaju hibridizacije prostog GA mogu se videti u [Kra98b].

2.6.4.3 Konverzija argumenata problema u genetski kod jedinke

Pošto je u ovoj implementaciji primenjen princip da se poboljšane jedinke po pravilu vraćaju nazad u populaciju, u slučaju korišćenja heuristika za poboljšavanje rešenja, potrebna je i funkcija koja konvertuje argumente rešavanog problema u genetski kod jedinke. Nju je u matematičkom smislu jednostavno definisati kao inverznu funkciju prvog dela vrednosne funkcije. Mogu se javiti određene teškoće pri realizaciji, pošto se zahteva da ova funkcija bude efikasna.

2.6.4.4 Konfigurisanje parametara problema

Ova funkcija (u slučajevima kada je potrebna) u potpunosti prati način konfigurisanja parametara samog GA, samo na nivou konkretnog problema. Odgovarajući parametri vezani za konkretan problem mogu biti:

- Podaci o pojedinačnim heuristikama za generisanje početne populacije sa pojedinačnim i ukupnim brojem njihovih podvarijanti;
- Parametri potrebni za izvršavanje heuristika za poboljšavanje rešenja tokom izvršavanja GA;
- Informacije o imenu datoteke sa instancom rešavanog problema, datoteke sa eventualnim optimalnim rešenjem i sličnim informacijama.

2.6.4.5 Operatori ukrštanja i mutacije zavisni od prirode problema

U GA implementaciji ravnopravno mogu učestvovati i operatori ukrštanja i mutacije koji zavise od prirode problema. Oni, za razliku od operatora opisanih u odeljku 2.4, nisu opšteg tipa, već su primenljivi samo za određenu užu klasu problema.

2.7 Moguća unapređenja

Jedno od najznačajnijih poboljšanja date implementacije je paralelizacija, koja je već ostvarena i detaljno opisana u narednom poglavlju. Dalja unapređenja GANP implementacije su moguća u nekoliko pravaca:

- Primena raznih pristupa za bolju realizaciju kodiranja pomoću realnih brojeva i efikasnije rešavanje odgovarajućih problema numeričkog tipa relativno velikih dimenzija;
- Pokušati hibridizaciju GA sa većim brojem drugih metoda, što bi moglo dalje poboljšati performanse ove implementacije.

3. PARALELNA IMPLEMENTACIJA (PGANP)

Na osnovu teoreme o shemama može se uočiti pojava unutrašnjeg paralelizma GA. Pored toga, postoje dodatne mogućnosti za paralelizaciju GA pošto gotovo sve funkcije u GA implementaciji, sadrže visok stepen međusobne nezavisnosti, pa se mogu relativno uspešno paralelizovati.

U ovom poglavlju će biti prikazana paralelna GA implementacija za rešavanje NP-kompletnih problema (PGANP - Paralelni Genetski Algoritam za rešavanje NP-kompletnih problema - Parallel Genetic Algorithm for solving the NP-complete problems). Dati paralelni genetski algoritam je implementiran korišćenjem WMPI razvojnog okruženja, koji podržava konstrukcije MPI standarda verzije 1.1 . Implementiran je distribuirani model paralelizacije GA, pri čemu su izdvojeni neki zajednički delovi koji mogu biti iskorišćeni i za implementaciju nekog drugog modela paralelizacije. Opis opštih karakteristika paralelne implementacije je dat u odeljku 3.2, a pojedinačnih funkcija vezanih distribuirani model u odeljku 3.3.

3.1 Neke od postojećih paralelnih GA implementacija (PGA)

Paralelizacija GA se najčešće odvija po distribuiranom ili centralizovanom modelu. U distribuiranom modelu PGA svaka potpopulacija primenjuje poseban operator selekcije, koji se izvršava nezavisno od drugih. Drugi model ima centralizovani mehanizam selekcije cele populacije, odnosno svih potpopulacija.

U [Tns87] je primenjen PGA za optimizaciju Walsh-olikih funkcija korišćenjem paralelnog računara sa 64 procesora, povezanih kao hiperkocka (hypercube). Korišćen je distribuirani PGA, koji se u literaturi naziva i model ostrva (island model), gde svaki procesor izvršava genetski algoritam na sopstvenoj potpopulaciji. Dobre jedinke se periodično razmenjuju između susednih procesora, i uključuju se u potpopulaciju suseda u narednim generacijama. Razmena po susednim potpopulacijama, koje odgovaraju različitim dimenzijama hiperkocke, se odvija u različitim vremenskim trenucima. Iz prikazanih rezultata se vidi skoro linearno ubrzanje, u odnosu na tradicionalni sekvencijalni GA izvršen 1000 generacija, a preporučuje se 8 potpopulacija kao najbolji kompromis. Jedno poboljšanje date PGA implementacije je opisano u [Tns89].

Paralelna implementacija razmatrana u [Sta91] koristi nešto drugačiji pristup. Svaki procesor šalje kopije nekoliko svojih najboljih jedinki susednim procesorima, koji ih smeštaju na mesto najlošijih jedinki u svojoj potpopulaciji. Globalna veličina populacije je ostala fiksirana tokom izvršavanja, a

eksperimentisano je sa različitim veličinama potpopulacija dodeljenih svakom procesoru. Najbolji rezultati su dobijeni korišćenjem promenljivog nivoa mutacije, koji tokom izvršavanja GA raste do neke, unapred zadate vrednosti, paralelno sa rastom sličnosti jedinki u populaciji. Pri povećavanju broja potrebno je mnogo pažljivije odrediti granice i tok promene nivoa mutacije, jer je kod manjih potpopulacija mnogo veća mogućnost preuranjene konvergencije, ili gubitka genetskog materijala. Rezultati izvršavanja takođe ukazuju da je veoma bitno pravilno odrediti učestalost razmene jedinki jer ako je ona previše česta, ili jako retka, dolazi do značajnog pogoršanja performansi date implementacije.

PGA za rešavanje problema K-podele (K-partition problem) je opisan u [Coh91]. Svaki od 16 procesora povezanih u hiperkocku, izvršava GA na potpopulaciji od 80 jedinki uz razmenu jedinki na svakih 50 generacija. U datom problemu su u razmatranje uključene i nekorektne jedinke, pa je funkcija prilagođenosti računata po formuli $f(x) = c(x) + \lambda \cdot p(x)$, gde je $c(x)$ prilagođenost jedinke a $p(x)$ kaznena funkcija. Eksperimenti sa korišćenjem kaznene funkcije su pokazali da su najbolji rezultati dobijeni ako je za svaku potpopulaciju generisan slučajan broj $\lambda \in [0, 1]$, umesto fiksnog izbora $\lambda=1$ za sve potpopulacije.

Gordon i Whitley su u radu [Grd93] izvršili uporednu analizu osam različitih PGA i verziju Goldberg-ovog prostog GA ([Gol89]) na optimizaciji nekoliko raznolikih test funkcija. Testiranje je pokazalo da je distribuirani "model ostrva" postigao dobre rezultate, čak i na najtežim test-primerima.

Jedna od najpoznatijih paralelnih implementacija GA je data u [Müh92]. Ona je iskorišćena za rešavanje problema trgovačkog putnika (Traveling Salesman Problem - TSP) i problema podele grafa (graph partitionig problem). Korišćen je centralizovani model PGA, pa je populacija globalna, a svaka jedinka je dodeljena tačno jednom procesoru. Pri tome se genetski operatori selekcije, ukrštanja i mutacije vrše isključivo uz pomoć susednih jedinki, odnosno, jedinki koje pripadaju susednim procesorima u datoj paralelnoj arhitekturi, bez ikakve globalne kontrole. Nakon toga se kvalitet rešenja svake jedinke dodatno poboljšava korišćenjem lokalnog pretraživanja. Predlaže se sporo prosleđivanje dobrih jedinki kroz celu populaciju kao metod za sprečavanje preuranjene konvergencije. Dodatne informacije o raznim mogućnostima date PGA implementacije se mogu dobiti u [Müh91a] i [Müh91b].

U [Las91] je opisana PGA implementacija za rešavanje problema podele grafa korišćenjem 64-procesorskog transpjuterskog sistema. Selekcija se odvija nezavisno za svaku jedinku, korišćenjem nekoliko susednih jedinki u dvodimenzionalnoj višeprocesorskoj arhitekturi. Roditelj se zamenjuje potomkom, ukoliko ovaj ima veću ili jednaku vrednost najmanje jedne susedne jedinke. Najbolji rezultati su dobijeni korišćenjem velike veličine populacije uz mali broj suseda.

Dobri rezultati, pri rešavanju TSP-a su postignuti korišćenjem PGA na 64-procesorskom transpjuterskom sistemu kao što se može videti u [Grg91]. Veličina globalne populacije je bila 64, uz 8 suseda za svaku jedinku. Prilagođenost pojedinačne jedinke se računa relativno u odnosu na susede, a ne globalno za celu populaciju. Mehanizam selekcije u datoj implementaciji sadrži elitističku strategiju, u kojoj je potomak izabran u narednu generaciju, samo u slučaju da je bolji od svog roditelja. U paralelnoj implementaciji je ostvarena mala međuprocesorska komunikacija, zbog relativno malog broja

suseda, pa je ubrzanje bilo blisko linearnom. Detaljnije informacije o osnovi date PGA implementacije se mogu naći i u [Grg89].

Nešto drugačija PGA implementacija je izložena u radu [Fog91]. Ona je korišćena za rešavanje problema kontrole u realnom vremenu uz pomoć višeprocesorskog transpjuterskog sistema. Prvi procesor izvršava GA na populaciji od 250 jedinki, uz pomoć ostalih procesora u izračunavanju vrednosti jedinki. Pomoć se sastoji u tome da prvi procesor šalje genetske kodove jedinki ostalim procesorima, a oni izračunavaju njihove vrednosti i prosleđuju ih nazad prvom procesoru. Pošto je u datom problemu računanje vrednosne funkcije vremenski vrlo zahtevno, dati prisatup je bio pogodan i postignuti su zadovoljavajući rezultati. Postignuto je skoro linearno ubrzanje pri paralelnom radu do 20 procesora, međutim, ubrzanje pri radu većeg broja procesora brzo opada.

Levine je za rešavanje problema podele skupa (Set Partitioning Problem - SPP) primenio oba globalna modela paralelizacije GA:

- Distribuirani model je primenjen u implementaciji opisanoj u [Lvi93a] i [Lvi93b]. Dati PGA je zasnovan na relativno nezavisnim potpopulacijama, gde svaka od njih izvršava stacionarni GA, i one povremeno razmenjuju svoje najbolje jedinke. Odabir jedinke, koja će biti izbačena iz populacije da bi se na njeno mesto kopirala najbolja jedinka susedne populacije, vrši se turnirskom selekcijom. Dati pristup, gde se izbacuje najlošija jedinka na turniru sa nekoliko slučajno izabranih jedinki, se u praksi pokazao boljim od stalnog izbacivanja najlošije jedinke u populaciji. To se može objasniti činjenicom da, u prvom slučaju, i najlošija jedinka u populaciji ima neke šanse za izbor u narednu generaciju pa time doprinosi raznovrsnosti genetskog materijala. Izvršavanje je obavljeno na paralelnom računaru IBM SP1 sa 128 procesora, i dobijeni su vrlo dobri rezultati. Pri tome su rešene neke SPP instance veće dimenzije, sa nekoliko desetina hiljada celobrojnih promenljivih, koje su se pojavljivale u praksi.
- PGAPack paralelna biblioteka za razvoj paralelnih genetskih algoritama koristi i mogućnost postojanja jedinstvene globalne populacije, a poslovi izračunavanja vrednosnih funkcija se dele među procesorima odnosno procesima. Za paralelizaciju je korišćen model prosleđivanja poruka pa je, osim izvršavanja na specijalizovanom paralelnom računaru, ostavljena i mogućnost korišćenja mreže radnih stanica. PGAPack biblioteka je dostupna preko Interneta [Lvi95a], uz detaljno uputstvo dato u [Lvi95b]. Ova implementacija je takođe iskorišćena za rešavanje problema podele skupa, a rezultati su izloženi u radu [Lvi96].

Detaljniji pregled osnovnih karakteristika distribuiranog modela paralelizacije GA, kao i uporedna analaiza sa ostalim modelima se može naći u [Bld95].

Postoji još i veliki broj drugih PGA implementacija, koje zbog obima ovog rada, ne mogu sve biti detaljnije opisane, a navodimo literaturu gde se mogu pronaći detaljniji opisi: [Col91], [Shr91] i [Sho93].

3.2 Zajednički deo

MPI standard podržava paralelne konstrukcije relativno visokog nivoa, uz istovremeno očuvanje efikasnosti same implementacije. Time je omogućeno da programski kod ove paralelne implementacije bude relativno kratak a da ubrzanje bude blisko optimalnom.

3.2.1 Opis paralelne strukture

Svi podaci specifični za PGANP implementaciju su grupisani u paralelnu strukturu. Osnovna paralelna struktura je po načinu korišćenja podeljena na podstrukture koje su zadužene za:

- Arhitekturu zadatu od strane korisnika;
- Razmenu jedinki;
- Prosleđivanje rešenja;
- Kraj izvršavanja GA.

Osim date postoji i struktura zadužena za opis podataka u konfiguracionoj datoteci, kao i više nizova funkcijskih pokazivača, ali su oni manje važni za opis paralelne implementacije pa će detaljno biti opisani u Dodatku B.

3.2.1.1 Paralelna arhitektura

Kao što se može videti kasnije, arhitektura zadata od strane korisnika može, u nekim slučajevima, znatno uticati na performanse paralelnog GA. Ukoliko sve procese koji se izvršavaju na paralelnom računaru posmatramo kao čvorove, a susedne procese koji razmenjuju jedinke obeležimo kao grane, implicitno je definisan graf razmene jedinki tokom izvršavanja paralelnog GA. Pošto se jedinke razmenjuju između susednih procesa samo u distribuiranom modelu, dati graf nema uticaja na centralizovani model izvršavanja.

Za sve modele paralelizacije je značajan način prosleđivanja GA i problem strukture od osnovnog procesa kao svim ostalim procesima. Ovakav način prosleđivanja se može formalno opisati nekom drvoidnom strukturom, koju nazivamo *drvo prosleđivanja*. U takvoj formalizaciji, za svaki proces možemo definisati dva pojma: *sused roditelj* i *susedi potomci*. Svaki proces, osim osnovnog, ima tačno jednog suseda roditelja od koga dobija GA i problem strukturu na početku i prosleđuje ih svojim susedima potomcima. Ova struktura (drvo prosleđivanja) se, ali u obrnutom smeru, koristi u distribuiranom modelu za prikupljanje konačnih rešenja i njihovo prosleđivanje ka osnovnom procesu.

Data podstruktura sadrži sledeće podatke:

- Redni broj tekućeg procesa;
- Broj svih suseda (susednih procesa), niz njihovih indeksa i redni broj tekućeg suseda;
- Indeks suseda roditelja;
- Broj suseda potomaka i niz njihovih indeksa;
- Niz indikatora koji označavaju da li je odgovarajući sused završio sa radom svoj lokalni GA;
- Pokazivač na funkciju za inicijalizaciju date paralelne arhitekture;
  Pokazivači na funkcije za slanje GA odnosno problem strukture, kao i odgovarajuće funkcije za njihov prijem.

3.2.1.2 Razmena jedinki

U distribuiranom modelu se jedinke razmenjuju relativno retko, najčešće samo jednom u nekoliko generacija, i to samo između potpopulacija koje se izvršavaju na susednim procesima. Zbog toga, nije toliko važna efikasnost razmene jedinki i sinhronizacija procesa u okviru višeprocesorskog računara, već su mnogo važniji učestanost razmene i izbor odgovarajućih jedinki koje se razmenjuju.

Ukoliko bi razmena jedinki bila suviše česta, vrlo brzo bi potpopulacije sadržale više kopija jednih istih jedinki, tada bi efekti paralelizacije bili zanemarljivi i rezultati dobijeni paralelnim izvršavanjem bi bili slični sekvencijalnom izvršavanju GA na samo jednom procesu (procesoru). Pri suviše retkoj razmeni jedinki, genetski algoritmi koji se izvršavaju na pojedinačnim procesima su praktično nezavisni, što daje nešto bolje rezultate nego pri suviše čestoj razmeni jedinki, ali su performanse paralelizacije i dalje relativno loše. Uz dobro izabranu učestanost razmene jedinki, postiže se dobar kompromis između sledećih komponenti pretrage paralelnog GA: istraživanja (exploration) i iskorišćenja (exploitation). To doprinosi punom korišćenju prednosti paralelizacije, koje se ogledaju u boljem kvalitetu rešenja, uz istovremeno kraće vreme izvršavanja, u odnosu na sekvencijalni GA.

Podaci grupisani u podstrukturi za razmenu jedinki su:

- Pokazivači na funkcije za slanje i prijem jedinki;
- Pomoćni nizovi za prijem i slanje celobrojnih i realnih vrednosti, kao i broj članova datih nizova. Funkcijski pokazivači se šalju preko celobrojnih rednih brojeva izabrane funkcije u odnosu na zadati poredak;
- Koliko jedinki šalje osnovni proces odjednom na izračunavanje ostalim procesima. Pošto se taj broj razlikuje na početku slanja i kasnije, memorišu se tri vrednosti (koristi se samo u centralizovanom modelu);
- Učestanost razmene jedinki između susednih potpopulacija, odnosno, broj generacija genetskog algoritma između dve razmene (distribuirani model);
- Pokazivači na funkcije za izbor jedinke koja se šalje susedima, odnosno, jedinki koje se izbacuju iz populacije (distribuirani model);
- Broj učesnika, redom, na turnirima gde se biraju jedinke za slanje susedima odnosno izbacivanje iz populacije, ako se vrši turnirski izbor jedinki (distribuirani model).

3.2.1.3 Prosleđivanje rešenja

Podstruktura za prosleđivanje rešenja u distribuiranom modelu služi za prikupljanje konačnih rešenja od suseda potomaka, i prosleđivanje konačnog rešenja za dati proces susedu roditelju. Pri tome se rešenje lokalnog GA koji se izvršavao na tekućem procesu, dopunjuje rezultatima dobijenim od suseda potomaka, i na taj način se formira rešenje koje se prosleđuje dalje susedu roditelju.

Za funkcionisanje ovog segmenta paralelnog GA neophodni su sledeći podaci:

- Pokazivači na funkcije za inicijalizaciju, slanje i prijem konačnog rešenja GA u distribuiranom modelu;
- Memorijski prostor gde se smeštaju konačna rešenja suseda potomaka.

3.2.1.4 Kriterijum završetka paralelnog GA

Jedan od važnih aspekata paralelnog GA je i kriterijum za završetak izvršavanja. Pošto se u distribuiranom modelu na svakom procesu izvršava lokalni GA, postoje dva kriterijuma završetka: globalni i lokalni.

U slučaju lokalnog kriterijuma završetka, dati proces prekida izvršavanje lokalnog GA, svim susedima šalje poruku o tome, ali i dalje nastavlja komunikaciju sa susednim procesima (prijem i prosleđivanje njihovih poruka). Pri tome se prikupljaju i memorišu konačna rešenja od suseda potomaka. Po

prikupljanju svih rešenja, ona se kombinuju sa rešenjem tekućeg GA i konačno rešenje se šalje susedu roditelju.

Po prijemu od suseda roditelja poruke o globalnom kraju rada, ona se prosleđuje susedima potomcima i prekida se izvršavanje lokalnog GA. Kao i u prethodnom slučaju prikupljaju se konačna rešenja i šalju susedu roditelju.

Data podstruktura sadrži sledeće podatke:

- Indikator da li je nastupio globalni kraj izvršavanja paralelnog GA (distribuirani model);
- Pokazivači na funkcije za slanje i prijem poruke o kraju izvršavanja GA (globalnom ili lokalnom).

3.2.2 Shema izvršavanja paralelnog algoritma

Shema izvršavanja paralelnog algoritma je data na slici 3.1, gde prefiksi M, S i A redom označavaju da se odgovarajuća funkcija izvršava samo na glavnom procesu, na ostalim procesima ili na svim procesima.

```
M_Učitavanje_Paralelne_Konfiguracije();
M_Slanje_Paralelne_Konfiguracije();
S_Prijem_Paralelne_Konfiguracije();
A_Inicijalizacija_Paralelne_Arhitekture();
A_Inicijalizacija_GA_Strukture();
M_Učitavanje_GA_Strukture();
M_Slanje_GA_Strukture();
S_Prijem_i_prosleđivanje_GA_Strukture();
M_Učitavanje_Problem_Strukture();
M_Slanje_Problem_Strukture();
S_Prijem_i_prosleđivanje_Problem_Strukture();
M_Štampanje_Konfiguracionih_Datoteka();
while (! Globalni_Kraj_GA() )
    Izvršavanje_Paralelnog_Modela_GA();
M_Štampanje_Rešenja();
```

Slika 3.1 Opšta shema paralelnog algoritma

Inicijalizacija i učitavanje paralelne strukture, koja sadrži sve podatake potrebne za paralelnu implementaciju, se vrši prvo na osnovnom procesu. On zatim svakom od ostalih procesa pojedinačno šalje datu paralelnu strukturu. Oni uporedo vrše inicijalizaciju paralelne strukture i prijem njenih podataka.

Po prijemu cele paralelne strukture vrši se inicijalizacija odgovarajuće paralelne arhitekture zadate od strane korisnika (hiperkocka, jednodimenzionalni niz, dvodimenzionalni niz, i sl.). U distribuiranom modelu je važna uloga paralelne arhitekture jer ona definiše koje potpopulacije razmenjuju jedinke tokom rada lokalnih GA. Ona takođe određuje i način prosleđivanja konačnih rešenja svakog lokalnog GA, tako da na kraju stignu do osnovnog procesa. Poželjno je da odgovarajuća zadata paralelna arhitektura odgovara fizičkim vezama između procesora datog višeprocesorskog računara, jer se u tom slučaju poboljšava efikasnost međuprocesorske komunikacije.

Pošto je izvršen prijem paralelne strukture i inicijalizacija na datom procesu počinje sa izvršavanjem paralelni GA. Osnovni proces učitava GA strukturu i podatke specifične za problem koji se rešava. Koristeći izabranu paralelnu

arhitekturu oni se zatim prosleđuju do svih ostalih procesa. U većini slučajeva pri prenosu paralelne strukture, GA strukture ili problem strukture, pogodno je njihovo pakovanje i prenošenje u što manjem broju poruka, što se podstiče kroz konstrukcije MPI standarda. Time se štedi vreme potrebno za međuprocesorsku komunikaciju i doprinosi većoj efikasnosti cele paralelne implementacije.

Po prijemu podataka važnih za rad GA i parametara datog problema, izvršava se glavna procedura zadata funkcijom *Izvršavanje_Paralelnog_Modela_GA*() sve do globalnog kraja rada GA koji je definisan funkcijom *Globalni_Kraj_GA*(). Izvršavanje glavne procedure se razlikuje u zavisnosti od izabranog modela paralelizacije i procesa na kome se izvršava (osnovni proces ili neki od ostalih procesa). Detaljnije o raznim varijantama ove funkcije se može videti u narednom odeljku.

Po završetku rada glavne procedure osnovni proces štampa rešenje i svi procesi prekidaju rad.

3.3 Distribuirani model

Kao što je već rečeno distribuirani model je vrlo često i efikasno korišćen za paralelizaciju GA. Svaki proces izvršava genetski algoritam na sopstvenoj potpopulaciji, uz povremenu razmenu nekih, najčešće najboljih jedinki. Pri tome je frekvencija razmene vrlo važan faktor za uspešno funkcionisanje PGA. Funkcija *Izvršavanje_Paralelnog_Modela_GA*() sa slike 3.1 koja odgovara distribuiranom modelu je detaljnije prikazana na slici 3.2 .

```
gener++;
Prijem_Jedinki_od_Suseda();
for (i=N_elite+1; i<N_pop; i++)
  if (! Primljen_od_Suseda(i))
      p_i = Vrednosna_Funkcija(i);
Računanje_Funkcije_Prilagođenosti();
if(gener % f_razmene == 0)
 {
     Izaberi_Jedinke_za_Slanje();
     Pošalji_Jedinke_Susedima();
 }
Prijem_Poruke_O_Kraju_GA();
if(! Kraj_Lokalnog_GA())
 {
     Selekcija();
     Ukrštanje();
     Mutacija();
 }
else
     Slanje_Poruke_O_Kraju_Lokalnog_GA();
```

Slika 3.2 Shema distribuiranog PGA

3.3.1 Razmena jedinki između potpopulacija

Kao što možemo videti na slici 3.2 na početku svake generacije distribuiranog GA se vrši prijem jedinki od susednih procesora i njihovo

uključivanje u svoju potpopulaciju. Pošto je njihova vrednost već bila izračunata, direktno se preuzima bez potrebe da se ponovo računa. Pri preuzimanju jedinki od susednih procesora, prethodno se mora osloboditi mesto za njeno smeštanje. Pri tome se izbor jedinke za izbacivanje iz tekuće potpopulacije, da bi na njeno mesto bila smeštena preuzeta jedinka, može vršiti na različite načine. Najčešće se primenjuje pristup da se uvek izbacuje najlošija jedinka u potpopulaciji, ali takav izbor se može vršiti i turnirski, pri čemu se izbacuje najlošija jedinka na turniru. Time se daje određena šansa i najlošijoj jedinki u potpopulaciji da ostane, što u nekim slučajevima može da se odrazi pozitivno na izvršavanje lokalnog GA. Međutim, pošto u praksi oba ova pristupa daju vrlo slične rezultate, obično se primenjuje onaj prvi, pošto je nešto jednostavniji.

Posle toga izračunavaju se vrednosti jedinki u populaciji za koje je to potrebno (nije već ranije poznata njihova vrednost) i računaju se odgovarajuće prilagođenosti svih jedinki u populaciji. Ukoliko je nastupila generacija u kojoj se vrši slanje jedinki, bira se odgovarajuća jedinka i šalje svim susednim procesorima. Izbor jedinke za slanje susedima takođe može da se vrši na više načina. Najjednostavnije je ukoliko biramo uvek najbolju jedinku u potpopulaciji, a izbor se takođe može vršiti i turnirski (šaljemo jedinku koja je najbolja na odgovarajućem turniru).

3.3.2 Globalni i lokalni završetak izvršavanja GA

Tokom daljeg izvršavanja proverava se da li postoji neka poruka o eventualnom kraju izvršavanja GA. Ukoliko takva poruka postoji, vrši se njen prijem, a zatim su moguća dva slučaja, u zavisnosti od sadržine date poruke.

Ako je data poruka donosila informaciju o globalnom završetku izvršavanja GA, ona se prosleđuje dalje procesorima "potomcima" i prekida se dalje izvršavanje lokalnog GA. Nakon toga se samo primaju konačna rešenja od suseda "potomaka", kombinuju sa rešenjem lokalnog GA na tekućem procesoru i na kraju prosleđuju susedu "roditelju". Kada osnovni procesor primi sva rešenja od svojih suseda "potomaka" i generiše konačno rešenje, prekida se rad paralelnog programa i svi procesori prekidaju svoje izvršavanje.

Po eventualnom prijemu poruke o lokalnom završetku izvršavanja GA na nekom od susednih procesora, beleži se ta informacija. Pri tome su moguće dve različite strategije:

- Takvim susedima se više ne šalju jedinke, jer su oni završili izvršavanje svog lokalnog GA:
- Jedinke se i dalje šalju svim susedima a dati susedni procesori pošto ne mogu sami da koriste ovakve jedinke, jer su završili rad svog lokalnog GA, prosleđuju ih dalje. Pri tome se beleže i informacije o prvom pošiljaocu jedinke, čime se efikasno spečava ciklično prosleđivanje jedinki.

Ukoliko je završeno izvršavanje lokalnog GA i svi susedi su završili izvršavanje svojih lokalnih genetskih algoritama, odgovarajuća poruka se šalje susedu "roditelju". Ukoliko osnovni procesor primi takvu poruku od svih suseda potomaka i u međuvremenu je i sam završio svoj lokalni GA, on generiše poruku o globalnom završetku izvršavanja GA i šalje je svojim susedima "potomcima".

3.3.3 Genetski operatori

Ukoliko nije ispunjen nijedan kriterijum za završetak lokalnog GA (ili globalni završetak GA), izvršavaju se redom genetski operatori selekcije, ukrštanja i mutacije. Oni se primenjuju na potpuno isti način kao i u sekvencijalnom GA, što važi i za ostale slične aspekte GA (politika zamene generacija, izbacivanje višestruke pojave jedinki u populaciji, itd), pa ih nećemo ovde posebno navoditi.

U suprotnom, ako je završen lokalni GA, nema više smisla izvršavanje genetskih operatora i ostalih sličnih aspekata, već se do momenta globalnog završetka GA izvršava samo:

- prosleđivanje konačnih rešenja;
- prosleđivanje poruka o kraju izvršavanja GA;
- eventualno prosleđivanje jedinki.

4. KEŠIRANJE GA

U ovoj implementaciji je prvi put primenjeno keširanje genetskih algoritama, kao način za poboljšavanje njihovih performansi. Razlog za primenu takve tehnike je pojava ponavljanja istih genetskih kodova u toku izvršavanja GA. U većini slučajeva je pogodnije zapamtiti date genetske kodove i njihove vrednosti u prvom pojavljivanju, a zatim u sledećim pojavljivanjima direktno očitati vrednosti, umesto ponovnog računanja. Napomenimo da keširanje ni na koji način ne utiče na kvalitet rešenja dobijenog pomoću GA, već samo služi za smanjenje vremena izvršavanja GA.

4.1 Tehnike keširanja

Keširanje je uobičajena i često korišćena tehnika za poboljšavanje performansi računarskih sistema. Primenjuje se već dugo kod operativnih sistema, a i za ubrzanje rada nekih programskih paketa. Ovde ćemo napraviti pregled nekih uspešnih tehnika keširanja različitih namena, i mogućnosti primene sličnih pristupa za keširanje GA.

U [Tnb92] su opisane tehnike keširanja primenjene za dizajn nekih operativnih sistema. Pri tome je dat opis nekih opštih tehnika keširanja i pregled njihovih karakteristika. Kao jedna od jednostavnih za implementaciju, a u praksi često vrlo efikasna, prikazana je tehnika najstarijeg korišćenog člana (Least Recently Used - LRU). Data tehnika je takođe primenjena i u većini programa za keširanje WWW, kao što se može videti u [Pit94] i [Mar96].

Jedna ideja o keširanju pomoću heš (HASH) tabela je prikazana u [Wlh92]. Ovde je primenjena u potpuno drugačijem kontekstu (izbor reprezentacije podataka za interaktivnu vizuelizaciju), i ta ideja se pokazala kao univerzalna i vrlo korisna pri keširanju GA.

Još neke od korisnih tehnika keširanja se mogu videti u [Pla88], [Mat91], [Kam92], [Nie92] i [Dah94].

4.2 Prethodne tehnike keširanja GA

Prva ideja o korišćenju keširanja za smanjenje vremena izvršavanja GA se pojavila pri rešavanju prostog lokacijskog problema (SPLP) i detaljno je opisana u radu [Kra97b]. Data metoda je primenjena na unapređenje prostog genetskog algoritma i testirana na SPLP instancama male dimenzije. U tim uslovima je dobijeno značajno ubrzanje izvršavanja GA (20-40%), ali su kasnije dobijeni znatno skromniji rezultati pri primeni složenijih genetskih operatora i izvršavanjem na problemima velike dimenzije. Uočeno je da su glavni nedostaci

date metode keširanja izbor strategije i primenjena linearna struktura podataka (jednodimenzioni niz) za njenu implementaciju.

4.2.1 LRU primenjen na linearnoj strukturi

I pored uočenih nedostataka u implementaciji, data ideja o keširanju GA se pokazala veoma perspektivnom. Nastavljeno je sa eksperimentima za nalaženje bolje strategije keširanja i strukture podataka prilagođene datoj strategiji. Od nekoliko strategija preuzetih iz literature, najpogodnijom se pokazala LRU. Eksperimentisano je i sa nekoliko raznih struktura podataka (RB stabla, heš-tabela), ali se one nisu mogle uspešno uklopiti u LRU strategiju. Pri tome je nivo prepoznavanja istih jedinki (duplikata) bio znatno smanjen, a time i performanse keširanja. Zbog toga je zadržana linearna struktura podataka. Struktura datog algoritma za keširanje, zasnovanog na LRU strategiji, se može videti na slici 4.1 . Pri tome dati programski segment zamenjuje funkciju za računanje vrednosti funkcije *Vrednosna_Funkcija()* u originalnoj šemi GA datoj na slici 1.1 .

```
if ( Sadrži (keš_memorija, genetski_kod_jedinke) )
    Dodeli_Vrednost(jedinka, keš_blok)
else
 {
    Vrednosna_Funkcija();
    if(Popunjena(keš_memorija))  Obriši(keš_memorija, Najstariji_Blok());
    Dodaj(keš_memorija, jedinka);
 }
Postavi_za_Najnoviji(keš_blok);
```

Slika 4.1 LRU strategija

U radu [Kra99b] je detaljno opisana data metoda za keširanje i prikazani su rezultati primene na prostom GA i na složenijem GA za rešavanje SPLP. Keširanje je poboljšalo vreme izvršavanja obe varijante GA na svim instancama datog problema. Međutim, rezultati keširanja (ubrzanje u odnosu na osnovni GA) složenijeg GA su bili prilično lošiji od rezultata keširanja prostog GA. To se može objasniti time da je primenom složenijih genetskih operatora dobijeno rešenje boljeg kvaliteta u dosta kraćem vremenu izvršavanja, pa je ostalo daleko manje potencijala za dalja poboljšavanja.

Takođe su na SPLP instancama veće dimenzije, u obe varijante GA, dobijeni znatno lošiji rezultati keširanja u odnosu na instance manje dimenzije. Višestruki su uzroci ove pojave, a jedan od njih je relativno mala veličina keš memorije koja je primenjena, imajući u vidu način njenog pretraživanja. Detaljniji opis svih aspekata koji utiču na ove pojave se može videti u radu [Kra99b].

Keširanje je kao metoda već uspešno iskorišćeno za poboljšanje performansi GA implementacije za rešavanje NP-kompletnih problema, kao što se može videti u radovima [Kra98b], [Kra99a] i [Kra99c].

4.2.2 LRU primenjen na dvostrukoj heš-tabeli

Da bi se prevazišli gore pomenuti problemi (koji se javljaju u prethodnom algoritmu za keširanje GA) primenjena je posebna struktura podataka,

dvostruka heš-tabela. Ona se sastoji od dve heš-tabele koje sadrže pokazivače na blokove keš memorije. Svaki blok keš memorije se može naći pretragom po proizvoljnoj heš-tabeli.

Kao što se može videti na slici 4.1, najvažnije komponente programa za keširanje GA zasnovanog na LRU strategiji, su sledeće funkcije:

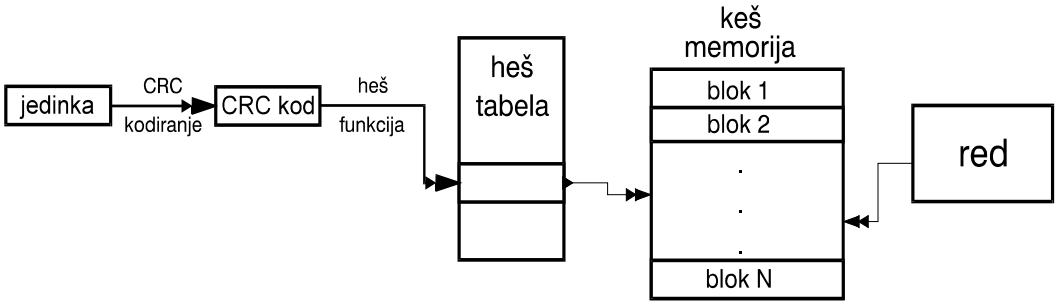
- *Sadrži*(), koja pretražuje keš memoriju, i ispituje da li ona sadrži dati genetski kod jedinke;
- *Dodaj*(), gde se tekuća jedinka smešta u slobodan blok keš memorije;
- *Obriši*(), koja izbacuje iz keš memorije blok (jedinku) koji najduže vremena nije korišćen i time oslobađa mesto za smeštanje nove jedinke.

Funkcije *Sadrži*() i *Dodaj*() su implementirane preko prve heš-tabele, dok je za realizaciju funkcije *Obriši*() neohodna druga heš-tabela.

Dvostruka heš-tabela se, za razliku od ostalih ranije pomenutih struktura podataka, uspešno uklopila u LRU strategiju. Sa druge strane, dobijena je vrlo efikasna implementacija, jer kao što je poznato, operacije pretrage, dodavanja i brisanja elementa u heš-tabeli u proseku zahtevaju konstantno O(1) vreme izvršavanja.

4.3 Implementacija pomoću heš-red strukture

Iako se operacije pretrage, dodavanja i brisanja blokova nad keš-memorijom izvršavaju u prosečno konstantnom vremenu primenom prethodnog pristupa, moguća je modifikacija koja u praksi nešto poboljšava vreme izvršavanja. To je ostvareno zamenom druge heš-tabele i korišćenjem reda (queue). Na taj način su funkcije *Dodaj*() i *Obriši*() nešto uprošćene, i imaju manje osnovnih operacija, što ih donekle ubrzava. Iako poboljšanje vremena izvršavanja (u odnosu na dvostruku heš-tabelu) nije spektakularno, razlike se mogu uočiti. Na slici 4.2 je dat shematski prikaz ove strukture.



Slika 4.2 Heš-red struktura

4.3.1 Heš-funkcija

Osnovna ideja pri projektovanju svake heš-tabele je što ravnomerniji raspored blokova u njoj. Za heš-funkciju je izabrana metoda množenja (multiplication method). Iako se u teoriji ([Crm90]), kao nešto bolja prikazuje univerzalna heš-funkcija (universal hashing), u našem slučaju se ne pokazuju nikakve razlike. Stoga je izabrana metoda množenja, koja je jednostavnija za implementaciju.

Razlozi za uspešnu primenu metode množenja se mogu objasniti specifičnim uslovima koji važe u primenjenoj heš-tabeli. Argument heš-funkcije,

u datom slučaju, nije genetski kod jedinke, već njegova CRC vrednost. Poznato je da pri CRC preslikavanju slični argumenti daju potpuno različite vrednosti. Zbog toga su CRC vrednosti već prilično ravnomeno raspoređene u 32-bitnom opsegu, pa je tada zadatak heš-funkcije za što ravnomernijim rasporedom blokova u velikoj meri olakšan.

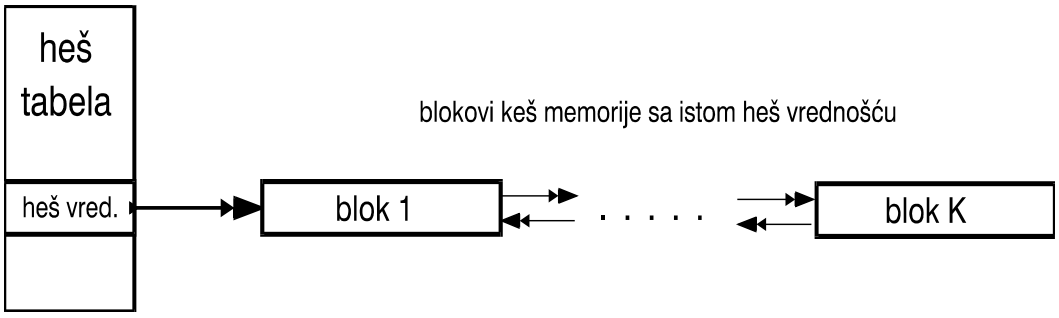
Računanje heš-funkcije uz pomoć metode množenja je vrlo jednostavno, a ona se implementira pomoću formule:

$$h(k) = \mathbf{int}(m * \mathbf{frac}(k*A)). \quad (4.1)$$

Vrednost *m* je veličina heš-tabele, *k* argument heš-funkcije, a A je unapred izabrana konstanta, pri čemu funkcija *int* računa celobrojni, a *frac* razlomljeni deo izraza. U ovoj implementaciji je izabrano $A = \frac{\sqrt{5}-1}{2} \approx 0.618034$ (ova vrednost je preporučena u [Knu73]).

4.3.2 Rešavanje kolizije

Kolizija je pojava kada postoji više blokova sa istom vrednošću heš-funkcije. U tekućoj implementaciji keširanja GA, za sprečavanje kolizije koristi se otvoreno hešovanje (collision resolution by chaining). Dati pristup razrešava koliziju formiranjem povezane liste za svaku poziciju heš-tabele, pa se na taj način može pojaviti i više blokova sa istom vrednošću. Pošto su u datoj implementaciji spregnuti heš-tabela i red, umesto obične (jednostruke) povezane liste, za svaku poziciju heš-tabele je neophodno da se koristi dvostruko povezana lista, kao što se može videti na slici 4.3 .



Slika 4.3 Rešavanje kolizije primenom otvorenog hešovanja

Pretraživanje heš-tabele se odvija u dve faze. U prvoj fazi se izračunava vrednost heš-funkcije za dati blok i na osnovu te vrednosti određuje se pokazivač na povezanu listu. Zatim se u drugoj fazi, odgovarajuća povezana lista pretražuje sekvencijalno, do pronalaženja traženog elementa. Pošto je u datoj implementaciji broj pozicija u heš-tabeli jednak broju vrednosti heš-funkcije, a data heš-funkcija prilično ravnomerno raspoređuje blokove, dobijene dvostruko povezane liste su kratke, i brzo se pretražuju.

Novi element se ubacuje na prvo mesto dvostruko povezane liste, a ostali blokovi sa tom vrednošću se pomeraju za jedno mesto, što se najlakše implementira, a relativno je brzo. Izbacivanje elementa iz heš-tabele je iste složenosti kao pretraživanje, uz dodatno postavljanje pokazivača u odgovarajućoj dvostruko povezanoj listi.

Otvoreno hešovanje za rešavanje kolizije heš-tabela je detaljnije opisano u [Crm90] i [Pau97], gde se mogu videti i ostali pristupi rešavanju kolizije, kao i njihova uporedna analiza. Detaljnije informacije o heš-tabelama se mogu naći i u [Knu73], [Brs88] i [Man91].

Napomenimo i to da, osim potrebe za realizacijom preko dvostruko povezane liste, nema nikakvih drugih uticaja činjenica da se razrešava pojava kolizije na heš-red strukturi, umesto na običnoj heš-tabeli.

4.3.3 Dobijanje CRC kodova

U slučaju keširanja GA javlja se potreba da u svakoj generaciji, za svaku jedinku ispitujemo da li već postoji u keš memoriji (funkcija *Sadrži*() na slici 4.1). Prilikom poziva date funkcije potrebno je izvršiti nekoliko poređenja genetskih kodova dve jedinke, gde prva pripada populaciji u nekoj generaciji GA, a druga je zapamćena u određenom bloku keš memorije.

U većini primena poređenje reči je efikasno, jer se one najčešće razlikuju već u nekom od prvih nekoliko slova, pa se poređenje brzo završava. Međutim, genetski kodovi imaju nezgodnu osobinu da se posle nekoliko generacija GA grupišu u određeni region pretrage pa postanu vrlo slični.

Ta osobina još više dolazi do izražaja ukoliko su genetski kodovi relativno veće dužine pa tada direktno poređenje može biti vrlo sporo. U tom slučaju je pogodnije poređenje genetskih kodova preko njihovih CRC vrednosti. Pri tome je neophodno izračunavanje CRC vrednosti za svaku jedinku u populaciji, ali se to izvršava samo jednom. Posle toga je poređenje vrlo efikasno, jer se genetski kodovi dve jedinke ne porede direktno, već se prvo uporede njihove CRC vrednosti. Ako su one različite, genetski kodovi jedinki se sigurno razlikuju. Direktno poređenje celih genetskih kodova je potrebno samo kada su CRC vrednosti jednake, što se u praksi dešava vrlo retko.

Pošto za svaku jedinku postoji izračunata CRC vrednost njenog genetskog koda, heš-funkcija se primenjuje direktno na tu CRC vrednost, a ne na ceo genetski kod. Time, ne samo što je ubrzano i uprošćeno računanje heš-funkcije, već se jedinke ravnomernije smeštaju kao blokovi u keš memoriju.

Za računanje 32-bitne CRC vrednosti je, u ovoj implementaciji, primenjen algoritam zasnovan na čitanju gotovih koeficijenata (table-driven algorithm) opisan u [Wlm93]. Pošto su koeficijenti uvek isti, ima ih relativno malo (256), a njihovo izračunavanje je vremenski relativno zahtevno, oni se posebno unapred računaju, i smeštaju u datoteku ”CRC.DAT”. Pri inicijalizaciji se samo dati koeficijenti pročitaju iz datoteke i smeste u niz *crctable.* Na osnovu koeficijenata iz datog niza se vrlo efikasno računa CRC vrednost, kao što se može videti sa slike 4.4 koja sadrži odgovarajući programski kod u jeziku C.

```
unsigned long FindCRC(int len, char *msg)
{ int l;
  unsigned long crc;

   crc = INIT;
   l = len;
   while(l--)
     crc = (crc>>8) ^ crctable[(crc ^ *msg++) & 0xFFL];
   crc = crc ^ XOROT;

   return crc;
}  /* FindCRC */
```

Slika 4.4 Funkcija za računanje CRC vrednosti

Svi parametri potrebni za računanje CRC vrednosti su dati u tabeli 4.1, a već su primenjeni u programskim specifikacijama za: PKZip, AUTODIN II, Ethernet i FDDI. Dati parametri sadrže redom sledeće informacije:

- Ime pridruženo odgovarajućem algoritmu;
- Broj bitova rezultata;
- Generator polinoma koji se kasnije primenjuje za računanje CRC vrednosti. Od izbora ovog parametra u velikoj meri zavisi kvalitet dobijenih CRC vrednosti, pa je preuzet direktno iz literature ([Tnb81], p. 130-132), gde se može naći i detaljan opis;
- Početna vrednost u registru, pri računanju CRC koda;
- Logička vrednost koja označava da li se vrši obrtanje ulaznih bitova, pri čemu bit na poziciji 0 postaje bit najveće težine, ili ne, kada je bit na poziciji 0 najmanje težine;
- Parametar koji slično prethodnom, označava da li se konačna vrednost u registru invertuje ili ne, pre vraćanja rezultata;
- Vrednost na koju se primenjuje operacija isključive disjunkcije (XOR) po bitovima sa tekućom vrednosti registra posle eventualne inverzije definisane prethodnim parametrom RefOut;
- Poslednji parametar nije obavezan, već samo služi za proveru korektnosti algoritma. On predstavlja CRC vrednost ASCII stringa "123456789" odnosno heksadecimalnog niza 0x313233343536373839.

Tabela 4.1 Parametri CRC algoritma

Parametar	Tip	Vrednost
Name	string	"CRC-32"
Width	integer	32
Poly	hex. long int.	04C11DB7
Init	hex. long int.	FFFFFFFF
RefIn	boolean	True
RefOut	boolean	True
XorOut	hex. long int.	FFFFFFFF
Check	hex. long int.	CBF43926

Detaljan opis datih parametara, kao i ostalih karakteristika CRC kodiranja, je dat u [Tnb81], [Nel91] i [Wlm93]. Opis još nekih efikasnih CRC implementacija se može videti u [Grf87] i [Sar88].

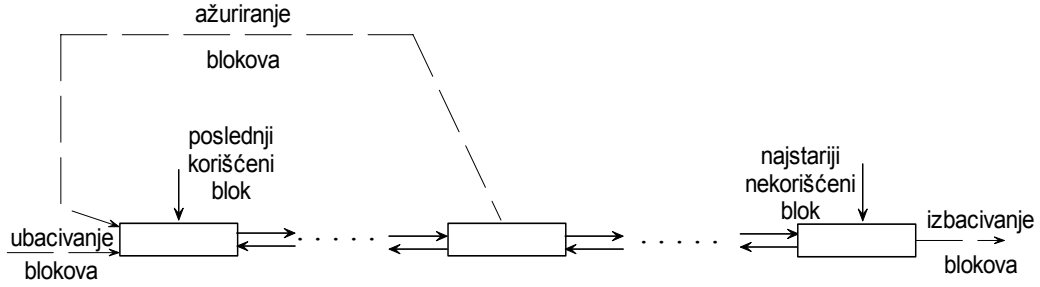
4.3.4 Korišćenje heš-tabele

Implementacija funkcije *Sadrži*() nameće ideju da se u jednoj heš-tabeli memorišu pokazivači na jedinke, po heš-funkciji koja zavisi od genetskog koda svake jedinke. Kao što je rečeno heš-funkcija se primenjuje na 32-bitni CRC genetskog koda jedinke, čime je smeštanje blokova u heš-tabelu ravnomernije. Pri tome je, takođe, ubrzana procedura poređenja dve jedinke, što dodatno poboljšava performanse cele implementacije.

Funkcija *Dodaj*() je slična funkciji *Sadrži*(), jer se na osnovu genetskog koda jedinke računa CRC, a nakon toga heš-funkcija računa poziciju gde treba datu ubaciti jedinku.

4.3.5 Korišćenje reda

Jedina problematična funkcija za implementaciju je *Obriši*(), koja mora da oslobodi memorijski prostor ukoliko se smešta nova jedinka, a keš memorija je potpuno popunjena. Da bi se ova funkcija efikasno implementirala, javlja se potreba za drugom strukturom podataka. U početku je korišćena druga heš-tabela, koja numerički vrednuje vreme poslednjeg pojavljivanja date jedinke. Kasnije je druga heš-tabela zamenjena red strukturom, čime je cela implementacija pojednostavljena.



Slika 4.5 Korišćenje red strukture pri keširanju

4.4 Eksperimentalni rezultati

Metoda za keširanje GA pomoću heš-red strukture, koja je detaljno opisana u prethodnom odeljku, je primenjena u praksi za rešavanje prostog lokacijskog problema (SPLP), dizajniranja mreže neograničenog kapaciteta (UNDP) i problema izbora indeksa (ISP). U ovom odeljku su dati uporedni rezultati primene GANP implementacije sa keširanjem u odnosu na slučaj kada je keširanje isključeno. Pošto su svi ostali parametri vezani za izvršavanje GA isti u oba slučaja, nećemo ih ovde posebno opisivati, jer su detaljno opisani u naredna tri poglavlja.

Pri testiranju su primenjene dve varijante keširanja GA, i upoređene sa slučajem kada nije korišćeno keširanje. U prvoj varijanti (CRCHashQueue) je pri poređenju jedinki pomoću CRC vrednosti, u slučaju pozitivnog odgovora kasnije

izvršeno i direktno poređenje. U drugoj varijanti (CRCHashQueue2) poređenje jedinki se vrši samo pomoću CRC vrednosti.

Pošto je testiranje obavljeno na većem broju instanci svakog od rešavanih problema, u ovom odeljku su (u tabelama 4.2 - 4.4) radi bolje preglednosti, prikazani samo prosečni rezultati u svakoj grupi instanci. Detaljni rezultati u vezi sa svakom pojedinačnom instancom dati su u dodatku C.

Rezultati testiranja su u tabelama 4.2 - 4.4 prikazani na odgovarajući način gde svaka kolona sadrži redom sledeće informacije:

- Imena instanci;
- Broj generacija genetskog algoritma;
- Vreme izvršavanja genetskog algoritma bez primene keširanja na datim instancama;
- Procenat ponovljenih jedinki tokom izvršavanja GA;
- Vreme izvršavanja genetskog algoritma sa uključenim keširanjem (CRCHashQueue). Faktor ubrzanja izvršavanja GA u kome je primenjeno keširanje, u odnosu na slučaj kada nema keširanja.
- Vreme izvršavanja i faktor ubrzanja za slučaj kada je primenjena funkcija CRCHashQueue2.

Pošto genetski operatori u opštem slučaju daju nedeterminističke rezultate, pri testiranju je fiksirana početna vrednost u generatoru slučajnih brojeva, što je omogućilo determinizam u izvršavanju. Zbog toga su kvalitet dobijenog rešenja i broj generacija GA isti u slučaju bez keširanja i sa keširanjem, pa je broj generacija naveden samo u prvom slučaju, a kvalitet dobijenih rešenja je potpuno izostavljen. On nije neophodan pošto nije od značaja za merenje performansi keširanja GA, a može se videti u narednim poglavljima.

U ovom odeljku takođe nisu date informacije o ostalim parametrima GA, kao i o opisima konkretnih instanci koje su korišćene za testiranje, jer su te informacije detaljno prikazane u okviru pojedinačnih problema. Ovde su samo prikazane karakteristike ubrzanja koja se dobijaju primenom keširanja na razne vrste instanci datih problema.

Tabela 4.2 Rezultati keširanja GA za SPLP

Instance	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
41 - 74	16.4	0.2146	27.12%	0.2169	0.989	0.2177	0.986
81 - 104	37.8	0.4958	29.43%	0.4833	1.026	0.480	1.033
111 - 134	129.2	2.014	33.07%	1.9308	1.043	1.9275	1.045
A - C	514.0	63.68	35.45%	44.09	1.444	44.06	1.445
MO	109.2	3.010	20.0%	2.824	1.066	2.812	1.070
MP	180.2	9.010	23.30%	8.130	1.108	8.118	1.110
MQ	257.2	19.99	24.42%	17.17	1.165	17.13	1.167
MR	445.2	60.91	26.83%	49.36	1.234	49.30	1.235
MS	857.8	257.72	27.73%	200.97	1.282	199.84	1.290

Tabela 4.3 Rezultati keširanja GA za UNDP

Instance	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
MA	19.0	0.362	22.53%	0.362	1.000	0.350	1.034
MB	135.6	4.046	31.23%	3.262	1.240	3.262	1.240
MC	151.6	6.296	24.01%	4.800	1.312	4.774	1.319
MD	101.2	5.932	16.67%	5.340	1.111	5.340	1.111
ME	961.6	74.97	41.88%	45.46	1.649	45.49	1.648

Tabela 4.4 Rezultati keširanja GA za ISP

Instance	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
AA	75.4	3.308	33.02%	2.614	1.265	2.600	1.272
AB	68.0	2.978	28.53%	2.430	1.226	2.440	1.220
AC	93.4	4.026	35.22%	3.034	1.327	3.020	1.333
AD	109.6	4.724	37.07%	3.518	1.343	3.480	1.357
AE	118.8	5.076	39.73%	3.790	1.339	3.788	1.340
AF	392.6	16.128	51.52%	10.194	1.582	10.128	1.592
AG	762.4	30.1	50.40%	19.038	1.581	18.96	1.588
AH	4735	182.12	53.00%	123.73	1.472	123.24	1.478

4.5 Kratka analiza dobijenih rezultata

Kao što se može videti iz tabela 4.2 - 4.4 primena keširanja je poboljšala performanse GA u značajnoj meri, što je doprinelo uspešnoj primeni na NP-kompletne probleme opisane u narednim poglavljima. Jedini slučaj kada je GA sa keširanjem sporiji u odnosu na klasični GA, je pri testiranju na instancama 41-74. Međutim i tada je razlika samo oko 1.5%, a vremenski manje od 1ms, što je praktično zanemarljivo.

Metoda množenja i izbor otvorenog hešovanja za heš-funkciju su se pokazale uspešnim za izbegavanje kolizije, čime se u heš-tabeli dobija, u proseku, mali broj blokova sa istom vrednošću heš-funkcije. Na taj način odgovarajuća dvostruko povezana lista sadrži mali broj članova, što doprinosi efikasnosti pretrage. Na osnovu toga su u praksi dobijeni sasvim dobri rezultati primenom heš-red strukture za keširanje GA.

Pozitivni rezultati su dobijeni čak i u slučaju kada je procenat jedinki koje se ponavljaju mali, tako da i u tim slučajevima data tehnika keširanja pokazuje poboljšanje u odnosu na osnovni GA. Ušteda u vremenu izvršavanja je obično oko 10-20%, ali u nekim slučajevima je i daleko više. Za SPLP instance A-C, UNDP instance ME i ISP instance AF-AH poboljšanje je više od 40%.

Testiranjem metode *CRCHashQueue2* na instancama sva tri problema (SPLP, UNDP i ISP), nijednom se nije desilo da CRC kodovi jedinki budu jednaki, a da same jedinke budu različite. Međutim, iako je ova metoda u

najvećem broju slučajeva brža od osnovne (*CRCHashQueue*), razlike u brzini su vrlo male. Zbog toga se preporučuje korišćenje osnovne metode.

Osim nekih izuzetaka, uočljivo je da keširanje GA postiže veću uštedu pri rešavanju instanci veće dimenzije. To se može objasniti većim brojem generacija, a samim tim i većom mogućnošću za pojavljivanjem istih jedinki tokom generacija.

Mogući pravci daljeg istraživanja u ovoj oblasti su:

- Primena strategije keširanja složenije od LRU;
- Eksperimentisanje sa raznim strukturama podataka pogodnim za datu izabranu strategiju.

5. PROST LOKACIJSKI PROBLEM

Posebnu klasu zadataka kombinatorne optimizacije predstavljaju lokacijski problemi. Oni imaju veliku primenu u raznim oblastima, a posebno u snabdevanju, planiranju zaliha i računarskim mrežama. Detaljnije informacije o ovim problemima, njihovoj klasifikaciji i metodama njihovog rešavanja se može naći u preglednim radovima [Fra83], [Aik85], [Dea85], [Lov88], [Mir90] i [Fra92]. Osim metoda opisanih u njima, pojedinačni opisi nekih specifičnih metoda se mogu naći i u [Abr95] i [Shm97]

Prost lokacijski problem je osnovni predstavnik ove klase problema, i mnogi lokacijski problemi se mogu formulisati pomoću njega. Postoji nekoliko naziva na engleskom jeziku koji odgovaraju ovom problemu: Simple Plant Location Problem (SPLP), Uncapacitated Facility Location Problem, Uncapacitated Warehouse Location Problem, Uncapacitated Plant Location Problem.

Postoji nekoliko direktnih uopštenja ovog problema, koja su opisana u radovima [Ard96a], [Cap96] i [Hlm97a], gde su date i metode njihovog rešavanja. Ove metode se u najvećem broju slučaja oslanjaju na metode za rešavanje osnovnog problema uz neke modifikacije.

5.1 Formulacija problema

Neka je dat skup $I = \{1, ..., m\}$ potencijalnih lokacija snabdevača, i skup $J = \{1, ..., n\}$ korisnika. Postavljanje snabdevača na potencijalnu lokaciju $i \in I$ podrazumeva fiksne troškove f_i. Svaki korisnik $j \in J$ zahteva količinu robe b_j, uz transportne troškove snabdevanja sa i-te lokacije c_{ij} (po jedinici robe). Bez ograničenja opštosti možemo transportne troškove c_{ij} pomnožiti zahtevanom količinom robe b_j, pri čemu se zahtev korisnika normalizuje na $b_j = 1$ (videti [BeJ93]).

Potrebno je naći plan uspostavljanja snabdevača na neke od tih lokacija, tako da ukupni troškovi budu minimalni. Pri tome u razmatranje ulaze fiksni troškovi postavljanja snabdevača i promenljivi troškovi transporta do svakog korisnika.

Problem se matematički može formulisati na sledeći način:

$$\min\left(\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i\right) \tag{5.1}$$

uz uslove:

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \text{za svakog korisnika } j \in J; \tag{5.2}$$

$$0 \le x_{ij} \le y_i \ \wedge \ y_i \in \{0,1\}, \quad \text{za svakog snabdevača } i \in I \text{ i svakog korisnika } j \in J; \qquad (5.3)$$

gde je x_{ij} količina robe koju snabdevač i isporučuje korisniku j. Niz binarnih promenljivih y_i označava da li je na potencijalnoj lokaciji i uspostavljen snabdevač ($y_i = 1$) ili nije ($y_i = 0$).

Neka je oznaka za skup postavljenih snabdevača $E = \{i \mid y_i = 1\}$, kardinalnosti $e = |E|$.

Primer 5.1. Datom problemu odgovara sledeći realni zadatak: U nekom naselju postoji n stambenih zgrada, i m potencijalnih lokacija za prodavnice (ili trafike, domove zdravlja, benzinske pumpe, kafane, servisne službe, ...). Potrebno je u datom naselju optimalno rasporediti prodavnice na neke od potencijalnih lokacija, tako da budu blizu stanovnicima naselja, ali uzimajući u obzir i troškove zakupa poslovnog prostora prodavnica. Najčešće su dati uslovi suprotstavljeni, pa lokacije u centru naselja koje su relativno najbliže svim stanovnicima, imaju visoke troškove zakupa poslovnog prostora, a lokacije na periferiji sa niskim troškovima zakupa, zahtevaju velike transportne troškove.

5.2 Načini rešavanja

Za prost lokacijski problem (SPLP) je poznato da pripada klasi NP-kompletnih problema. Jedan od dokaza datog tvrđenja može se naći u preglednom radu [Krr83]. Posledica toga je veliki broj objavljenih radova, u kojima su izloženi raznovrsni načini rešavanja, primenom metoda matematičkog programiranja i kombinatorne optimizacije.

5.2.1 Posebni slučajevi

Iako je dati problem NP-težak, u nekim posebnim slučajevima ga je moguće rešiti u polinomskom vremenu izvršavanja. Iscrpna analiza takvih slučajeva, uz prikaz načina rešavanja polinomske složenosti u tim slučajevima, je data u radovima [Aqe90], [Grs94], [DSi96] i [Ryu92]. Još jedan primer rešavanja SPLP specijalne strukture je dat u [Jon95].

5.2.2 Opšte metode

Neki od važnijih preglednih članaka iz ove oblasti su [Krr83], [Crn90] i [Gao94]. Pošto opis svih važnih doprinosa pri njegovom rešavanju izlazi izvan okvira ovog rada, samo žemo se kratko ćemo osvrnuti na neke od najpoznatijih i najefikasnijih metoda:

DUALOC algoritam čiji je autor Erlenkotter, i koji je detaljno opisan u [Erl78], je dugo vremena bio najefikasniji algoritam za rešavanje SPLP. Osnova za datu metodu je dualna formulacija pridruženog problema linearnog programiranja (LP dual) u skraćenoj formi, koji doprinosi prostoj implementaciji dualne metode penjanja (dual ascent) i odgovarajuće dualne metode poravnanja (dual adjustment). Ako rešenje dobijeno primenom tih dveju metoda nije optimalno, proces se nastavlja primenom metode grananja i ograničavanja (branch-and-bound). Po završetku izvršavanja algoritma dobijeno je optimalno rešenje. U velikom broju SPLP instanci manje dimenzije, optimalno rešenje se dobija već u prvoj iteraciji, odnosno bez potrebe za grananjem.

U radu [Gui88] je prikazano jedno poboljšanje Lagranževe dualne metode penjanja, korišćenjem Lagranževe relaksacije uz pomoć Benderovih nejednakosti, generisanih u toku rada. Takođe je dat način pri kome se, povezivanjem date metode sa dobrom heuristikom polaznog problema (primal heuristic), može smanjiti jaz između polaznog i dualnog problema (integrality gap).

Koristeći formulaciju karakterističnu za problem pokrivanja, u [Sim89] je prikazana jedna varijanta dualnog simplex algoritma (stramlined dual simplex method). Rezultati na standardnim instancama ovog problema ukazuju na superiornost dualnih metoda, čiji su najpoznatiji predstavnici efikasne subgradijentne i dualne metode ispravke.

U radu [Koe89], Körkel iz osnova modifikuje primalno-dualnu verziju Erlenkotter-ovog DUALOC algoritma za nalaženje optimalnog rešenja. Rezultati izvršavanja na SPLP instancama veće dimenzije, prikazani u tom radu, su 10-100 puta bolji u odnosu na DUALOC.

Tačno rešenje pridruženog dualnog problema linearnog programiranja, koristi se u metodi opisanoj u radu [Con90], za nalaženje relaksacije SPLP preko ortogonalnih projekcija.

Alves i Almeida su u [Alv92] izvršile poređenje rezultata četiri verzije algoritma simuliranog kaljenja (simulated annealing), u odnosu na neke dobro poznate heuristike za rešavanje SPLP. Kvalitet rešenja dobijen metodama simuliranog kaljenja je vrlo dobar, ali vreme izvršavanja je višestruko duže.

Holmberg u [Hlm95] koristi primalno-dualne metode zasnovane na dekompoziciji. U polaznom (primalnom) potproblemu fiksira neke promenljive, pri čemu relaksira neke uslove u dualnom potproblemu, ali fiksira njihove Lagrange-ove množioce. U takvim slučajevima je problem mnogo lakši za rešavanje, u odnosu na polaznu postavku problema. Rezultati prikazani u radu sugerišu da su neke kombinacije ovakvih pristupa, bolje u odnosu na Erlenkotter-ovu dualnu metodu penjanja.

Jedan od relativno uspešnih pokušaja hibridizacije Add-heuristike i dualnih metoda je prikazan u [Tch87] i [Tch88]. Iako se datim načinom rešavanja dobijaju približna (suboptimalna) rešenja koja mogu biti i nešto lošijeg kvaliteta kod instanci veće dimenzije, vreme izvršavanja je obično vrlo kratko.

Osim prethodno opisanih, neke od zanimljivih ideja za rešavanje ovog problema se mogu naći i u radovima [Crn82], [You96], [Mrc97], [Chd97] i [Gln98]. Iako samostalna primena ovih metoda najčešće daje lošije rezultate u odnosu na dualne metode, njihov potencijal je u eventualnoj hibridizaciji međusobno, sa dualnim metodama ili GA.

5.2.3 Metode pomoću genetskih algoritama

U [Kra96a] je opisana metoda za rešavanje SPLP pomoću GA. Primenjen je prost genetski algoritam , a vrednosna funkcija (objective value function) je vrlo jednostavno implementirana. Korišćena je veličina populacije od 20 jedinki, uz maksimalan broj od 1000 generacija. Jedina korekcija u odnosu na prost GA, je predstavljao direktan izbor najbolje jedinke u narednu generaciju. Pri tome je uvek čuvana najbolja jedinka, i time izbegnuta teorijska mogućnost degradacije kvaliteta postignutog rešenja. Pri ovako maloj veličini populacije i relativno malog maksimalnog broja generacija, postignuti su zapaženi rezultati na standardnim instancama do veličine 100×1000.

Početna implementacija zasnovana na prostom GA za rešavanje SPLP je u praksi pokazala dobre rezultate, ali ne i dovoljno dobre da može da se uporedi sa najboljim rezultatima dobijenim pomoću ostalih metoda. Zbog toga se pristupilo postepenom poboljšanju osnovne implementacije, koje je redom opisano u radovima [Kra98b], [Kra99a], [Kra99c], kao i u ovom poglavlju.

5.3 GA implementacija

5.3.1 GA reprezentacija

Za rešavanje problema (SPLP) izabrano je binarno kodiranje potencijalnih lokacija snabdevača. Svaki bit u genetskom kodu jedinke, čija je vrednost 1 označava da je na datoj lokaciji postavljen snabdevač, a vrednost 0 da nije. Genetski kod jedinke je alociran kao niz 32-bitnih reči, radi bržeg izvršavanja, pri primeni genetskih operatora.

Primer 5.2.

Za m=5, genetski kod jedinke **01011**|**00000001000000010001110101**
1. - 5. 6. - 32.

označava da:

- su na lokacijama **2, 4** i **5** su postavljeni snabdevači;
- na lokacijama **1** i **3** nisu postavljeni snbdevači;
- se ignorišu bitovi **6**-**32**.

Iz genetskog koda se direktno nalazi niz indikatora y_i (i=1, 2, ... , m) koji kazuju da li je na odgovarajućoj potencijalnoj lokaciji postavljen snabdevač ili ne. Pošto za snabdevača na određenoj lokaciji ne postoji ograničenje u kapacitetu lokacije i pošto je poznato na kojim su sve lokacijama postavljeni snabdevači a na kojima nisu (niz y_i), svaki korisnik može da izabere sebi najbližeg snabdevača. Primetimo da se, u slučaju kada su snabdevači fiksirani, minimizacijom troškova za svakog korisnika ujedno dobija i minimum ukupnih troškova.

5.3.2 Vrednosna funkcija

Vrlo važan deo koji garantuje brzu implementaciju funkcije za računanje vrednosti jedinke (vrednosne funkcije - objective value function) je izbor prave strukture podataka. Radi ubrzanja pri računanju vrednosne funkcije, koristimo dodatni memorijski prostor za memorisanje rednih brojeva potencijalnih lokacija. Za svakog korisnika lista indeksa potencijalnih lokacija je uređena u neopadajućem poretku troškova transporta.

Pri inicijalizaciji programa, formiramo listu indeksa za svakog klijenta, i to korišćenjem ugrađene *qsort()* funkcije. Pri tome je zauzeće memorije oko 50% veće, ali je nekoliko puta brže izvršavanje programa.

Zavisno od broja postavljenih lokacija za snabdevanje e i izračunate vrednosti $e_0 = c \cdot \sqrt{m}$, predloženo je korišćenje dve različite strategije za računanje vrednosne funkcije. Konstanta c je eksperimentalno određena na nivou 0.4 - 0.5, jer su tada dobijeni najbolji rezultati u praksi.

5.3.2.1 Računanje u slučaju e > e_0

Ako je *e* dovoljno veliko (e > e_0) u računanju vrednosti jedinke koristimo listu indeksa. Za svakog korisnika nalazimo najpovoljnijeg snabdevača, na sledeći način:

- Za datog korisnika nalazimo odgovarajuću listu indeksa;
- Pretražujemo datu listu do prve pojave y_i=1;
- Za snabdevanje tekućeg korisnika je izabran snabdevač sa lokacije *i*.

Primenom date procedure, pošto je svaka vrsta uređena u neopadajućem poretku troškova transporta izbrane su najpovoljnije lokacije za snabdevanje svakog od korisnika. Pošto je broj uspostavljenih lokacija za snabdevanje veliki (e > e_0), potrebno je samo nekoliko koraka za nalaženje prvog elementa koji zadovoljava y_i=1, u listi indeksa, pa je vreme izvršavanja vrednosne funkcije malo.

Nalaženje niza y_i iz genetskog koda date jedinke je vremenske složenosti O(m). Za svakog od *m* korisnika, potrebno je prosečno *m/e* koraka za pretraživanje liste indkesa, pri nalaženju najpogodnije lokacije za snabdevanje. Zbog toga je vremenska složenost vrednosne funkcije u ovom slučaju jednaka $O\left(m+n\cdot\frac{m}{e}\right)$.

5.3.2.2 Računanje u slučaju e ≤ e_0

Prethodna procedura daje dobre rezultate samo ako je *e* veliko, pa je niz *y* "gust" (sadrži mnogo elemenata jednakih jedan). U suprotnom, ako je niz *y* "redak", prethodna strategija daje loše rezultate, jer je potrebno mnogo više koraka pri pretraživanju liste indeksa, da bi pronašli element za koji je y_i=1.

Zbog toga, u datom slučaju, primenjujemo drugačiju strategiju za nalaženje najpovoljnijih lokacija za snabdevanje. Pri tome više ne koristimo listu indeksa, jer to nije celishodno, već formiramo niz *o* koji sadrži indekse nenultih članova niza *y* (odnosno elemente za koje je y_i=1*)*. Dakle, o_j = i, ako y_i = 1 predstavlja *j*-ti nenulti element po redosledu pojavljivanja u nizu *y*. Veličina niza *o* je mala, jer je niz *y* "redak", pa ima malo nenultih elemenata.

Pri nalaženju najpovoljnije lokacije za snabdevanje datog korisnika, pretražujemo samo niz *o*. Iako niz *o* nije uređen u nekom poretku (kao na primer lista indeksa), pa zahteva pretraživanje po svim članovima niza, on je male dužine, jer je *e* malo, što garantuje brzo izvršavanje. Konstrukcija niza *o* zahteva O(m) vremena, ali se ona izvršava samo jednom na početku, a koristi se *n* puta, pri nalaženju najpovoljnije lokacije snabdevanja za svakog korisnika.

Za pretraživanje niza *o* je potrebno *e* koraka, pa je tada ukupna vremenska složenost date vrednosne funkcije jednaka $O(m + n\cdot e)$.

5.3.2.3 Teorijska ocena složenosti

U ovom odeljku ćemo dati teorijsku ocenu složenosti celokupne funkcije za računanje vrednosti jedinke.

Teorema 5.1. U opštem slučaju vreme izvršavanja vrednosne funkcije nije veće od $O\left(m+n\cdot\sqrt{m}\right)$.

Dokaz: Za $e > e_0$ primenjujemo prvu strategiju, pa je

$$\frac{m}{e} < \frac{m}{e_0} = \frac{m}{c \cdot \sqrt{m}} = \frac{1}{c}\sqrt{m} \tag{5.4}$$

odnosno

$$m + n \cdot \frac{m}{e} < m + \frac{1}{c} \cdot \mathrm{n} \cdot \sqrt{m} \tag{5.5}$$

pa je složenost u najnepovoljnijem slučaju jednaka $O\left(m + n \cdot \sqrt{m}\right)$.

Ako je $e \le e_0$ imamo

$$m + n \cdot e \le m + n \cdot e_0 = m + c \cdot n \cdot \sqrt{m} \tag{5.6}$$

pa je složenost druge strategije u najnepovoljnijem slučaju jednaka $O\left(m + n \cdot \sqrt{m}\right)$, što je dobijeno i u prvom slučaju. ♦

Iako postoje teoretske mogućnosti za primenu algoritama za računanje vrednosne funkcije složenosti $O(m + n \cdot \log m)$ takve metode ne daju bolje rezultate u praksi. To možemo objasniti ako imamo u vidu da m u praksi ne prelazi nekoliko hiljada, pa je tada $\sqrt{m} \approx \log_2 m$.

5.3.3 Genetski operatori

5.3.3.1 Selekcija

Eksperimentisano je sa nekoliko različitih operatora selekcije, i najbolje rezultate pri rešavanju datog primera je pokazala selekcija zasnovana na rangu (rank-based selection). Razlike su posebno značajne pri poređenju sa prostom (rulet) selekcijom, što se generalno slaže sa ocenom datom u radu [BeD93a]. Za rešavanje datog problema je primenjeno linearno smanjenje rangova, sa korakom 0.012, od nivoa 2.5 za najbolju jedinku, do nivoa od 0.712 za najlošiju jedinku. Kao što je rečeno u drugom poglavlju, jedinka učestvuje na "ruletu" proporcionalno svom rangu, pri čemu se bira 50 novih jedinki u populaciji.

5.3.3.2 Ukrštanje i mutacija

Pri izvršavanju GA implementacije za rešavanje datog problema, uniformno ukrštanje je dalo nešto bolje rezultate u odnosu na ostale šeme ukrštanja, iako razlike u performansama, pri poređenju sa jednopozicionim, dvopozicionim i višepozicionim ukrštanjem nisu bile velike. Primenjen je nivo ukrštanja od $p_{cross} = 0.85$, uz verovatnoću razmene proizvoljnog mesta (bita) $p_{unif} = 0.3$, što znači da se približno 30% bitova razmenjuje između jedinki.

Prosta mutacija je implementirana pomoću Gausove (normalne) raspodele, radi bržeg izvršavanja. Nivo mutacije zavisi od veličine problema, i dat je formulom (5.7).

$$p_{mut} = \begin{cases} 0.01, & m \le 50 \\ \dfrac{1}{2m}, & m > 100 \end{cases} \tag{5.7}$$

Za razliku od ukrštanja, gde se variranjem parametara performanse menjaju relativno malo, pri variranju vrednosti nivoa mutacije, se dobijaju drastično drugačiji rezultati, naročito za SPLP instance veće dimenzije. Ovako izabran

nivo mutacije se pokazao kao najbolji kompromis između komponente istraživanja i komponente eksploatacije u GA pretrazi, korišćenih za rešavanje prostog lokacijskog problema.

5.3.4 Ostali aspekti

5.3.4.1 Generisanje početne populacije

Početna populacija se generiše na slučajan način. Vršeni su i eksperimenti sa inicijalizacijom GA pomoću heuristika. Time je dobijena bolja funkcija prilagođenosti jedinki populacije u prvoj generaciji, ali zbog manje početne raznovrsnosti genetskog materijala, gradijent rasta funkcije prilagođenosti u narednim generacijama je primetno manji. Pri tome se vrlo brzo praktično anuliraju prednosti bolje funkcije prilagođenosti u prvoj generaciji.

Pri generisanju početne populacije je primenjena ista učestanost bitova 0 i 1 u genetskom kodu ($p_{b0} = 1/2$, $p_{b1} = 1/2$). Zbog toga on sadrži približno isti broj 0 i 1, što se pokazalo pogodnim pri rešavanju datog problema.

5.3.4.2 Uklanjanje višestrukih jedinki iz populacije

Da bi se izbegla mogućnost preuranjene konvergencije, uklanjaju se sve višestruke pojave iste jedinke u populaciji. Uklanjanje se vrši implicitno, postavljanjem prilagođenosti date jedinke na nulu, čime ona gubi šansu da se pojavi u narednoj generaciji.

5.3.4.3 Politika zamene generacija

Za rešavanje datog problema je primenjena veličina populacije od 150 jedinki, i korišćena je elitistička strategija (elitist strategy), koja je detaljnije opisana u odeljku 2.3.3.3 . Pri tome u svakoj generaciji 2/3 populacije (100 jedinki sa najboljom funkcijom prilagođenosti) direktno prelazi u sledeću generaciju. Kao što je već rečeno, da date jedinke ne bi bile u povlašćenom položaju, umanjuje im se prilagođenost po formuli (2.9). One zatim, zajedno sa ostalim (nepovlašćenim) jedinkama učestvuju u izboru preostalih 50 jedinki (1/3) populacije u novoj generaciji.

Dati postupak se u ovom slučaju pokazao daleko bolji od standardnog generacijskog GA (generational GA), kao što se može videti i iz rezultata datih u tabeli 5.1 . Ne samo što se, u proseku, dobijaju bolji rezultati već stohastička greška manje utiče na performanse GA, pa su i rezultati izvršavanja uniformniji. To proizilazi iz činjenice da se najbolje jedinke čuvaju iz generacije u generaciju pa evolucioni pritisak nije previše jak. Pošto evolucioni pritisak neposredno deluje samo na relativno lošije jedinke, pri tome se povećava mogućnost njihovog poboljšanja, uz istovremeno čuvanje gena dobrih jedinki.

Primer 5.3. U tabeli **5.1** su dati rezultati izvršavanja GANP implementacije na MR1 instanci prostog lokacijskog problema. U prvom slučaju je primenjen generacijski GA sa populacijom od 50 jedinki, dok je u drugom slučaju polulacija sadržala 150 jedinki od kojih je 100 bilo elitnih. Da bi rezultati mogli da budu korektno upoređeni, u oba slučaja je GA izvršen 20 puta sa tačno 1000 generacija, i date su prosečne vrednosti.

Kao što se može videti u tabeli 5.1 primenom generacijskog GA u svakom izvršavanju je dobijeno rešenje sa greškom većom od 100% u odnosu na optimalno rešenje. Povećavanjem date populacije (50 jedinki) za novih 100 elitnih jedinki, vreme izvršavanja nije povećano (čak je nešto smanjeno kao

posledica keširanja), a u svakom izvršavanju je dobijeno optimalno rešenje. U velikom broju slučajeva je optimalno rešenje dobijeno već posle 300-500 generacija.

Tabela 5.1 Poređenje generacijskog GA i elitne strategije

Veličina populacije	Broj elitnih jedinki	Broj ponovljenih blokova (keširanje)	Srednje vreme izvršavanja (s)	Srednja greška	Minimalna greška
50	-	10.42	164.73	211.7%	125.3%
150	100	41.46	151.72	Opt	Opt

5.4 Rezultati

5.4.1 Ulazni podaci

Standardne SPLP instance manje dimenzije čija su imena *41-134, A-C* su direktno preuzeti sa Interneta, iz ORLIB kolekcije [BeJ96a]. Detaljnije o ORLIB kolekciji test primera se takođe može videti i u [BeJ90a]. Karakteristike ovih instanci datog problema se mogu videti u tabeli 5.2.

Tabela 5.2 Karakteristike SPLP instanci (ORLIB)

Imena instanci	Broj instanci u grupi	Dimenzija	Veličina datoteka na disku
41 - 44, 51, 61 - 64, 71-74	13	16×50	10 KB
81 - 84, 91 - 94, 101 - 104	12	25×50	16 KB
111-114,121-124,131-134	12	50×50	32 KB
A - C	3	100×1000	1.2 MB

Pošto su dati problemi relativno manje veličine, pa ne mogu realno predstaviti mogućnosti opisane GA implementacije, od strane autora ovog rada su generisane SPLP instance veće dimenzije. U svakoj od grupa *MO, MP, MQ, MR, MS* i *MT* je generisano po 5 instanci na slučajan način, koristeći sledeću proceduru:

- Zahtev svakog od korisnika b_i se bira kao slučajan ceo broj iz intervala $[b_{min}, b_{max}]$;
- Cene transporta jedinične količine robe se uzima kao slučajan realni broj iz intervala $[c_{min}, c_{max}]$, a zatim se dobija normalizovana cena transporta robe, množenjem sa količinom robe b_i ;
- Za svaku potencijalnu lokaciju $i \in I$, računa se $S_i = \sum_{j=1}^{n} c_{ij}$ koja predstavlja kumulativne troškove zadovoljenja svih korisničkih zahteva samo korišćenjem date lokacije;
- Na kraju, fiksne troškove f_i dobijamo inverznim skaliranjem kumulativnih troškova S_i u interval $[f_{min}, f_{max}]$ pomoću formule (5.8).

$$f_i = f_{\max} - \frac{(S_i - S_{\min}) \cdot (f_{\max} - f_{\min})}{S_{\max} - S_{\min}} \qquad (5.8)$$

Ulazni parametri za generator SPLP instanci i tipovi parametara se mogu videti u tabeli 5.3. Karakteristike generisanih instanci se mogu videti u tabeli 5.4.

Tabela 5.3 Tipovi parametara SPLP instanci

Opis parametra	Minimum	Maksimum	Tip
Cena transporta	c_{min}	c_{max}	real
Fiksni troškovi	f_{min}	f_{max}	real
Zahtev korisnika	b_{min}	b_{max}	integer

Tabela 5.4 Karakteristike generisanih SPLP instanci

Instance	Dimenzija	f	c	b	Veličina datoteka na disku
MO1-MO5	100×100	50 - 300	2 - 10	1 - 5	100 KB
MP1-MP5	200×200	100 - 600	2 - 10	1 - 5	400 KB
MQ1-MQ5	300×300	150 - 900	2 - 10	1 - 5	900 KB
MR1-MR5	500×500	100 - 600	0.5 - 5	1 - 5	2.4 MB
MS1-MS5	1000×1000	200 - 1200	0.5 - 5	1 - 5	9.5 MB
MT1-MT5	2000×2000	400 - 2400	0.5 - 5	1 - 5	35.3 MB

5.4.2 Eksperimentalni rezultati

Izvršavanje je obavljeno na PC kompatibilnom računaru sa procesorom AMD 80486 na 133 MHz sa 64MB osnovne memorije (64 MIPS, 21.8 Mflops). Pošto genetski operatori daju nedeterminističke rezultate, svaka instanca problema je izvršavana više puta. Radi bolje preglednosti, u tabeli 5.5 su dati samo prosečni rezultati u svakoj grupi instanci, a detaljni rezultati izvršavanja se mogu videti u dodatku C.

U tabeli 5.5 vidimo da je u svim izvršavanjima rezultat bilo optimalno rešenje (najbolje dobijeno rešenje), uz relativno kratko vreme izvršavanja, čak i za SPLP instance vrlo velike dimenzije.

Tabela 5.5 Rezultati GA

Instance	Broj izvrš.	Sred. br. gener.	Sred. vreme izvršavanja	Opt. (NDR)	Ostala rešenja
41 - 74	13×20	17.6	0.241	260	-
81 - 104	12×20	32.2	0.420	240	-
111 - 134	12×20	106.3	1.304	240	-
A - C	3×20	1 220	74.09	60	-
MO	5×20	123.7	2.84	100	-
MP	5×20	188.2	7.29	100	-
MQ	5×20	264.5	15.18	100	-
MR	5×20	423.6	41.95	100	-
MS	5×20	925.2	199.25	100	-
MT	5×20	1 806	919.78	100	-

5.4.3 Poređenje sa ostalim implementacijama

5.4.3.1 Karakteristike dualnih metoda

Od svih prethodnih metoda za rešavanje prostog lokacijskog problema u opštem slučaju, u praksi su do sada najbolje rezultate dale dualne metode ([Erl78], [Sim89], [Koe89], [Hlm95]). Karakteristike dualnih metoda:

- Optimalno rešavaju dati problem, korišćenjem metode grananja-i-ograničavanja;
- U opštem slučaju su eksponencijalne složenosti;
- Vrlo brzo rešavaju SPLP instance manje dimenzije;
- Vrlo lako implicitno odbacuju neperspektivne potencijalne lokacije. Čak i u slučaju SPLP instanci velike dimenzije, brzo dolaze do optimalnog rešenja, ako je broj suboptimalnih rešenja relativno mali;
- Vreme izvršavanja je ekstremno veliko, samo u slučaju problema vrlo velike dimenzije, sa malim brojem neperspektivnih potencijalnih lokacija i velikim brojem suboptimalnih rešenja.

Jedna efikasna implementacija za rešavanja datog problema, koja je javno dostupna, je Erlenkotter-ov DUALOC, detaljno opisan u radovima [Erl78] i [Krr83]. Da bi poređenje performansi bilo verodostojnije, na SPLP instancama su izvršene dve njegove varijante:

- Puna implementacija koja koristi metodu grananja-i-ograničavanja (BnB) za dobijanje optimalnog rešenja;
- Redukovana metoda (heuristika) koja sadrži samo metodu penjanja (dual ascent) i metodu poravnanja (dual adjustment), bez primene BnB.

5.4.3.2 DUALOC

Rezultati izvršavanja na istom računaru (AMD 80486), na istim SPLP instancama su dati u tabeli 5.6. Slično kao u tabeli 5.5, u svakoj koloni su redom predstavljeni:

- Imena instanci;
- Srednji broj iteracija pri izvršavanju;
- Srednje vreme izvršavanja;
- Kvalitet dobijenog rešenja - greška u odnosu na optimalno rešenje.

Tokom izvršavanja je daleko najbolje rezultate pokazala varijanta DUALOC-a pri kojoj se u početnom čvoru metode grananja i ograničavanja (branch-and-bound) vrši maksimalno dualno poboljšavanje, dok se u kasnijim čvorovima vrši jednostruko dualno poboljšavanje (maximum/one-pass dual improvement). Zbog toga su prikazani samo rezultati dobijeni primenom te varijante DUALOC algoritma.

Primetimo da DUALOC uvek na kraju izvršavanja daje optimalno rešenje. Međutim u nekim slučajevima je izvršavanje trajalo predugo, pa je prekinuto posle određenog vremena. U tom slučaju je data greška u odnosu na najbolje dobijeno rešenje (NDR), jer je ono dobijeno pomoću GA, pa nije bilo moguće dokazati njegovu optimalnost.

Tabela 5.6 Rezultati DUALOC-a

Instance	Srednji br.	Sr. vreme	Opt.	Ostala

	iteracija	izvršavanja		rešenja
41 - 74	1	<0.01	13	-
81 - 104	4	<0.01	12	-
111 - 134	3	<0.01	12	-
A - C	1 123	25.17	3	-
MO	26 268	32.54	5	-
MP	126 395	369.71	5	-
MQ	499 011	2913.7	5	-
MR	7 875 657	75 964	3	2 (8.05%)

5.4.3.3 Redukovani DUALOC

Pošto redukovana verzija ne sadrži grananje (BnB), vreme izvršavanja joj je vrlo kratko, ali daje samo suboptimalno rešenje. Rezultati su prikazani u tabeli 5.6 na isti način kao i u prethodnom slučaju.

Napomenimo da tokom izvršavanja na test primerima velike dimenzije, u nekim slučajevima nije bilo poznato optimalno rešenje (MR1, MR2, MS1-MS5). Zbog toga su, u tim slučajevima, rezultati redukovanog DUALOC-a upoređeni sa najboljim dobijenim rešenjima (NDR), koji su dobijeni tokom izvršavanja GA, ali nije mogla da bude dokazana njihova optimalnost.

Tabela 5.7 Rezultati DUALOC-a bez BnB

Instance	Srednji br. iteracija	Sr. vreme izvršavanja	Opt.	Ostala rešenja
41 - 74	1	<0.01	13	-
81 - 104	4	<0.01	12	-
111 - 134	3	<0.01	12	-
A - C	335.6	6.50	0	3 (5.74%)
MO	214.4	0.352	0	5 (9.74%)
MP	695.4	2.74	0	5 (9.63%)
MQ	1 450	9.80	0	5 (10.11%)
MR	2 886	42.61	0	5 (13.35%)
MS	9 625	348.8	0	5 (15.29%)

5.4.3.4 Analiza dobijenih rezultata

Kao što vidimo iz tabela 5.5 - 5.6 , obe verzije DUALOC-a su skoro trenutno (manje od 0.01 sekundi), u samo nekoliko iteracija rešile SPLP instance 41-134. GANP implementacija je takođe prilično uspešno rešila ove probleme, ali ipak je izvršavanje trajalo dosta duže (0.24 -1.3 sekundi) . Ovo je rezultat različite prirode algoritama koji su primenjeni. Pošto su rešavane SPLP instance male dimenzije, i efikasno odbačene još neke neperspektivne potencijalne lokacije, dualnom metodom penjanja (dual ascent) su praktično odmah dobijena optimalna rešenja. Za razliku od DUALOC-a, genetskom algoritmu je kao robusnoj metodi za pretraživanje bilo potrebno ipak neko vreme da stigne do (optimalnog) rešenja.

Pri rešavanju instanci A-C je DUALOC i dalje bio brži od GANP (25.17 sekundi prema 74.09), ali su razlike sada daleko manje nego u prethodnom slučaju.

Redukovani DUALOC je vrlo brzo rešio date instance (6.5 sekundi), ali je dobijeno rešenje bilo lošeg kvaliteta (prosečna greška 5.74% od optimalnog rešenja).

Pri rešavanju svih kasnijih instanci (MO-MS), redukovani DUALOC je dobijao rešenja vrlo lošeg kvaliteta (9.74% - 15.29%), što je nivo greške koji je neprihvatljiv za bilo kakvu praktičnu primenu. Primetimo da porastom dimenzije problema koji rešavamo, vreme izvršavanja redukovanog DUALOC-a mnogo brže raste u odnosu GANP, tako da se već MS instance sporije rešavaju pomoću redukovanog DUALOC-a u odnosu na genetski algoritam (uz ogromnu razliku u kvalitetu dobijenog rešenja). Zbog toga će u narednom delu biti poređene samo performanse originalnog DUALOC-a i GANP.

Oba algoritma (DUALOC i GANP) su optimalno rešila sve MO-MQ instance, ali je vreme izvršavanja genetskog algoritma 10-200 puta kraće. Razlika u brzini je još veća (oko 1800 puta) za instance iz klase MR i utisak je da dalje eksponencijano raste sa povećanjem dimenzije problema. Za neke od MR instanci (MR1 i MR2) DUALOC čak i nije dobio optimalno rešenje, jer je izvršavanje prekinuto posle određenog vremena. Pri tome je dobijeno rešenje lošije za oko 8% od NDR koje je dobijeno pomoću GANP.

Pokušaćemo da dobijene rezultate objasnimo i analiziramo imajući u vidu način na koji date metode (DUALOC i GANP) dolaze do rešenja. DUALOC primenjuje BnB pretraživanje, koje nalazi optimalno rešenje datog problema, ali je dati algoritam, u opštem slučaju, eksponencijalne složenosti. Ovakav pristup se pokazao vrlo uspešnim pri rešavanju SPLP instanci sa relativno malim brojem približnih (suboptimalnih) vrednosti, koje su bliske optimalnom rešenju. U tom slučaju se pretraga vrlo brzo usmerava samo na manji broj perspektivnih potencijalnih lokacija snabdevača, pa se relativno brzo dolazi do optimalnog rešenja. U suprotnom, ako je veliki broj približnih rešenja blizak optimalnom, tada skoro sve potencijalne lokacije imaju šansu da pripadaju optimalnom rešenju. Pošto se u tom slučaju ne može realno smanjiti pretraživački prostor, pretraga traje vrlo dugo, uz vrlo mali napredak tokom svake od BnB iteracija.

Genetski algoritam, nasuprot tome, predstavlja robustan način rešavanja NP-kompletnih problema, pa su zbog toga razlike u vremenu izvršavanja na malim i velikim SPLP instancama daleko manje nego kod dualnih metoda. Pri tome se uspešno rešavaju čak i instance problema sa velikim brojem približnih rešenja bliskih optimalnom. Ti slučajevi čak i pogoduju izvršavanju GA, jer se vrlo brzo nakon početka rada genetskog algoritma u populaciji pojavljuju, a zatim i preovlađuju, jedinke sa vrednostima bliskim optimalnom rešenju. Njihovom rekombinacijom se zatim vrlo lako poboljšavaju date vrednosti, i postupak obično dosta brzo konvergira u optimalnom rešenju.

Zauzeće memorijskog prostora, iako manje značajno od vremena izvršavanja, je ipak relativno bitan faktor pri rešavanju ovog problema. I u ovom segmentu rada je GANP implementacija pokazala dobre rezultate, jer je zauzeće memorije slično kao kod PDLOC-a, a oko 2 puta manje u odnosu na DUALOC. Na primer za MS instance dimenzije 1000×1000 GANP zahteva oko 12 MB memorije, dok DUALOC alocira 24 MB.

5.4.3.5 Ostale metode

Pri poređenju sa ostalim metodama, autor ovog rada je bio prinuđen na empirijsko vrednovanje performansi datih implementacija. Na osnovu informacija dostupnih iz literature, može se zaključiti da su najbolji algoritmi oko

1-2 reda veličine brži od DUALOC-a. Na osnovu toga, možemo očekivati da GA implementacija na MR, MS, MT i sličnim "teškim" SPLP instancama velike dimenzije, daje značajno bolje rezultate od ostalih implementacija.

Realizacije drugih metoda koje se pominju u literaturi (videti: [Dea85], [Aik85], [Gui88], [Bum89], [Koe89], [Con90], [Crn90], [Krr90], [Alv92], [Dea92], [Ryu92], [BeJ93], [Glv93], [Gao94], [DSi95], [Hlm95], [Tch95], [Wat96], [Hlm97a], [Tch97]) nisu javno dostupne. Stoga je teško napraviti poređenje GANP-a sa njima. Međutim, na osnovu opisa u literaturi, posredno se može nešto zaključiti i o ovim metodama. S obzirom da najbolji od njih daju rezultate bolje od DUALOC-a za 1 do 2 reda veličine, zaključujemo da za instance velike dimenzije opisane u ovom radu (MR-MT), GA daje bolje rezultate što se tiče vremena izvršavanja.

5.5 Rezultati paralelnog izvršavanja

Izvršavanje PGANP na SPLP instancama (kao i instancama ostalih problema) je izvršeno na lokalnoj mreži Računarske laboratorije Matematičkog fakulteta. Ona se sastoji od 8 računara 486/100MHz sa 16MB memorije pod Windows NT 3.51 operativnim sistemom. Mreža je tipa zvezde, sa maksimalnim ukupnim protokom od 10 Mbita.

Zbog zauzetosti Računarske laboratorije izvršavanje je obavljeno na samo 2 instance manje dimenzije (134 i A). PGA je izvršavan po 10 puta na svakoj od njih, i to korišćenjem redom 1, 2, 4 i 8 procesora. Za prosleđivanje jedinki između potpopulacija tokom izvršavanja paralelnog GA korišćena je virtuelna arhitektura hiperkocke.

Zbog relativno skromnih hardverskih mogućnosti (nedostatak operativne memorije), u nekim trenucima su pojedini računari prekidali rad celog sistema. Zbog toga pri izvršavanju na većem broju procesora (4 odnosno 8) nisu dobijeni rezultati u svakom od 10 izvršavanja. Zbog toga su u tabeli 5.8 date informacije samo o broju uspešnih izvršavanja.

Tabela 5.8 Rezultati PGA za SPLP

Instanca	Broj proc.	Broj izvrš.	Rešenje	Sred. br. gener.	Sred. vreme izvrš. (s)	Faktor ubrzanja
134	1	10	Opt	71.3	4.62	
	2	10	Opt	46.4	3.56	1.298
	4	8	Opt	47.875	3.515	1.314
	8	4	Opt	33	3.03	1.525
A	1	10	Opt	289	42.87	
	2	10	Opt	188.5	31.33	1.368
	4	10	Opt	188.8	31.98	1.340
	8	2	Opt	137	25.33	1.692

Analizom rezultata paralelnog izvršavanja na SPLP instancama datim u tabeli 5.8 se može utvrditi da u ovom slučaju povećavanjem broja procesora porast performansi nije blizak linearnom.

Jedan od razloga je odnos *vreme izvršavanja*/*vreme učitavanja podataka* koji je za ove instance relatvno mali (oko 5 puta pa čak i manje). Zbog toga se pri izvršavanju redovno dešavalo da neki procesori ulazne podatke dobiju skoro

u isto vreme kada GA završava rad. Time ovi procesori uspevaju da izvrše samo nekoliko generacija i time ne utiču dovoljno na poboljšavanje performansi PGA u meri u kojoj je to očekivano.

Drugi razlog za gorepomenutu pojavu su i relativno skromne performanse date računarske mreže. Na primer, veličina instance A na disku je oko 1.2 MB (za smeštanje u operativnoj memoriji je potrebno nešto manje). Zbog toga je u najboljem (teorijskom) slučaju za njeno fizičko prosleđivanje ka ostalih 7 procesora potrebno najmanje oko 10-tak sekundi. Time se može objasniti relativno kasno uključivanje nekih procesora u izvršavanje lokalnih GA, i njihov nedovoljni uticaj na poboljšanje performansi.

I pored ovih razloga, PGA je ipak u nekoj meri poboljšao performanse sekvencijalnog GA za rešavanje SPLP, a pri izvršavanju na paralelnim računarima boljih performansi instanci problema veće dimenzije moguće je očekivati još bolje rezultate.

6. PROBLEM DIZAJNIRANJA MREŽE NEOGRANIČENOG KAPACITETA

Problemi optimizacije na mrežama su izuzetno značajni u praksi, pogotovo u poslednje vreme. Problem dizajniranja mreže neograničenog kapaciteta (Uncapacitated Network Design Problem - UNDP) je jedan od baznih predstavnika u toj klasi, i vrlo je primenljiv u izboru topologije neke konkretne mreže. Iako ima primena u proizvodnji, distribuciji robe i saobraćaju, najznačajnije primene u poslednje vreme su telekomunikacije i računarske mreže.

Kao i prethodni problemi UNDP, takođe, pripada klasi NP-kompletnih problema (videti [Joh78], [Mag84], [Hlm86] i [Bla89]). Dokaz tog tvrđenja sledi svođenjem UNDP na problem određivanja Steiner-ovog stabla, za koji je već bilo poznato da pripada klasi NP-kompletnih problema. Ceo dokaz je vrlo detaljno opisan u [Hlm86].

6.1 Formulacija problema

Neka je dat skup artikala C, i orjentisani graf G = <N,A>, gde je N skup čvorova, a A skup grana koje povezuju čvorove tog grafa. Za svaki artikal $k \in C$ je dat polazni čvor $o(k)$, destinacija $d(k)$, i količina r_k koju treba transportovati. Za svaku granu grafa $(i,j) \in A$, ukoliko želimo da je koristimo, dati su fiksni troškovi njenog uspostavljanja f_{ij}, i troškovi transporta jedinične količine artikla k preko date grane c_{ij}^k.

Potrebno je odrediti koje grane treba uspostaviti, i na koji način organizovati transport svih artikala, tako da ukupni troškovi budu minimalni. U ukupnim troškovima učestvuju troškovi uspostavljanja izabranih grana i troškovi transporta svakog artikla. Transport nekog artikla preko određene grane je moguć samo ako je grana prethodno uspostavljena.

Pošto je neograničen kapacitet svake grane, cela količina artikla r_k može biti poslata duž istog puta. Zbog toga se, bez ograničenja opštosti, problem može skalirati pa se transportni troškovi c_{ij}^k. množe sa r_k, čime se količina artikla normalizuje i postaje $r_k = 1$.

Problem se matematički može formulisati na sledeći način:

$$min \left(\sum_{k \in C} \sum_{(i,j) \in A} c_{ij}^{\ k} \cdot x_{ij}^{\ k} + \sum_{(i,j) \in A} f_{ij} \cdot y_{ij} \right) \qquad (6.1)$$

uz uslove

$$\sum_{j:(i,j) \in A} x_{ij}^{\ k} - \sum_{i:(j,i) \in A} x_{ji}^{\ k} = b_i^k \qquad \forall i \in N, \forall k \in C \qquad (6.2)$$

$$0 \le x_{ij}^{k} \le y_{ij} \qquad \forall (i,j) \in A,\ \forall k \in C \tag{6.3}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{6.4}$$

gde je x_{ij}^{k} količina artikla $k \in C$ koja se transportuje preko grane $(i,j) \in A$. Promenljiva y_{ij} kazuje da li je grana (i,j) uspostavljena, a b_i^k je protok artikla k kroz čvor i, kao što možemo videti iz formule (6.5):

$$y_{ij} = \begin{cases} 1, & ako\ je\ grana\ (i,j)\ uspostavljena \\ 0, & nije\ uspostavljena \end{cases} \qquad b_i^k = \begin{cases} 1, & i = o(k) \\ -1, & i = d(k) \\ 0, & inace \end{cases} \tag{6.5}$$

Problem može poslužiti i za poboljšavanje postojećeg dizajna mreže, tako što se za sve već uspostavljene grane postavi $f_{ij} = 0$, što automatski u rešenju za te grane generiše $y_{ij} = 1$.

6.2 Načini rešavanja

Problemi dizajna mreže su posebna klasa optimizacionih problema, a u poslednje vreme su postali predmet posebnog proučavanja, pa se kao posledica toga pojavio veći broj radova iz te oblasi. Klasifikacija datih problema se može naći u [Tnl83] i [Mag84]. U ovoj literaturi je dat i pregled dotadašnjih metoda korišćenih za njihovo rešavanje. Kasnije se istraživanje dalje intenziviralo pa je korišćen širok spektar metoda za rešavanje problema iz ove oblasti. U radovima [Mig88] i [Ers92] se može naći pregled nekih od ovih metoda. Pri rešavanju nekih od problema dizajna mreže i ostalih sličnih problema, su korišćeni i genetski algoritmi.

6.2.1 Korišćenje GA za rešavanje mrežnih problema

U radu [Pal94] je razvijena posebna tehnika za kodiranje stabla (tree) od strane GA. Ovaj način je zatim iskorišćen pri rešavanju jednog od mrežnih problema pomoću genetskih algoritama. Više informacija o ovom načinu rešavanja i o karakteristikama datog problema je dato u [Pal95]. GA je primenjen (videti [Cox91]) i za rešavanje problema dinamičkog rutiranja u posebnoj vrsti telekomunikacionih mreža.

6.2.2 Postojeće metode za rešavanje UNDP

Ovde ćemo spomenuti samo neke metode, koje su davale rešenja relativno dobrog kvaliteta i za probleme veće dimenzije.

U radu [Mag86] je primenjeno Bender-ovo razlaganje (Benders decomposition) kao osnova za rešavanje UNDP. Samostalna primena ovog metoda nije dala očekivane rezultate, zbog relativno velikog broja novih uslova koji su nastali pomoću Bender-ovih sečenja (Benders cuts). Zbog toga je ovaj metod (koji inače daje optimalno rešenje) dodatno poboljšan korišćenjem heuristika na rešenje dobijeno pomoću osnovne metode u početnoj i kasnijim iteracijama. Iako je sličan pristup primenjen i u još nekim radovima ([Bas87] i [Mig88]), pokazalo se da su nedostaci samo ublaženi, a ne i otklonjeni pa su rezultati nešto lošiji od direktnog pristupa pri metodi grananja i sečenja (Branch-and-Bound - BnB).

Unakrsno razlaganje (cross decomposition) je takođe primenjivano za rešavanje datog problema, kao kombinacija Bender-ovog razlaganja i Lagrange-ove relaksacije. Ova metoda je posebno razvijena za rešavanje problema mešovitog celobrojnog programiranja (Mixed Integer Programming - MIP) , i detaljno opisana u radu [VRo83]. U [Hlm90] je dokazano da dati postupak konvergira ka rešenju u konačnom broju koraka. Međutim, pokazalo se da iako nešto bolji od prethodne metode (Bender-ovog razlaganja), pri rešavanju UNDP ipak zadržava neke njene nedostatke. Pomenimo i neka njena unapređenja: metoda unakrsnog razlaganja pomoću srednje vrednosti (mean value cross decomposition) opisanu u [Hlm92] i hibridizacija unakrsnog razlaganja i uopštene metode opisane u [Kor65].

Jedna od značajnih heuristika za rešavanje UNDP je i dualna metoda penjanja (dual ascent), opisana u radu [Bla89]. Ona vrlo brzo daje rešenje relativno dobrog kvaliteta (greška 1% - 4% od opitmalnog rešenja), ali dobijeno rešenje ne može dalje da se poboljšava. Pokušaji da se data metoda uklopi kao subgradijentna u neku drugu metodu nisu dali očekivane rezultate (videti [Hel94] i [Hlm97b]).

Lagranževa heuristika koja je uspešno korišćena za rešavanje lokacijskih problema ([Hlm97a]), prilagođena je datom problemu i primenjena uz BnB. Dati pristup otklanja neke nedostatke prethodnih metoda pa su dobijeni odlični rezultati pri izvršavanju na UNDP instancama velike dimenzije (70 čvorova, 1000 grana i 138 artikala), koji su optimalno rešeni.

Osim navedenih, pomenimo i još neke metode opisane u sledećim radovima: [Lin82], [Tar83], [Bla87], [Brme95] i [Grö95]. Za rešavanje UNDP se mogu koristiti i programski paketi za rešavanje opštijeg problema mešovitog celobrojnog programiranja. Napomenimo da su dati programski paketi projektovani za proizvoljan problem mešovitog celobrojnog programiranja, pa ne uključuju specifične karakteristike ovog problema, što se ogleda u prilično lošijim performansama. Njihova prednost je u opštoj primeni na bilo koji problem iz date klase, gde je potrebno zadati samo funkciju koja se optimizuje i uslove zadatka, bez potrebe za dodatnim programiranjem. Još neke metode za rešavanje problema dizajniranja mreža i ostale korisne informacije se mogu naći i u radovima [Hlm91], [Hei92] i [Sas97].

6.3 GA implementacija

6.3.1 Kodiranje i vrednosna funkcija

Ovaj problem formulisan kao (6.1) - (6.4), ima ukupno $(|C|+1) \cdot |A|$ promenljivih (y_{ij} i $x_{ij}^k : (i,j) \in A, k \in C$), i u takvom obliku nije pogodan za primenu GA na UNDP instancama veće dimenzije. Međutim, broj promenljivih potrebnih za kodiranje se može smanjiti, na sledeći način. Ukoliko fiksiramo elemente y_{ij}, dati problem se svodi na $|C|$ problema nalaženja najkraćeg puta u grafu. U tom slučaju preostale promeljive x_{ij}^k možemo generisati relativno lako, algoritmom polinomske složenosti za nalaženje najkraćeg puta u grafu. Na taj način se kodiraju samo elementi niza y_{ij} $(i,j) \in A$, što je mnogo pogodnije za primenu genetskog algoritma, jer su oni binarne vrednosti ($y_{ij} \in \{0,1\}$).

Zbog svega prethodno navedenog, svaki bit u genetskom kodu jedinke predstavlja odgovarajuću vrednost za niz y_{ij}. Na taj način je generisan podgraf $G' = <N, A'>$ grafa G, gde je $A' = \{(i,j) \mid (i,j) \in A \wedge y_{ij} = 1\}$. On predstavlja podgraf

čije su grane uspostavljene, i preko kojih se mogu transportovati artikli. Na datom redukovanom podgrafu G' se za svaki artikal $k \in C$ primenjuje algoritam za nalaženje najkraćeg puta, i na taj način se rekonstruišu elementi x_{ij}^k.

Za nalaženje najkraćeg puta u grafu je primenjen Dijkstra-in algoritam, prvi put opisan u [Dij59], a može se naći u skoro svim knjigama iz oblasti algoritama. Koristeći neke posebne strukture podataka (primer jedne od njih je detaljno opisan u [Fre84]), postoje modifikovane implementacije Dijkstra-inog i Bellman-ovog algoritma ([Blm58]) koje nešto brže rešavaju problem nalaženja najkraćeg puta u grafu. Međutim, nijedna od tih modifikacija nije apsolutno najbolja, već je svaka od njih pogodna za neku posebnu klasu instanci. Opisi nekoliko ovakvih metoda su dati u [Joh77], [Gll88] i [GolA93], a njihova uporedna analiza i rezultati izvršavanja na raznim tipovima grafova se mogu videti u radu [Che93].

U tom radu možemo uočiti da su razlike u vremenima izvršavanja značajne tek za instance vrlo velike dimenzije (grafovi sa nekoliko hiljada čvorova), što višestruko prevazilazi dimenzije UNDP instanci koje su rešavane u ovom radu. Zbog toga je u ovom radu, pri rešavanju UNDP problema, primenjivana osnovna varijanta Dijkstra-inog algoritma, a u budućnosti je moguće unapređenje korišćenjem neke od modifikacija.

Teorema 6.1. U opštem slučaju vreme izvršavanja vrednosne funkcije nije veće od $O(|A| + |C| \cdot |N|^2)$.

Dokaz: Za dobijanje niza y_{ij} iz genetskog koda je potrebno $|A|$ koraka. Posle toga je potrebno još $|C|$ puta primeniti Dijkstra-in algoritam za nalaženje najkraćeg puta, koji je vremenske složenosti $O(|N|^2)$, pa je ukupno vreme izvršavanja najviše $O(|A| + |C| \cdot |N|^2)$ ♦

6.3.2 Genetski operatori

6.3.2.1 Selekcija

Kao i u primeni na prethodni problem (SPLP) zbirno su najbolji rezultati dobijeni korišćenjem selekcije bazirane na rangu. Takođe su primenjeni i isti parametri datog selekcionog operatora (linearno smanjivanje rangova od nivoa 2.5 za najbolju do 0.712 za najlošiju jedinku sa korakom 0.012). Međutim, za razliku od prethodnog problema, vrlo su male razlike pri izvršavanju u odnosu na rezultate turnirske selekcije. Čak je turnirska selekcija dala bolje rezultate na nekim UNDP instancama, ali su ukupni rezultati korišćenjem selekcije bazirane na rangu zbirno ipak nešto bolji.

6.3.2.2 Ukrštanje i mutacija

Operator uniformnog ukrštanja je i u ovom slučaju pokazao najbolje performanse pri izvršavanju, mada su i dalje male razlike u odnosu na druge načine ukrštanja. Zadržan je nivo ukrštanja $p_{cross} = 0.85$ i verovatnoće razmene bitova $p_{unif} = 0.3$.

Primenjena je i dalje prosta mutacija implementirana korišćenjem Gausove (normalne) raspodele, ali je nivo mutacije malo izmenjen i dat je pomoću formule:

$$p_{mut} = \frac{1}{2 \cdot |A|} \tag{6.6}$$

Kao i u prethodnom slučaju dobar izbor nivoa mutacije značajno utiče na performanse celog genetskog algoritma.

6.3.2.3 Generisanje početne populacije

I dalje se kao najpovoljniji izbor pokazalo generisanje početne populacije na slučajan način, iako su vršeni eksperimenti i sa drugim pristupom, gde se delimično ili potpuno početna populacija generiše pomoću raznih heuristika.

Ranije je, za rešavanje SPLP, korišćeno generisanje početne populacije sa istom učestanošću 0 i 1 u genetskom kodu. Takav pristup je bio dobar pri izvršavanju nekih UNDP instanci, ali je kod drugih bio vrlo loš. U nekim takvim slučajevima je GA prekidao izvršavanje već u prvoj generaciji, jer nije postojala ni jedna korektna jedinka. U drugim slučajevima je u početnoj generaciji bilo jedna ili samo nekoliko korektnih jedinki, pa je GA počeo sa izvršavanjem, ali je uz veoma smanjenu raznolikost genetskog materijala vrlo brzo nastupila preuranjena konvergencija u suboptimalnom rešenju vrlo lošeg kvaliteta.

Uzrok za ovu anomaliju u izvršavanju GA je u činjenici da su UNDP instance uglavnom retki grafovi, sa relativno malim odnosom grana i čvorova (oko 3:1). Pri generisanju početne populacije sa istom učestanošću 0 i 1, indukovani podgraf G' sadrži isti broj čvorova kao i polazni graf ali samo približno 50% njegovih grana. Iako je graf G povezan (dati uslov je ispunjen pri generisanju instanci), to vrlo često ne važi za njegov indukovani podgraf G', pošto je kod njega odnos grana i čvorova samo oko 1.5:1, a nekada i manje. Ako G' nije povezan velika je verovatnoća da polazni $o(k)$ i odredišni čvor $d(k)$ za neki artikal $k \in C$ pripadaju različitim komponentama povezanosti, i da ne postoji put između njih u indukovanom podgrafu G'. U tom slučaju je jedinka nekorektna, a ako takve jedinke u populaciji preovlađuju, javlja se gorepomenuta anomalija.

U takvim slučajevima se pokazalo neophodnim generisanje početne populacije sa različitim učestanostima 0 i 1 u genetskom kodu. Vrlo je efikasno implementirana varijanta sa učestanostima $p_{b0} = 1/4$, $p_{b1} = 3/4$ i $p_{b0} = 1/8$, $p_{b1} = 7/8$. Moguće su i inverzne sheme ($p_{b0} = 3/4$, $p_{b1} = 1/4$ i $p_{b0} = 7/8$, $p_{b1} = 1/8$), ali njihova upotreba do sada nije bila neophodna.

Korišćenjem vrednosti $p_{b0} = 3/4$ za generisanje početne populacije indukovani podgraf G' sadrži prosečno 50% više grana u odnosu na osnovnu varijantu. To se u praksi pokazalo dovoljnim da indukovani graf G' bude povezan, u slučaju onih instanci koje nisu mogle biti rešavane osnovnom varijantom. Time je većina jedinki u početnoj populaciji korektna, pa je u potpunosti izbegnuta gorepomenuta anomalija, i sprečena mogućnost preuranjene konvergencije kod ovakvih UNDP instanci.

6.3.3 Ostali aspekti

GA primenjen sa elitističkom strategijom je i pri rešavanju UNDP postigao dobar kompromis između komponenti istraživanja novih regiona pretrage i iskorišćenja već dobijenih dobrih jedinki. Pri tome populacija sadrži 150 jedinki od čega 2/3 populacije (100 jedinki) direktno prlazi u narednu generaciju, a na preostali deo (1/3 populacije što predstavlja 50 jedinki) se primenjuju genetski operatori selekcije, ukrštanja i mutacije.

Na isti način kao i ranije uklanjanje višestruke pojave jedinki u populaciji doprinosi većoj raznolikosti genetskog materijala i sprečavanju preuranjene konvergencije.

6.4 Rezultati

6.4.1 Ulazni podaci

Pošto ne postoji biblioteka UNDP instanci dostupna preko Interneta, a pokušaji da se one dobiju ličnim kontaktima nisu bili uspešni, izvršeno je njihovo generisanje (na slučajan način) od strane autora ovog rada.

Imajući u vidu teorijske aspekte ovog problema za svaku instancu je generisan povezan graf, što garantuje da će dati zadatak imati rešenje. Iako je moguće da postoji rešenje a da graf G ne bude povezan, ovakvi slučajevi nisu razmatrani pri generisanju instanci, jer oni ne predstavljaju suštinu samog problema, i nisu uobičajeni u praksi.

Osnovne karakteristike generisanih UNDP instanci se mogu videti u tabeli 6.1, a postupak generisanja je sličan postupku generisanja SPLP instanci. Takođe se prvo na slučajan način iz zadatog intervala generišu svi troškovi transporta (c_{ij}^k), a zatim se na osnovu njih obrnutim skaliranjem u zadati interval generišu fiksni troškovi. Međutim za razliku od prethodnog problema gde se skaliranje vrši na osnovu zbira troškova transporta po svakoj potencijalnoj lokaciji, ovde se računa zbir troškova transporta (c_{ij}^k) po svakom artiklu $k \in C$. Tada se njihovim obrnutim skaliranjem dobijaju fiksni troškovi f_{ij}.

Tabela 6.1 Karakteristike generisanih UNDP instanci

Instance	Dimenzija C, N, A	f	c	b	Veličina datoteka na disku
MA	10×5×15	2-6	1-3	3-4	2.5 KB
MB	10×10×32	2-6	1-3	3-4	5 KB
MC	10×15×50	2-6	1-3	3-4	7.7 KB
MD	10×20×70	2-6	1-3	3-4	10.7 KB
ME	10×30×120	2-6	1-3	3-4	18.2 KB

6.4.2 Eksperimentalni rezultati

Izvršavanje je obavljeno na istom PC kompatibilnom računaru kao i u slučaju prethodnog problema (AMD 80486 na 133 MHz sa 64MB osnovne memorije). Međutim zbog tehničkih problema koji su se u međuvremenu javili (otkazala je keš memorija na matičnoj ploči) performanse ovog računara su smanjene za oko 30-40%. Kao i ranije svaka instanca problema je izvršavana više puta, zbog toga što genetski operatori daju nedeterminističke rezultate. Radi bolje preglednosti, u tabeli 6.2 su takođe dati samo prosečni rezultati u svakoj grupi instanci, a detaljni rezultati izvršavanja se mogu videti u dodatku C.

Tabela 6.2 Rezultati GA za UNDP

Instance	Broj izvrš.	Sred. br. gener.	Sred. vreme izvršavanja	NDR	Ostala rešenja
MA	5×20	25.8	0.474	100	-
MB	5×20	99.6	2.580	100	-
MC	5×20	141.2	4.844	100	-
MD	5×20	325.2	11.98	99	1 (0.534%)
ME	5×20	2 267	91.47	49	51 (1.53%)

Kao što se može videti iz tabele 6.2 u svim izvršavanjima za instance MA-MC dobijeno je isto rešenje (NDR). Pri izvršavanju MD instanci samo jednom je dobijeno rešenje lošije od NDR, a već pri primeni na ME instance veće dimenzije ovakva rešenja su dobijena u oko 50% slučajeva. Pri tome možemo uočiti da osim dobijanja rešenja lošijeg kvaliteta raste srednji broj generacija, kao i odgovarajuće vreme izvršavanja, u odnosu na instance manjih dimenzija.

I pored toga možemo zaključiti da su dobijeni relativno dobri rezultati i da je GA pogodna metoda za rešavanje ovog problema (UNDP). Uz manje modifikacije i eventualnu hibridizaciju sa drugim metodama, možemo očekivati još bolje rezultate.

6.5 Rezultati paralelnog izvršavanja

Kao i u slučaju prethodnog problema i izvršavanje PGANP na UNDP instancama je izvršeno na istoj lokalnoj mreži. Takođe je Izvršavanje obavljeno na samo 2 instance manje dimenzije (MA1 i MC1). PGA je na isti način izvršavan po 10 puta na svakoj od njih, korišćenjem redom 1, 2, 4 i 8 procesora i takođe primenom virtuelne arhitekture hiperkocke. Jedini izuzetak je instanca MC1 koja je sa 4 procesora izvršavana 11 puta (u .BAT datoteci je omaškom ostavljeno jedno izvršavanje više). Kao i pri izvršavanju SPLP instanci, dolazilo je do tehničkih problema, pa je notiran samo broj uspešnih izvršavanja, i njihovi rezultati su predstavljeni u tabeli 6.3 .

Tabela 6.3 Rezultati PGA za UNDP

Instanca	Broj proc.	Broj izvrš.	Rešenje	Sred. br. gener.	Sred. vreme izvrš. (s)	Faktor ubrzanja
MA1	1	10	NDR	25.7	1.56	
	2	10	NDR	14.8	1.15	1.357
	4	5	NDR	12.6	1.07	1.458
MC1	1	10	NDR	367.8	26.33	
	2	10	NDR	234.8	19.76	1.332
	4	11	NDR	276.18	23.91	1.101
	8	8	NDR	104.38	9.99	2.636

Analizom rezultata paralelnog izvršavanja na UNDP instancama datim u tabeli 6.3 se može utvrditi da je u ovom slučaju povećavanjem broja procesora poboljšanje performansi nešto bolje u odnosu na prethodni problem.

To se može objasniti činjenicom da je odnos *vreme izvršavanjalvreme učitavanja podataka* prilično veći nego kod prethodnog problema. To se moglo videti i pri izvršavanju, jer su procesori uspeli da u toku rada GA izvrše sličan broj generacija na svojim lokalnim potpopulacijama. Na taj način pošto su svi procesori aktivno učestvovali u radu GA, svi su skoro podjednako uticali na poboljšanje performansi paralelnog GA, pa su i dobijeni bolji rezultati.

Relativno skromne performanse date računarske mreže su i dalje bile ograničavajući faktor, ali zbog manje relativne veličine instanci (u odnosu na dimenziju), nisu toliko uticale na degradaciju ukupnih performansi paralelne GA implementacije. Kao i kod prethodnog problema, i u ovom slučaju je moguće

očekivati da rezultati budu još bolji pri izvršavanju na paralelnim računarima boljih performansi instanci problema veće dimenzije.

7. PROBLEM IZBORA INDEKSA

Problem izbora indeksa (Index selection problem - ISP) je od suštinske važnosti pri dizajniranju baza podataka. Potrebno je izabrati neke od mogućih indeksa, tako da njihovim kreiranjem i kasnijim korišćenjem, minimiziramo vreme odziva sistema za upravljanje bazom podataka. Detaljan opis problema i analiza svih aspekata njegovog uticaja na rad samog sistema za upravljanje bazom podataka izlazi izvan okvira ovog rada. Zbog toga ćemo u ovom poglavlju dati samo najosnovnije informacije o samom problemu i osnovnim aspektima njegovog rešavanja u okviru kombinatorne optimizacije, a detaljnije o njegovoj primeni na baze podataka se može naći u radovima [Fin88] i [Cap95a]. Dokaz da ISP pripada klasi NP-kompletnih problema je detaljno opisan u [Com78].

7.1 Formulacija problema

Neka je $N = \{1, 2, ..., n\}$ skup svih indeksa, $P = \{1, 2, ..., p\}$ skup svih korisnih konfiguracija sastavljenih od datih indeksa, i $M = \{1, 2, ..., m\}$ skup svih upita neke baze podataka. Za postavljanje nekog indeksa $j \in N$ potrebno je utrošiti određeno vreme $f_j > 0$. Ukoliko su postavljeni svi indeksi u nekoj konfiguraciji $k \in P$, tada se za izvršavanje upita $i \in M$ može uštedeti vreme $g_{ik} \geq 0$. U praksi je najčešće za veliki broj parova (i, k) ušteda $g_{ik} = 0$, što se može objasniti time da određena konfiguracija ima uticaja samo na neke upite iz skupa M.

Potrebno je postaviti neke od indeksa, tako da ukupno vreme izvršavanja svih upita sistema za upravljanje bazom podataka bude minimalno, odnosno da ukupna ušteda vremena bude maksimalna. Problem se matematički može iskazati pomoću formula (7.1) - (7.3) koje detaljnije opisuju sve uslove koje treba rešenje da ispunjava:

$$max \left(\sum_{i \in M} \sum_{k \in P} g_{ik} \cdot x_{ik} - \sum_{j \in N} f_j \cdot y_j \right) \tag{7.1}$$

uz uslove

$$\sum_{k \in P} x_{ik} \leq 1 \qquad \forall i \in M \tag{7.2}$$

$$x_{ik}, y_j \in \{0,1\} \qquad \forall i \in M, \forall j \in N, \forall k \in P \tag{7.3}$$

Rešenje je zadato preko sledećih promenljivih, čiji detaljniji opis možemo videti u (7.4):

- y_{jj} označava da li je indeks j postavljen;

- x_{ik} daje informaciju o tome da li su postavljeni svi indeksi konfiguracije k što utiče na uštedu pri upitu i.

$$y_j = \begin{cases} 1, & ako\ je\ indeks\ j\ postavljen \\ 0, & ako\ nije\ postavljen \end{cases} \quad x_{ik} = \begin{cases} 1, & \text{postavljeni svi indeksi konf. k pri upitu i} \\ 0, & inace \end{cases} \tag{7.4}$$

7.2 Načini rešavanja

Dve metode za rešavanje problema selekcije indeksa (ISP) su date u radu [Cap95a]. Jedna je heuristika zasnovana na Lagrange-ovoj dekompoziciji pogodno definisanog potproblema ranca. Ona je primenjena na instancama datog problema velike dimenzije (m, n i p su nekoliko hiljada), ali je rešenje u nekim slučajevima bilo lošeg kvaliteta. Druga metoda je koristila ISP formulaciju preko 0-1 celobrojnog linearnog programiranja gde je korišćena metoda grananja i ograničavanja (Branch-and-Bound - BnB) zasnovana na LP relaksaciji, koja daje optimalno rešenje. Prezentirani su rezultati izvršavanja na UNDP instancama gde su m, n i p su nekoliko stotina.

U [Cap95b] je prikazano jedno poboljšanje prethodne metode pomoću tehnike grananja i odsecanja (Branch-and-Cut - BnC), pri čemu je ISP formulisan kao problem pakovanja skupa (set packing problem) . U tom slučaju se matematički model ovog problema može izraziti isključivo preko nejednakosti oblika klike (clique inequalities), čiji je broj polinomske složenosti u odnosu na veličinu datog problema. Na taj način je gornja granica problema dobijena rešavanjem LP relaksacije u prethodnom pristupu, poboljšana korišćenjem Chvávtal Gomory-jevim pristupa na nejednakosti klike. Ova metoda daje optimalno rešenje, a primenjena je na UNDP instance dimenzije do m = 100, n = 100 i p = 1000.

Osim ovih relatino novijih radova, neke od korisnih ideja za rešavanje ovog problema možemo naći i u radovima [Ip83], [Bon85], [Hat85], [Bar90] i [Gla90].

7.3 GA implementacija

Pošto je broj nenula vrednosti g_{ik} ($i \in M$, $k \in P$) relativno mali u odnosu na dimenziju matrice, memorišu se samo nenulte vrednosti. To se za svaki upit i postiže pomoću tri promenljive: koliko je vrednosti $g_{ik} \neq 0$ za taj upit, niz tih vrednosti i niz njihovih k indeksa. Na taj način se postiže i brže izračunavanje vrednosne funkcije, jer se pretraživanje ne vrši preko svih vrednosti matrice g, već samo preko vrednosti koje mogu učestvovati u rešenju ($g_{ik} \neq 0$). Sličan zapis se koristi i kod memorisanja sadržaja konfiguracija, gde se za svaku konfiguraciju beleži koliko indeksa sadrži i svi njihovi redni brojevi.

7.3.1 Kodiranje i vrednosna funkcija

Pri rešavanju datog problema je izabrano binarno kodiranje niza postavljenih indeksa y. Svaki bit u genetskom kodu jedinke, čija je vrednost 1 označava da je indeks postavljen (y_j = 1), a vrednost 0 da nije (y_j = 0). Pošto je iz genetskog koda direktno očitana vrednost niza y, na osnovu njega pronalazimo koje su konfiguracije aktivne (svi njihovi indeksi su aktivni).

Za svaki upit se nalazi ona aktivna konfiguracija koja mu najbolje odgovara, odnosno kada je ušteda vremena najveća (maksimalno g_{ik}). Tada samo

izračunamo zbir takvih najvećih g_{ik} vrednosti po svim upitima, i oduzmemo fiksna vremena za indekse koje postavljamo.

Ceo proces nalaženja vrednosti jedinke se može ubrzati ako pri inicijalizaciji parametara problema uredimo vrednosti g (za svaki indeks posebno) u nerastućem poretku. U tom slučaju se nalaženje najpogodnije aktivne konfiguracije za svaki upit, svodi na nalaženje prve aktivne konfiguracije, ako takva postoji. Pošto su odgovarajući g koeficijenti uređeni, prva nađena aktivna konfiguracija ima ujedno i maksimalnu vrednost.

Ako je dobijena vrednost jedinke negativna, takvo rešenje odbacujemo, nijedan od indeksa ne postavljamo ($y_j = 0$, za sve $j \in N$). Pošto u tom slučaju nijedan indeks nije aktivan, a takođe ni jedna konfiguracija, vrednost jedinke je 0, što je bolje od prethodnog negativnog rešenja.

Globalna shema izvršavanja vrednosne funkcije, gde su neki manje važni aspekti pojednostavljeni, se može videti na slici 7.1, a teorijska procena vremena izvršavanja te funkcije je formulisana u teoremi 7.1 .

```
for (j = 0; j <n; j++) y_j = Gen[ j ];
for (k = 0; k <p; k++)
     if( Aktivni_svi_indeksi(k) ) aktivna_k = 1;
     else aktivna_k = 0;
vrednost = 0;
for (i = 0; i < m; i++)
 {
     l = 0;
     while (! aktivna_k[l] && l <p_i) l++;
     if(l < p_i ) vrednost = vrednost + g_i [l ];
     else
       {
          vrednost = 0;
          Prekini_Računanje();
       }
 }
for (j = 0; j < n; j++)
     if(y_j == 1) vrednost = vrednost - f_j ;
if(vrednost < 0) vrednost = 0;
```

Slika 7.1 ISP vrednosna funkcija

Teorema 7.1. Složenost vrednosne funkcije GA za rešavanje ISP je $O((m+n)\cdot p)$.

Dokaz: Nalaženje niza y iz genetskog koda zahteva n koraka, pa je vreme izvršavanja $O(n)$. U svakoj konfiguraciji teorijski može biti po najviše n indeksa, iako ih je u praksi uglavnom mnogo manje. Zbog toga je u najnepovoljnijem slučaju potrebno vreme $O(n\cdot p)$ za nalaženje svih aktivnih konfiguracija. Isti slučaj je i pri traženju najpogodnijih konfiguracija za svaki upit, gde je teorijska ocena vremena izvršavanja $O(m\cdot p)$, iako je to u praksi, u najvećem broju slučajeva, mnogo brže. U poslednjem delu se od uštede oduzima vreme potrebno za postavljanje indeksa i tu je potrebno $O(n)$ vreme. Zbog toga je, u najnepovoljnijoj varijanti izračunavanja, potrebno vreme $O((m+n)\cdot p)$ za izvršavanje vrednosne funkcije. ♦

7.3.2 Genetski operatori i ostali aspekti GA

Pri rešavanju ISP su primenjeni isti genetski operatori kao i u slučaju prethodnih problema (selekcija zasnovana na rangu, uniformno ukrštanje i prosta mutacija realizovana korišćenjem normalne raspodele). Takođe je korišćena elitistička strategija zamene generacija i uklanjanje višestruke pojave jedinki u populaciji, što je i ovog puta efikasno spečilo pojavu preuranjene konvergencije. Primenjena je ponovo veličina populacije od 150 jedinki (od čega 100 elitnih), kao i generisanje početne populacije sa istom učestanošću 0 i 1 u genetskom kodu, kao pri rešavanju SPLP.

Jedina razlika u odnosu na prethodne probleme je korišćenje nivoa mutacije koji se menja tokom generacija. Primenjeno je eksponencijalno smanjenje nivoa mutacije od 0.01 u početnoj generaciji i on teži ka vrednosti 0.002 koja se nikada ne dostiže. Pri tome za svakih 300 generacija se nivo mutacije smanji za polovinu opsega.

7.4 Rezultati

7.4.1 Ulazni podaci

Za razliku od prethodno rešavanih problema koji su bili nešto opštijeg tipa i mogu modelirati veći broj praktičnih problema, ovaj problem je specifičan i njegova osnovna namena je smanjivanje vremena izvršavanja sistema za upravljanje baze podataka. Zbog toga vrednosti ulaznih podataka moraju biti posebno izabrane, tako da što vernije preslikavaju osobine realnih baza podataka.

Pošto su u radu [Cap95a] detaljno opisani svi aspekti ovog problema vezani za primenu na konkretnim bazama podataka u praksi, oni su zatim iskorišćeni za generisanje verodostojnih ISP instanci. U radu [Cap95b] je dato detaljno uputstvo za konstruisanje instanci datog problema. U skladu sa tim uputstvom su, na slučajan način, generisane odgovarajuće ISP instance klase A (8 grupa po 5). Odgovarajuće instance klase B nisu korišćene, pošto zbog štamparske greške u radu [Cap95b] nisu mogle biti ispravno rekonstruisane. Parametar *s* predstavlja početnu vrednost za generator slučajnih brojeva, a *t* služi kod generisanja matrice g_{ik}.

Tabela 7.1 Karakteristike ISP instanci

Instance	Broj instanci u grupi	Parametri		Dimenzija			Veličina datoteke
		t	*s*	*m*	*n*	*p*	na disku (KB)
AA	5	200	111 - 555	50	50	500	57.5
AB	5	175	111 - 555	50	50	500	57.5
AC	5	150	111 - 555	50	50	500	57.5
AD	5	125	111 - 555	50	50	500	57.5
AE	5	100	111 - 555	50	50	500	57.5
AF	5	75	111 - 555	50	50	500	57.5
AG	5	50	111 - 555	50	50	500	57.5
AH	5	25	111 - 555	50	50	500	57.5

Detaljne informacije o načinu generisanja ovih ISP instanci i ostalim njihovim karakteristikama se takođe mogu naći u radu ([Cap95b]). Napomenimo da su

instance na kojima je izvršavan GA istih osobina kao originalne iz rada [Cap95b], iako nisu potpuno iste, zbog različito postavljenog generatora slučajnih (pseudoslučajnih) brojeva.

7.4.2 Eksperimentalni rezultati

Izvršavanje GANP implementacije za rešavanje ISP je obavljeno na istom računaru kao i kod prethodnog problema. Zbog problema tehničke prirode performanse su mu bile oko 40 MIPS, 14 Mflops). Rezultati GA su prikazani u tabeli 7.2, a uporedno sa njima je u tabeli 7.3 dat prikaz rezultata metode grananja-i-sečenja (BnC) preuzeti iz rada [Cap95b]. Ovi rezultati su dobijeni testiranjem na računaru vrlo sličnih performansi (radna stanica HP 9000/720, 80 MHz, 16 MB memorije, 59 Specs, 58 MIPS, 18 Mflops). U obe tabele (7.2 i 7.3) su, radi preglednosti, rezultati grupisani i prikazane njihove srednje vrednosti, a detaljni rezultati se mogu naći u Dodatku C ovog rada (GA), odnosno u radu [Cap95b] za (BnC).

Napomenimo da rezultate ovih pristupa ipak nije moguće ravnopravno porediti, jer metoda grananja-i-sečenja može i da dokaže optimalnost dobijenog rešenja, dok GA nema takvu mogućnost.

Tabela 7.2 Rezultati GA

Instance	Broj izvrš.	Sred. br. gener.	Sred. vreme izvršavanja	NDR	Ostala rešenja
AA	5×20	75.6	3.398	100	-
AB	5×20	62.9	2.998	100	-
AC	5×20	93.4	3.872	100	-
AD	5×20	146.9	5.576	100	-
AE	5×20	221.9	7.966	100	-
AF	5×20	325.8	11.18	97	1 (0.22%)
AG	5×20	422.4	14.22	98	2 (0.254%)
AH	5×20	559.0	18.90	89	11 (3.31%)

Tabela 7.3 Rezultati BnC

Instance	Srednji broj iteracija	Vreme izvršavanja (s)	Kvalitet dobijenog rešenja
AA	1	1.08	Opt
AB	1	1.34	Opt
AC	1	2.26	Opt
AD	1	4.66	Opt
AE	2	29.26	Opt
AF	23	502.6	Opt
AG	244	7 136	Opt
AH	914	> 50 000	Subopt

7.4.3 Poređenje rezultata

Kao što možemo videti na osnovu rezultata u tabelama 7.2 i 7.3, GA je približno 2.5 puta sporiji pri rešavanju instanci iz grupa AA. Kasnije ta razlika opada, tako da je vreme izvršavanja instanci AD približno jednako.

Za razliku od GA, čije vreme izvršavanja relativno sporo raste na ostalim grupama instanci, kod BnC suprotno tome, ono raste eksponencijalno. Zbog toga je BnC sporiji od GA oko 5 puta na AE instancama, već oko 50 puta za AF, više od 600 puta za AG i više od 2500 puta izvršavanjem AH instanci.

Iako BnC ima mogućnost verifikacije optimalnog rešenja, što GA nema, rezultati su ipak u korist GA pri izvršavanju instanci AF-AH, jer vreme izvršavanja GA neuporedivo brže. Pri tome je kvalitet rešenja GA sasvim zadovoljavajući, jer je u oko 90% (pa i više) slučajeva dobio isto rešenje (NDR).

7.5 Rezultati paralelnog izvršavanja

Testiranje performansi PGANP za rešavanje ISP je obavljeno na sličan način kao i kod prethodna dva problema. Izvršavanje je obavljeno na 1 instanci srednje težine (AF1), korišćenjem redom 1, 2, 4 i 8 procesora, a rezultati su predstavljeni u tabeli 7.4 .

Tabela 7.4 Rezultati PGA za ISP

Instanca	Broj proc.	Broj izvrš.	Rešenje	Sred. br. gener.	Sred. vreme izvrš. (s)	Faktor ubrzanja
AF1	1	10	NDR	171.7	12.71	
	2	10	NDR	80.8	7.04	1.805
	4	5	NDR	57.4	5.31	2.394
	8	3	NDR	38.33	3.91	3.251

Analizom rezultata paralelnog izvršavanja datih u tabeli 7.4 može se videti da su dobijeni još bolji rezultati, poredeći ih sa rezultatima paralelizacije prethodna dva problema. Izvršavanjem na 8 procesora je dobijeno ubrzanje od 3.25 puta u odnosu na sekvencijalni GA. Ovo je vrlo značajno, imajući u vidu veoma dobre performanse sekvencijalne GA implementacije u poređenju sa BnC metodom.

Dobre performanse pri paralelnom izvršavanju su se mogle predvideti, jer su procesori uspeli da u toku rada GA izvrše skoro isti broj generacija na svojim lokalnim potpopulacijama. Na taj način pošto su svi procesori aktivno učestvovali u radu GA, svi su skoro podjednako uticali na poboljšanje performansi paralelnog GA, pa su i dobijeni još bolji rezultati, u odnosu na prethodne probleme. Takođe možemo očekivati da rezultati budu još bolji pri izvršavanju na paralelnim računarima boljih performansi ISP instanci veće težine.

8. ZAKLJUČAK

U ovoj disertaciji su prikazane sekvencijalna i paralelna GA implementacija sa primenom na rešavanju tri konkretna praktična NP-kompletna problema (prost lokacijski problem, problem dizajniranja mreže neograničenog kapaciteta i problem izbora indeksa). Detaljno su opisani svi aspekti obe implementacije, njihove primene na date probleme, kao i dalje mogućnosti primene na ostale NP-kompletne probleme. Pri tome je za svaki od rešavanih problema dat: kratak opis, primenjena GA metoda, dobijeni rezultati, pregled najvažnijih ostalih metoda i njihova uporedna analiza sa datom GA metodom.

8.1 Pregled primenjenih metoda i dobijenih rezultata

U realizaciji sekvencijalne GANP implementacije su primenjeni neki aspekti genetskih algoritama koji su se već dokazali u praksi pri rešavanju NP-kompletnih problema, kao i neki potpuno novi aspekti. Pri projektovanju date implementacije svi delovi koji su zajednički za primenu GA izdvojeni su u posebnu celinu i klasifikovani, čime je postignuta njihova maksimalna funkcionalnost. Efikasnom realizacijom tih delova obezbeđena je osnova za postizanje visokih performansi same implementacije. Svi ostali aspekti koji zavise od prirode samog problema su odvojeni u posebnu celinu, koja je za svaki konkretan problem realizovana tako da u najvećoj meri prati sve njegove dobre osobine. Pri tome je većina podataka grupisana u jednu od dve globalne strukture:

- GA struktura koja sadrži sve zajedničke informacije;
- Problem struktura sadrži specifične informacije o rešavanom problemu.

Izbor parametara važnih za rad GA vrši se vrlo fleksibilno pomoću konfiguracione datoteke. Pri tome se koriste funkcijski pokazivači, čijom dodelom se realizuju izabrani operatori uz očuvanje efikasnosti cele implementacije. Ovim načinom je moguće rekonfigurisanje sistema bez potrebe za ponovnim prevođenjem izvornog koda, što se pokazalo vrlo pogodnim kod testiranja raznih praktičnih aspekata GA. Takođe je predviđena i mogućnost lake nadogradnje za neke specifičnije primene GA. Programski kôd je napisan korišćenjem ANSI standarda programskog jezika C uvek kada je to bilo moguće, tako da je u velikoj meri prenosiv i razne druge platforme.

Pri paralelizaciji date implementacije su očuvani, ne samo principi osnovne GANP implementacije, već i veliki deo podataka i skoro sve implementirane funkcije. Svi dodatni podaci vezani za paralelno izvršavanje su grupisani u posebnoj globalnoj strukturi. Primenjen je distribuirani model paralelizacije GA,

Ovaj model favorizuje podelu celokupne populacije na delove, od kojih se svaki izvršava na nekom od procesa. Susedni procesi povremeno razmenjuju najbolje jedinke u svojim potpopulacijama, čime se ostvaruju efekti paralelnog izvršavanja.

Data paralelna implementacija (PGANP) je realizovana korišćenjem MPI standarda koji u velikoj meri koristi prednosti paralelizacije na višeprocesorskim računarima sa međuprocesorskom komunikacijom prosleđivanjem poruka. Na taj način je postalo moguće korišćenje paralelnih konstrukcija visokog nivoa, uz istovremeno maksimalno očuvanje efikasnosti cele implementacije. Kao višeprocesorski računar je korišćena mreža PC radnih stanica, koja se pokazala kao dobar izbor za realizaciju, testiranje i izvršavanje paralelnih aplikacija jer se postiže puna funkcionalnost uz nisku nabavnu cenu. Pri tome se programski kod, uz vrlo male izmene, direktno može izvršavati i na nekim mnogo moćnijim paralelnim računarima.

Keširanje GA je u velikoj meri poboljšalo performanse obe implementacije, naročito u primeni na NP-kompletne probleme opisane u ovom radu. To je generalna metoda koja se može primeniti i na ostale NP-kompletne probleme, kao i veliki broj drugih primena GA. Dodatna prednost keširanja je to što samo smanjuje vreme izvršavanja genetskog algoritma, a ne utiče na njegove ostale aspekte. Iako su testirani različiti pristupi, najbolje rezultate u praksi je pokazala LRU strategija implementirana pomoću dvostruke heš-tabele. Primenom heš-tabela je postignuto konstantno O(1) vreme potrebno za operacije pretraživanja, ažuriranja i dodavanja blokova u keš memoriji, što je teorijski optimum. Pri tome je više vremena potrebno za inicijalizaciju keširanja, što doprinosi nešto lošijim rezultatima na instancama manje dimenzije. Međutim, mnogo je značajnije što je vreme izvršavanja na praktičnim instancama veće dimenzije smanjeno za više od 20% kod sva tri rešavana problema, a u nekim slučajevima je ušteda iznosila i do 40%.

Pri rešavanju datih NP-kompletnih problema opisanih u ovom radu, eksperimentisano je sa nekoliko varijanti operatora selekcije i ukrštanja, a najbolje rezultate su pokazali selekcija zasnovana na rangu i uniformno ukrštanje. U najvećem broju slučajeva je mutacija, korišćenjem normalne raspodele, bila efikasnija od ostalih realizacija operatora mutacije. Nivo mutacije je određivan u zavisnosti od dimenzije date instance rešavanog problema. Elitistička strategija se pokazala kao dobar izbor i dobijeni su bolji rezultati u odnosu na ostale politike zamene generacija (generacijski i stacionarni GA). To se može objasniti mogućnošću direktnog prenošenja vrednosti jedinki u narednu generaciju, bez ponovnog poziva vrednosne funkcije. Korišćenjem dinamičke promene nivoa mutacije i ukrštanja su dobijeni nešto bolji rezultati u odnosu na njihove konstantne vrednosti tokom generacija, ali su razlike minimalne i daleko od očekivanih.

Za rešavanje prostog lokacijskog problema se najpogodnijom pokazala binarna reprezentacija niza potencijalnih lokacija snabdevača, koja u velikoj meri doprinosi dobrim performansama GA. Funkcija za računanje vrednosti jedinke je efikasno implementirana u zavisnosti od broja uspostavljenih lokacija za snabdevače. Ovakav pristup daje odlične rezultate na SPLP instancama sa više od 1000 potencijalnih lokacija za snabdevače i korisnika. Ova metoda se izuzetno preporučuje za SPLP instance velike dimenzije koji se ne mogu unapred uprostiti izbacivanjem neperspektivnih lokacija, i faktičkim smanjivanjem dimenzije problema koji se rešava. Na takvim instancama sve

prednosti GA dolaze do punog izražaja, što daje rešenje zadovoljavajućeg kvaliteta kombinovano sa relativno kratkim vremenom izvršavanja (kao što se može videti iz rezultata datih u poglavlju 5). Napomenimo da je pri izvršavanju na svim SPLP instancama dobijeno optimalno rešenje ili najbolje poznato rešenje, u slučajevima kada optimalno rešenje nije unapred bilo poznato. Postojeće rezultate je moguće proširiti i unaprediti u nekoliko pravaca:

- Izvršavanje datog programskog koda na SPLP instancama još veće dimenzije (preko 2000 potencijalnih lokacija snbdevača i korisnika) na računarskim sistemima sa više sistemske memorije;
- Hibridizacija genetskih algoritama i dualnih metoda, pošto su nezavisne i komplementarne, za dalje poboljšavanje performansi;
- Primena metode prikazane u ovom poglavlju za rešavanje nekih drugih lokacijskih problema: nalaženja p-medijane (p-median), lokacije snabdevača neograničenog kapaciteta (capacitated facility location), dinamička lokacija snabdevača (dynamic facility location), i drugih sličnih problema.

8.2 Naučni doprinos ovog rada

Neki od najvažnijih novih rezultata u ovom radu su:

- Programski kod sekvencijalne i paralelne GA implementacije (GANP i PGANP). On je projektovan tako da omogući relativno jednostavnu nadogradnju pri rešavanju novog problema, efikasno izvršavanje i fleksibilno zadavanje parametara.
- Ideja o keširanju GA za poboljšavanje njegovih performansi pri rešavanju raznih problema. Najbolji rezultati su postignuti upravo rešavanjem NP-kompletnih problema.
- Za efikasnu realizaciju LRU strategije osmišljena je heš-red struktura koja obezbeđuje konstantno O(1) vreme izvršavanja za pretragu keš memorije, ažuriranje starih i dodavanje novih jedinki u nju.
- Pogodno izabrani aspekti paralelne implementacije koji su omogućili poboljšanje performansi paralelne implementacije u odnosu na sekvencijalno izvršavanje.
- Neki od standardnih aspekata GA su modifikovani ili potpuno promenjeni u ovoj implementaciji radi dobijanja što boljih rezultata (mutacija implementirana korišćenjem normalne raspodele, korekcija prilagođenosti jedinki kod elitističke strategije, uklanjanje višestruke pojave jedinki u populaciji, itd).
- Vrednosna funkcija je, pri rešavanju SPLP, posebno projektovana i vrlo efikasno realizovana korišćenjem različitih strategija pretrage u slučajevima velikog odnosno malog broja uspostavljenih skladišta. Na taj način su uspešno rešene "složene" instance ovog problema velike dimenzije, u realno dostižnom vremenu izvršavanja.
- Pri rešavanju problema izbora indeksa je efikasno implementirana vrednosna funkcija, imajući u vidu prirodu samog problema. Na taj način su dobijeni rezultati koji su uporedivi sa ostalim metodama, a u nekim slučajevima čak i bolji.
- Pri rešavanju problema dizajniranja mreže neograničenog kapaciteta su primenjena brojna unapređenja osnovnog pristupa. Kao jedno od najvažnijih poboljšanja napomenimo izmenu u standardnom Dijkstra-inom algoritmu za nalaženje najkraćeg puta. Iskorišćena je pogodnost da se taj algoritam

ponavlja više puta u genetskom algoritmu, pa je on nešto brži, u odnosu na osnovnu varijantu.
- Osim praktičnih rezultata izvršavanja, data je i teorijska ocena efikasnosti vrednosne funkcije, jer one u najvećoj meri utiču na performanse cele GANP implementacije.

Pored dobijanja novih rezultata, posebna pažnja je poklonjena i sledećim stvarima:
- Klasifikacija i vrednovanje već postojećih aspekata GA i njihovo međusobno uklapanje u jednu celinu;
- Nalaženje rešenja što boljeg kvaliteta u dostižnom vremenu izvršavanja;
- Maksimalno smanjenje vremena izvršavanja;
- Što lakša implementacija nekog drugog NP-kompletnog problema, korišćenjem postojećeg programskog koda;
- Laka paralelizacija datog koda i implementacija na višeprocesorskom računaru, što utiče na dalje smanjenje njegovog vremena izvršavanja;
- Provera opravdanosti novih programskih rešenja i njihovo vrednovanje;
- Teorijsko opravdanje dobijenih praktičnih rezultata izvršavanja;
- Nalaženje uslova pri kojima se pojavljuju neželjeni efekti (preuranjena konvergencija, gubljenje genetskog materijala, itd), kao i nalaženje efikasne zaštite od njih.

Posle prikaza svih osobina sekvencijalne i paralelne GA implementacije, kao i dobijenih rezultata izvršavanja, može se izvesti zaključak da one predstavljaju moćno sredstvo za rešavanje NP-kompletnih problema. GANP i PGANP predstavljaju veoma važne alatke za rešavanje praktičnih instanci datih problema velike dimenzije, i dobro se dopunjuju sa ostalim metodama za rešavanje datih problema.

Naučno istraživanje obavljeno u ovom radu je određen doprinos oblastima kombinatorne optimizacije, genetskih algoritama i paralelnih algoritama. Deo dobijenih rezultata je već objavljen u inostranim i domaćim časopisima, a ostali delovi su u pripremi za objavljivanje.

Dodatak A Uputstvo za korišćenje programskog paketa

U ovom dodatku je dato kompletno uputstvo za instalaciju i korišćenje programskog paketa za rešavanje NP-kompletnih problema pomoću genetskih algoritama. Pošto paralelna implementacija (PGANP) čuva skoro sve aspekte sekvencijalne implementacije (GANP), ukoliko želimo da koristimo samo sekvencijalnu implementaciju, jednostavno ignorišemo opis paralelnih delova (odeljci A.1.2 i A.3). Opisana je verzija paketa za MS-DOS i WINDOWS operativne sisteme, na kojoj su i testirane performanse datih implementacija i dobijeni rezultati prikazani u ovom radu.

A.1 Instalacija

A.1.1 GANP izvršna verzija

Arhivu sa izvršnom verzijom paketa za rešavanje datog problema raspakujemo u poseban direktorijum. Ona se sastoji od sledećih datoteka:

problem.EXE	Ime ove datoteke zavisi od problema koji rešavamo, i ona predstavlja izvršnu verziju programa.
GENET.CFG	Osnovna konfiguraciona datoteka.
PROBLEM.CFG	Konfiguraciona datoteka za rešavani problem (*opciono*).
CRCTABLE.DAT	Datoteka sa koeficijentima polinoma za generisanje CRC kodova.
INVGAUS.DAT	U ovoj datoteci su smeštene vrednosti inverzne funkcije za normalnu (Gauss-ovu) raspodelu.
RANK.DAT	Ova datoteka sadrži niz rangova, koji se koristi ako je izbrana selekcija zasnovana na rangu. (*opciono*).

Osim navedenih mogu se, eventualno, pojaviti i neke druge datoteke specifične za dati problem, najčešće pri korišćenju heuristika za dobijanje početnog rešenja ili za poboljšavanje jedinki tokom generacija.

Program izvršavamo sledećom komandnom linijom:

problem kod

gde *kod* predstavlja kod instance problema koju rešavamo. Na osnovu tog koda, i datoteke *PROBLEM.CFG*, ako ona postoji, rekonstruišu se putanje i puna imena datoteke sa datom instancom i eventualne datoteke sa njenom optimalnom vrednošću. Na taj način se datoteke sa instancama problema i eventualne datoteke sa njihovim optimalnim vrednostima mogu, u opštem

slučaju, smestiti u proizvoljne nezavisne direktorijume. Takav način zadavanja može biti pogodan, u slučajevima kada je puno ime datoteke dugačko, ili se instance učitavaju sa read-only uređaja (CD ROM) pa nije moguće kasnije zadavanje datoteka sa optimalnim rešenjem na istoj lokaciji. Ukoliko datoteka *PROBLEM.CFG* ne postoji, parametar *kod* predstavlja ime ulazne datoteke, koja se u tom slučaju mora nalaziti u istom direktorijumu.

U toku izvršavanja GA, generišu se i sledeće datoteke sa dodatnim informacijama:

GENETEXE.CFG	Informacija o tome kako su postavljeni parametri osnovne GA strukture.
PROBLEXE.CFG	Postavljene vrednosti parametara problem strukture (*opciono*).
problem.REP	Izveštaj u toku izvršavanja GA, ako je određeno da se on štampa i u datoteku. Data datoteka sadrži samo informacije o poslednjem izvršavanju.
problem.SOL	Datoteka koja sadrži kratke izveštaje na kraju izvršavanja GA, o svakom izvršavanju datog programa. Dati su datum i vreme početka, vreme izvršavanja, dobijeno rešenje i kratak opis parametara GA.
problem.OUT	Detaljan izveštaj o dobijenom rešenju, na kraju rada GA. Pošto se pri novom izvršavanju informacije prepisuju preko starih, mogu se naći samo podaci o poslednjem izvršavanju.

A.1.2 GANP izvorni kod

U ovom odeljku će biti precizno opisan postupak za prevođenje izvornog koda sekvencijalne GA implementacije i dobijanje izvršne verzije programa. Moguće je korišćenje MICROSOFT, BORLAND (INPRISE) ili WATCOM prevodilaca za C programski jezik. Dati izvorni kod se sastoji od dve odvojene celine koje se raspakuju u posebne direktorijume:

OSNOVA.ZIP	Sadrži sve aspekte GA zajedničke za sve primene. Pošto je ovaj deo nezavisan od prirode rešavanog problema, za svaki novi problem se može koristiti isti direktorijum (OSNOVA_DIR).
problem.ZIP	Sadrži delove GA implementacije specifične za problem koji rešavamo, pa se za svaki novi problem mora formirati poseban direktorijum (problem_DIR).

Arhiva *OSNOVA.ZIP* se sastoji od sledećih datoteka opisanih u tabeli A.1. Izvorni kod je podeljen na datoteke po funkcionalnosti, pa je u svakoj od njih realizovan jedan od aspekata sekvencijalnog GA.

Tabela A.1 Osnovni deo GANP implementacije

Zaglavlje	Programski kod	Opis
CACHE.H	CACHE.C	Keširanje GA.
CONST.H	-	Konstante zajedničkog dela.
CROSS.H	CROSS.C	Ukrštanje.
FINISH.H	FINISH.C	Kriterijumi završetka.
FITNESS.H	FITNESS.C	Funkcije prilagođenosti.
GAFUN.H	-	Sadrži sva zaglavlja.
GAPARAMS.H	GAPARAMS.C	Inicijalizacija, učitavanje i konfigurisanje parametara GA.
GASTRUCT.H	-	GA struktura i struktura za opis podataka.
GLOIBALS.H	GLOBALS.C	Globalne definicije nizova funkcijskih pokazivača.
MAIN.H	MAIN.C	Glavna procedura GA.
MUT.H	MUT.C	Mutacija.
NEWGENER.H	NEWGENER.C	Politika zamene generacija.
OUTPUT.H	OUTPUT.C	Izveštaji na kraju rada GA.
RANDOM.H	RANDOM.C	Generisanje slučajnih (pseudoslučajnih) brojeva.
REPORT.H	REPORT.C	Izveštaji u toku rada GA.
SELECT.H	SELECT.C	Selekcija.

Arhiva *problem.ZIP* po pravilu sadrži datoteke koje su prikazane u tabeli A.2, iako su moguća i neka odstupanja.

Tabela A.2 Deo GANP koji zavisi od prirode problema

Zaglavlje	Programski kod	Opis
PRCONST.H	-	Konstante vezane za sam problem.
PROBLEM.H	PROBLEM.C	Problem struktura i ostali podaci vezani za dati problem.
problem.H	problem.C	Funkcije GA koje zavise od prirode problema.
PRPARAMS.H	PRPARAMS.C	Inicijalizacija, učitavanje i konfigurisanje parametara problema.
problemHEUR.H	problemHEUR.C	Heuristike za postavljanje početne populacije ili za poboljšavanje rešenja tokom generacija.

Procedura za prevođenje izvornog koda se u opštem slučaju sastoji od sledećih koraka:

- Formirati projekat i uneti sve .C datoteke iz direktorijuma *OSNOVA_DIR* i *problem_DIR*;
- Direktorijume *OSNOVA_DIR* i *problem_DIR* je potrebno dodati listi direktorijuma sa sistemskim zaglavljima (*INCLUDE_PATH*);
- U okviru projekta postaviti globalnu definiciju za izabrani prevodilac, koja odgovara konstrukciji *#define* (BORLAND, WATCOM ili VISC u zavisnosti od prevodioca);
- Gorepomenutu vrednost *kod* postaviti kao parametar komandne linije za izvršavanje programa;
- Izabrati odgovarajući operativni sistem za koji se generiše dati program (MS DOS ili neki od WINDOWS operativnih sistema), memorijski model, nivo informacija za ispravljanje programa (debug information);

- Pokrenuti komandu za prevođenje/povezivanje programskog koda (Build, Compile, Make ili Link).

A.1.3 PGANP

Zbog određenih zahteva za sigurnost izvršavanja i standardnim protokolima za komunikaciju u lokalnoj mreži, MPI standard nije implementiran za MS DOS operativni sistem, već samo za UNIX i Windows NT. U ovom radu je korišćena verzija WMPI za WINDOWS NT 3.51 i WINDOWS NT 4.0 .

Na početku instaliramo deo paketa WMPI (izvršna verzija i kratko uputstvo se može naći u [WMPI98a]), na svim računarima koji pripadaju datoj mreži. Zatim raspakujemo arhivu sa izvršnom verzijom PGANP paketa za rešavanje datog problema u poseban direktorijum, za svaki računar u mreži. Osim datoteka GANP implementacije, koje su ranije opisane, PGANP sadrži i sledeće dodatne datoteke:

problem.PG	Informacije o računarima (procesorima) uključenim u dato izvršavanje, i broj procesa koji se startuje na svakom od njih.
PARAL.CFG	Konfiguraciona datoteka za paralelni deo PGANP.

Ukoliko posedujemo samo jedan računar, a želimo da koristimo WMPI, tada možemo preskočiti dalja podešavanja, i direktno izvršavati program. Međutim, ukoliko želimo pravi višeprocesorski rad, moramo prethodno podesiti parametre svakog računara u mreži. Svi koraci u datoj proceduri podešavanja će biti detaljno objašnjeni za WINDOWS NT 4.0, a slični su i za ostale 32-bitne WINDOWS operativne sisteme. Konfigurisanje se vrši samo jednom, a po njenom obavljanju, nisu kasnije potrebne nikakve intervencije, već je sistem spreman za izvršavanje nakon podizanja operativnog sistema. Potrebno je:

- Podesiti IP adresu, DNS i WINS u meniju *Control Panel* => *Network* => *Protocols* => *TCP/IP Protocol* => *Properties*. Jedna od sigurnih varijanti su IP adrese 192.168.*.* sa maskom 255.255.255.0. DNS zadaje ime računara na Internetu pa je potrebno izbegavati imena koja već postoje. Kao primarnu WINS adresu možemo uzeti istu IP adresu. Potrebno je još uključiti LMHOSTS (Enable LMHOSTS Lookup).
- Veze između logičkih imena računara u datoj lokalnoj mreži i njihovih brojčanih IP adresa se mogu zadati preko datoteke WINNT_DIR\system32\drivers\etc\HOSTS .
- Na svakom od računara u mreži mora biti aktivan servis zadat programom WMPI_DIR\Servers\Service_p4.exe, koji se najjednostavnije instalira na sledeći način. U tom direktorijumu se startuje program
  createservc.exe imeserv WMPI_DIR\Servers
  gde *imeserv* možemo proizvoljno zadati. Zatim ga podesimo u meniju *Control Panel* => *Services* i startujemo. Ukoliko je izabrana opcija rada za običnog korisnika, potrebno je i odgovarajuće podešavanje privilegija datog korisnika, u suprotnom nije, jer administrator poseduje sve privilegije.

Slično sekvencijalnoj implementaciji program izvršavamo komandnom linijom:

problem kod [-p4dbg *nivo*] [-p4out *datot*] [-p4gm *vel*]

gde *kod* predstavlja kod instance problema koju rešavamo. Na osnovu tog koda se rekonstruišu putanje i puna imena ulazne datoteke i eventualne datoteke sa optimalnom vrednošću. One moraju biti smeštene na osnovnom računaru (osnovni proces - master), i mogu biti učitane samo od njega.

Ostali parametri su opcioni i koriste se u sledećim slučajevima:

- Parametrom -p4dbg *nivo* se može zadati koliko detaljne će biti dodatne informacije pri izvršavanju paralelnog programa. Ove informacije se koriste pri praćenju rada programa i eventualnom ispravljanju grešaka;
- One se mogu zapisati u datoteku *datot*, korišćenjem drugog opcionog parametra;
- Korišćenjem parametra -p4gm *vel* se zadaje količina memorije koja je dostupna svakom MPI procesu. Ova vrednost je po definiciji 1MB, a na ovaj način se može povećati.

Osim ovih postoji još nekoliko manje značajnih opcija koje se mogu koristiti pri izvršavanju programa, a detaljnije informacije o njima se mogu naći u [WMPI98a] ili [But94].

U toku izvršavanja PGA generiše se i datoteka PARALEXE.CFG koja prikazuje na koji su način postavljeni parametri u paralelnoj strukturi.

A.1.4 PGANP izvorni kod

Postupak za prevođenje izvornog koda paralelne GA implementacije je nešto komplikovaniji i moguće je koristiti samo MICROSOFT VISUAL C++ 4.0 prevodilac (ili kasnije verzije), uz pomoć WMPI programskog paketa. Izvorni kod PGANP se sastoji od tri odvojene celine koje se raspakuju u posebne direktorijume:

OSNOVA.ZIP	*Isti kao kod GANP.*
PARAL.ZIP	Sadrži sve aspekte vezane za paralelno izvršavanje GA. I ovaj deo je nezavisan od prirode rešavanog problema, pa se može koristiti isti zajednički direktorijum (PARAL_DIR).
problem.ZIP	*Isti kao kod GANP.*

Arhiva *PARAL.ZIP* se sastoji od sledećih datoteka opisanih u tabeli A.3. Izvorni kod je podeljen na datoteke po funkcionalnosti, pa je u svakoj od njih realizovan jedan od aspekata paralelizacije. Iz arhive *OSNOVA.ZIP* se koriste skoro sve datoteke, osim MAIN.C koja je morala biti promenjena i definisana paralelnom delu.

Tabela A.3 Paralelni aspekti PGANP

Zaglavlje	Programski kod	Opis
ARH.H	ARH.C	Paralelna arhitektura koja se primenjuje.
CHOICE.H	CHOICE.C	Izbor jedinki za slanje ili izbacivanje kod distribuiranog PGA.
CPGA.H	CPGA.C	Funkcije vezane za centralizovani model PGA.
DPGA.H	DPGA.C	Funkcije primenjene u distribuiranom modelu PGA.
PARAL.H	PARAL.C	Paralelna struktura i ostali podaci vezani za paralelizaciju.
PLCONST.H	-	Konstante specifične za PGANP.
PLFINISH.H	PLFINISH.C	Globalni i lokalni kriterijumi završetka PGA.
PLFUN.H	-	Zbirno predstavljena sva zaglavlja.
PLPARAMS.H	PLAPRAMS.C	Inicijalizacija, učitavanje i konfigurisanje paralelnih aspekata GA.
RECITM.H	RECITM.C	Prijem jedinke.
RECSOL.H	RECSOL.C	Prijem vrednosti jedinke (*centralizovani model*) odnosno prijem rešenja od suseda (*distribuirani model*).
SENDITM.H	SENDITM.C	Slanje jedinke.
SENDSOL.H	SENDSOL.C	Slanje vrednosti jedinke (*centralizovani model*) odnosno slanje rešenja susedu (*distribuirani model*).
STRCOMM.H	STRCOMM.C	Slanje i prijem GA i paralelne strukture pri inicijalizaciji.
-	MAIN.C	Glavna procedura (konzolna aplikacija).
-	VMAIN.C	Glavna procedura (grafičko okruženje).

Arhiva *problem.ZIP* u ovom slučaju osim već opisanih datoteka za sekvencijalno izvršavanje, sadrži i novu datoteku PRPARAL.C (zaglavlje joj je PRPARAL.H), koja realizuje paralelne aspekte PGANP zavisne od prirode problema.

Pošto je procedura za prevođenje izvornog koda prilično komplikovanija, biće izložena vrlo detaljno za VISUAL C++ 4.0 prevodilac. Ona se sastoji od više koraka:

- Formirati odgovarajući projekat za verziju konzolne aplikacije ili grafičkog okruženja. To se može učiniti iz menija *File => New => Project Workspace => Application* (odnosno *Console Application*);
- Uneti sve .C datoteke iz direktorijuma *OSNOVA_DIR* i *problem_DIR.* Iz direktorijuma *PARAL_DIR* uneti prvo datoteku *vmain.c* (grafičko okruženje) odnosno *cmain.c* (konzolna aplikacija), a zatim sve ostale datoteke sa izvornim kodom. Odgovarajući meni je *Insert => Files into Project.* Takođe uneti WMPI datoteke *vwmpi.lib*, *vwmpi.rc* i *vshell.lib* (grafičko okruženje) ili *cvwmpi.lib* i *vwmpi.rc* (konzolna aplikacija);
- Direktorijume *WMPI_DIR\include, OSNOVA_DIR, PARAL_DIR* i *problem_DIR* je potrebno dodati listi direktorijuma sa sistemskim zaglavljima (*INCLUDE_PATH*) pomoću *Build => Settings => C/C++ => Preprocessor => Additional incl. dir.*;
- U okviru projekta postaviti globalnu definiciju VISC na sledeći način *Build => Settings => C/C++ => General => Preprocessor definitions*;

- Izabrati biblioteku koja dozvoljava više niti (**multithreaded**) sa *Build => Settings => C/C++ => Code Generation => Use run-time library*;
- Gorepomenutu vrednost *kod* postaviti kao parametar komandne linije za izvršavanje programa pomoću *Build => Settings => General => Program arguments*;
- Pokrenuti komandu *Build => Update All Dependencies*, a zatim prevesti kod sa *Build => Build* (F7).

A.2 Konfigurisanje GA

Datoteka *GENET.CFG* služi za konfigurisanje raznih parametara i izbor odgovarajućih genetskih operatora. Sve funkcije se zadaju preko funkcijskih pokazivača, što predstavlja fleksibilan način njihovog zadavanja uz istovremeno očuvanje efikasnosti implementacije. Informacije u konfiguracionoj datoteci mogu biti zadate kao tekst oblika:

- **[Ime Sekcije]**
- **Param = Vr** - Vrednost **Vr** može biti numeričkog ili tekstualnog tipa;
- - Prazan red koji služi za razdvajanje različitih sekcija.

Svaka od sekcija, redom, opisuje: zajednički deo, funkcije, keširanje, prilagođenost, ukrštanje, mutaciju, selekciju, politiku zamene generacija i kriterijum završetka.

A.2.1 Osnovni deo

[Common]

Označava naziv sekcije koja sadrži osnovne informacije zajedničke za celu implementaciju.

InitialNumberOfItems = nnn

Broj *nnn* predstavlja veličinu populacije, odnosno broj jedinki u populaciji. Moguće su sve vrednosti u opsegu 1 do *MAXNITM*, gde konstanta *MAXNITM* u ovoj implementaciji uzima vrednost 500. Najčešće se ipak uzimaju vrednosti iz opsega 20-150.

OptimizationType = MAX/MIN

Ovaj aspekt opisuje vrstu optimizacije, odnosno da li se traži maksimalna ili minimalna vrednost jedinke.

Randomize = n

Vrednost $n > 0$ označava da je generator slučajnih brojeva na početku postavljen na datu vrednost, a $n = 0$, nasuprot tome, označava da je postavljen na osnovu funkcije za merenje vremena, što omogućuje veći nivo slučajnosti.

Prva opcija omogućava determinističko izvršavanje GA koje je korisno u nekim slučajevima, kao što su: opsežna analiza i praćenje rada GA, efikasnije nalaženje mogućih nedostataka i eventualnih grešaka i testiranje performansi keširanja GA.

Druga opcija se koristi posle dobijanja konačne verzije, pri rešavanju nekog unaped zadatog problema pomoću ove implementacije. Pošto se generator slučajnih brojeva postavlja nedeterministički, obično se izvršavanje ponavlja veći broj puta, a najčešće 10-20, pa se računa srednja vrednost dobijenih rezultata. Na ovaj način se smanjuje uticaj nedeterminizma ukoliko želimo preciznije merenje vremena izvršavanja na nekoj konkretnoj instanci problema.

A.2.2 Zajedničke funkcije

[Functions]

U ovoj sekciji su dati pokazivači na zajedničke funkcije pri izvršavanju GA. Iako je najveći broj ovih funkcija definisan u delu koji zavisi od prirode problema, njihov značaj je opšti, pa funkcijski pokazivač zadat u ovom zajedničkom delu omogućava pozivanje date funkcije globalno na nivou GA, nezavisno od prirode samog problema.

Input = fp

Ovaj red u konfiguracionoj datoteci služi za izbor funkcije za učitavanje podataka, i dodeljuje je odgovarajućem funkcijskom pokazivaču. Pošto učitavanje podataka zavisi od prirode samog problema, spisak odgovarajućih funkcija kandidata se može videti u datoteci *PROBLEM.C* u arhivi koja sadrži programski kod specifičan za dati problem. Pošto je učitavanje podataka obično jednoznačno, u velikom broju slučajeva postoji samo jedna funkcija kandidat. Međutim, sa funkcijskim pokazivačem je ostavljena mogućnost da za neki problem smeštanje podataka ne bude jednoznačno rešeno.

Init = fp

Na ovaj način se bira funkcija za inicijalizaciju aspekata GA koji zavise od prirode samog problema. Jedna od najvažnijih uloga te funkcije je postavljanje početne populacije na slučajan način ili pomoću heuristika.

ReportGA = fp

Izveštaj u toku izvršavanja GA, koji prikazuje karakteristike genetskog algoritma u svakoj generaciji, je takođe značajan za uspešno otkrivanje i otklanjanje anomalija koje se pojavljuju u radu GA. Po obimu detalja u prikazu i načinu prikaza postoji 7 različitih varijanti:

None	GA se izvršava bez ikakvog prikaza u toku izvršavanja.
ShortGAReport	Kratak prikaz na ekranu.
MediumGAReport	Nešto detaljniji prikaz na ekranu.
LongGAReport	Opsežan prikaz sa mnogo više detalja na ekranu
ShortGAReportFile	Kratak prikaz na ekranu i u datoteci.
MediumGAReportFile	Nešto detaljniji prikaz na ekranu i u datoteci.
LongGAReportFile	Opsežan prikaz sa mnogo više detalja na ekranu i u datoteci.

ReportProblem = fp

Elementi izveštaja u toku izvršavanja GA koji zavise od prirode problema se zadaju ovom opcijom. Oni, po pravilu, mogu sadržati detaljniji opis rešenja (najčešće je to najbolja jedinka) u svakoj generaciji, ako je to potrebno.

ArgumentsToGenCode = fp

Konvertuje argumente problema u genetski kod tekuće jedinke. Ovaj funkcijski pokazivač nije potreban u osnovnoj varijanti GA, već samo u nekim specifičnim slučajevima, na primer kada se koriste heuristike za poboljšavanje rešenja a dato poboljšano rešenje se posle toga vraća u populaciju.

GenCodeToArguments = fp
ArgumentsToF = fp

Vrednosna funkcija je jedan od najvažnijih aspekata za uspešnu primenu GA. Radi fleksibilnosti i očuvanja performansi je podeljena na dva dela koji se zadaju ovim redovima konfiguracione datoteke. Prvi deo pretvara genetski kod u argumente problema koji rešavamo, a drugi na osnovu tih argumenata izračunava vrednost date jedinke. Pri rešavanju problema opisanih u ovom radu je vrednosna funkcija bila jedinstveno definisana, a postojale su samo različite realizacije te funkcije u težnji za što boljim performansama. Međutim, ova implementacija ravnopravno dopušta i različite vrste kodiranja i odgovarajuće različite načine definisanja vrednosne funkcije.

OutputGA = fp

Ovom stavkom u konfiguracionoj datoteci se zadaje forma izveštaja na kraju izvršavanja GA, koji prikazuje konačne rezultate genetskog algoritma. Dati izveštaj se štampa na ekranu i u odgovarajućoj datoteci, ukoliko je potrebno da kasnije analiziramo podatke o rezultatima izvršavanja. Pri svakom novom izvršavanju, odgovarajući podaci se dodaju na postojeće, tako da ova datoteka sadrži izveštaje o svim izvršavanjima datog programa.

Po obimu detalja i načinu prikaza postoje 4 različite mogućnosti izveštaja:

None	Ne štampa se nikakav izveštaj.
ShortGAOutput	Kratak prikaz koji sadrži samo najosnovnije informacije.
MediumGAOutput	Nešto detaljniji prikaz.
LongGAOutput	Opsežan prikaz sa mnogo više detalja.

OutputProblem = fp

Prethodnom opcijom se zadaje izveštaj o izvršavanju GA, koji u velikoj meri sadrži opis aspekata koji su primenjeni, a samo osnovne informacije o dobijenom rešenju. Ukoliko korisnik želi detaljnije informacije o rešenju, one direktno zavise od prirode problema koji rešavamo pa se to mora zadati ovom opcijom. Pošto ove informacije opsežno opisuju dobijeno rešenje, one se najčešće ne mogu videti na ekranu, već se samo štampaju u odgovarajućoj datoteci. Zbog velikog obima ovih podataka, u slučaju izvršavanja na

instancama problema veće dimenzije, ova datoteka sadrži samo informacije o rešenju dobijenom u tekućem izvršavanju.

NBit = fp

U ovoj implementaciji je osim pojedinačne primene nekog od operatora ukrštanja dozvoljeno i kombinovanje više različitih operatora ukrštanja, i primena na jedinke u populaciji. Pri tome je svaki od pojedinačnih operatora zadužen za ukrštanje jednog dela jedinke. Granične pozicije bitova svakog dela u okviru jedinke, izračunavaju se pomoću posebne funkcije. Posle toga se svaki pojedinačni operator ukrštanja primenjuje direktno na odgovarajući deo jedinke, umesto na celu jedinku što je bio slučaj pri samostalnoj primeni.

U ovom redu konfiguracione datoteke se vrši izbor odgovarajuće funkcije koja generiše date granične pozicije bitova u slučaju kombinovanog ukrštanja. Ukoliko se primenjuje pojedinačni operator ukrštanja, ova funkcija nema nikakvih efekata na izvršavanje GA.

Napomenimo da kombinovano ukrštanje nije dalo rezultate koji su očekivani, u rešavanju problema opisanih u ovom radu. Međutim ova mogućnost je ostavljena ukoliko se pokaže kao pogodna u nekim budućim primenama.

ImprovingHeuristic = fp

Služi pri zadavanju heuristike za poboljšavanje rešenja (jedinki) u svakoj generaciji, ukoliko se koristi takav pristup.

[CachingGA]

Parametri vezani za keširanje GA se postavljaju u ovoj sekciji. Oni mogu značajno da utiču na uspešnost ove metode (procenat jedinki čija se vrednost preuzima direktno iz keš memorije umesto izračunavanja), a tako i na performanse cele GA implementacije.

Length = n

Kao što je već rečeno za keširanje GA se koristi sistemska memorija računara. Na ovaj način (postavljanjem ovog parametra) se zadaje maksimalna veličina sistemske memorije koja može biti utrošena za keširanje GA. Napomenimo, da time nije fiksno određena veličina keš memorije u KB, već ona zavisi od veličine svake od jedinki. Maksimalno je dopušten broj jedinki određen konstantom *MAXBUF*, koja u ovoj implementaciji iznosi 50 000.

U slučaju linearne strukture podataka (*FirstLinear*) se preporučuje da veličina keš memorije bude u opsegu 20 - 200, a u nekom od pristupa korišćenjem heš-tabela 500 - 50 000. Ovo je posledica činjenice da je pretraživanje heš-tabele u prosečno konstantnom vremenu izvršavanja, pa povećanje veličine keš memorije praktično ne utiče na povećanje vremena pretrage keš memorije. Međutim, pri alokaciji i inicijalizaciji velike keš memorije se ipak troši značajno vremena, iako se te operacije izvršavaju samo jednom (na početku rada GA).

Povećanje veličine keš memorije utiče primetno, ali manje nego što je to očekivano, na povećavanje broja jedinki koje su pronađene u keš memoriji.

Ovaj broj raste otprilike logaritamskom brzinom u zavisnosti od veličine keš memorije. Imajući u vidu gore navedeno ponekada se zadaje i nešto manja veličina keš memorije, od meksimalno dopuštene (50 000).

Function = fp

Datim dodeljivanjem se postavlja funkcija za keširanje, gde se bira između sledećih mogućnosti:

None	Ne vrši se nikakvo keširanje.
FirstLinear	Vrši se keširanje korišćenjem LRU strategije na linearnoj strukturi podataka u obliku niza. Primena date funkcije keširanja je detaljno opisana u radu [Kra99b]
CRCDHash	Pri realizaciji LRU strategije je primenjena dvostruka heš tabela. Pošto je u heš tabeli potrebno prosečno O(1) vreme za pretragu, ubacivanje i brisanje elementa, efikasnost implementacije za keširanje je najveća teorijski moguća.
CRCDHash2	Verzija prethodne strategije, ali je izostavljeno poređenje jedinki ukoliko su CRC kodovi jednaki.
CRCHashQueue	LRU strategija je primenjena uz pomoć kombinovane strukture podataka (heš tabela + red).
CRCHashQueue2	Verzija prethodne metode uz izostavljanje poređenja jedinki u slučaju kada su CRC kodovi jednaki.

Ukoliko želimo potpunu sigurnost pri izvršavanju preporučuje se izbor *CRCHashQueue* (ili *CRCDHash*), a ako je cilj što brže izvršavanje *CRCHashQueue2* (ili *CRCDHash2*). Napomenimo i da je, kod druge varijante, u praksi izmerena učestanost pogrešne procene jedinke znatno manja od 10^{-5} (teorijska procena je oko 10^{-9}), što je u velikoj većini primena praktično zanemarljivo i ne utiče na rad GA.

A.2.4 Prilagođenost

[Fitness]

Ova sekcija sadrži sledeće parametre:

Coeficient_a = a
Coeficient_b = b
Coeficient_c = c

Koeficijenti *a*, *b* i *c* se koriste kod linearnog skaliranja i sigma odsecanja. Ne mogu se dati apriorna uputstva za izbor ovih koeficijenata, već to zavisi od vrste problema koji rešavamo.

Power = x

Ukoliko vršimo stepenovanje prilagođenosti jedinke, kao osnova se koristi prethodno izračunata prilagođenost jedinke, pa se zatim vrši njeno stepenovanje sa zadatom vrednošću *x*.

Type = fp

Funkcija prilagođenosti može biti jedna od sledećih nekoliko mogućnosti:

None	Vrednost jedinke direktno postaje prilagođenost.
PureLinear	Linearno skaliranje f(x) = *aaa* * x + *bbb.*
DirectTo01	Direktno skaliranje u jedinični interval po formuli (2.6).
InverseTo01	Inverzno skaliranje u interval [0,1] po formuli (2.7) .
DirectToStretch	Linearno skaliranje, ali koeficijenti nisu konstantni već se određuju prema formulama (2.4) i (2.5).
DirectSigmaTruncation	Sigma odsecanje.

Preporučuje se neka od poslednje 4 funkcije prilagođenosti, ukoliko nije primenjena selekcija zasnovana na rangu. U suprotnom, sam operator selekcije vrši dodeljivanje prilagođenosti jedinkama na osnovu njihovih rangova u populaciji, pa treba izabrati što prostiju funkciju (neka od prve 2).

A.2.5 Ukrštanje

[Crossover]

U ovoj sekciji se zadaju parametri vezani za operator ukrštanja.

Probability = x

Nivo ukrštanja ($0 < x \leq 1$) određuje učestanost parova jedinki na kojima se vrši ukrštanje. Obično se preporučuje neki broj u opsegu 0.65 - 1, jer se to pokazalo uspešnim za većinu primena. Data preporuka nije stroga, pa se može zadati proizvoljna vrednost iz jediničnog intervala.

ProbabilityOfUniform = x

Ova vrednost određuje učestanost razmene bitova kod uniformnog ukrštanja. Za svaki par bitova u genetskom kodu obe jedinke se sa verovatnoćom *x* vrši razmena, a sa verovatnoćom 1-*x* oni ostaju na svojim starim mestima.

ChangingOfParameters = x

Ovaj parametar služi samo pri statičkoj ili dinamičkoj promeni nivoa ukrštanja tokom generacija. U suprotnom, ukoliko je nivo ukrštanja konstantan tokom generacija, ova vrednost nema nikakav uticaj na izvršavanje GA.

NumberOfIntervals = n

U slučaju višepozicionog ukrštanja, na ovom mestu se zadaje broj pozicija za ukrštanje.

Type = fp

Realizovano je nekoliko standardnih operatora ukrštanja, pa *fp* može imati jednu od vrednosti:

OnePoint	Najjednostavnije jednopoziciono ukrštanje.
TwoPoint	Dvopoziciono ukrštanje.
MultiPoint	Ukrštanje na više pozicija.
Uniform	Uniformno ukrštanje.

Osim datih, realizovano je još nekoliko operatora ukrštanja, kao i mogućnost kombinovanja više operatora istovremeno. Međutim, takvi pristupi primenjeni na NP-kompletne probleme rešavane u ovom radu nisu dali očekivane rezultate, pa nisu uključeni u GANP. Ukoliko neki od tih operatora u budućim primenama bude dokazao svoje kvalitete, njegovo uključivanje u GANP biće vrlo jednostavno.

Opšte uputstvo za to koji od datih operatora izabrati, se može detaljnije videti u odeljku 2.4.2 .

A.2.6 Mutacija

[Mutation]

Svi podaci vezani za operator mutacije se zadaju u ovoj sekciji.

Change = ON/OFF

Da li je nivo mutacije konstantan tokom generacija ili se dinamički menja?

Probability = x

Početni nivo mutacije, može ostati stalan ili se menjati tokom generacija.

ChangingBase = x

Vrednost koja služi kao osnova za izračunavanje nivoa mutacije u svakoj pojedinačnoj generaciji, ako se on menja statički ili dinamički. U slučaju statičke promene (primenjena je eksponencijalna funkcija), nivo mutacije se menja od osnovne vrednosti i teži ka ovoj vrednosti. Ako se nivo mutacije menja dinamički, pri računanju nivoa mutacije u svakoj generaciji, ova vrednost se množi funkcijom koja meri sličnost jedinki u populaciji. Detaljnije o ovome se može videti u odeljku 2.5.2 i dodatku B.

HalfGeneration = n

Ukoliko je zadata vrednost *n* pozitivna, ona predstavlja faktor kojim množimo eksponent, pri određivanju nivoa mutacije u slučaju statičke (eksponencijalne) promene. U praksi on predstavlja broj generacija potreban da se nivo mutacije promeni od početnog nivoa do sredine intervala u kome se mutacija menja.

Ako se primenjuje ovakav način promene nivoa mutacije, uglavnom se može preporučiti da ovaj parametar bude nekoliko puta manji od broja očekivanih

generacija u izvršavanju GA. Međutim, to zavisi i od izbora početne vrednosti nivoa mutacije i osnove za njeno izračunavanje.

Type = fp

U ovoj implementaciji je primenjen samo osnovni model mutacije, ali je on realizovan na različite načine. Svaki od njih ima nekih prednosti i nedostataka, pa se preporučuje u određenim slučajevima. Ukoliko izabrani operator mutacije ima parametre takve da on primetno odstupa od osnovnog modela, odmah se prekida rad i štampa odgovarajući izveštaj. Mogu se koristiti sledeći operatori mutacije:

Simple	Realizacija korišćenjem binomne raspodele.
SimpleSlow	Prosta mutacija implementirana po definiciji (bit po bit).
SimpleNorm	Mutacija realizovana pomoću normale raspodele.
SimpleNormAll	Mutacija zasnovana na normalnoj raspodeli koja se primenjuje direktno na celu populaciju.

Mogu se koristiti i operatori mutacije zavisni od prirode problema, ali se oni realizuju i zadaju u delu specifičnom za dati problem.

U najvećem broju slučajeva se može preporučiti poslednja varijanta (*SimpleNormAll*).

A.2.7 Selekcija

[Selection]

Ova sekcija sadrži sve podatke vezane za operator selekcije i njegovu uspešnu primenu.

NumberOfCompetitors = n

Ovaj podatak predstavlja veličinu turnira ako je primenjena turnirska selekcija. Obično se uzimaju manje vrednosti (2 - 4), jer suviše velika veličina turnira neopravdano favorizuje samo nekoliko najboljih jedinki. Na taj način se ostale jedinke izbacuju iz generacije, i nastupa preuranjena konvergencija.

AverageCompetitors = x

Dati parametar predstavlja srednju veličinu turnira kod fino gradirane turnirske selekcije. Pošto je ova vrednost realan (racionalan) broj, izvršavanje GA se može bolje podesiti nego u slučaju obične turnirske selekcije, gde je veličina turnira konstantna.

NumberOfGrantees = n

Ova vrednost se primenjuje kod zadavanja politike zamene generacija, i predstavlja broj jedinki koje prolaze u narednu generaciju bez primene operatora selekcije.

Type = fp

Primenjeni su sledeći genetski operatori selekcije:

Simple	Prosta rulet selekcija.
RankBasedRouletteByScalingToRandom	Selekcija zasnovana na rangu korišćenjem ruleta.
RankBasedDirect	Selekcija zasnovana na rangu korišćenjem direktnog izbora.
Tournament	Turnirska selekcija.
FineGrainedTournament	Fino gradirana turnirska selekcija.
StohasticReminderSelectByScalingLinear &Random	Selekcija primenom ostataka.

Od ovih načina selekcije u većini slučajeva je dobre rezultate postigla selekcija zasnovana na rangu korišćenjem ruleta, koja je vrlo uspešno doprinela izbegavanju pojave preuranjene konvergencije. Primena proste rulet selekcije i selekcije primenom ostataka je vrlo često davala loše rezultate.

A.2.8 Politika zamene generacija

[GenerationReplacementPolicy]

Politika zamene generacija je značajan aspekt GA, ona može u velikoj meri da utiče na kvalitet dobijenog rešenja i vreme izvršavanja genetskog algoritma. Veći deo podataka koje definišu ovaj aspekt GA je dat u ovom odeljku.

NumberOfElitist = n

Dati parametar služi za primenu elitističke strategije i njime se zadaje broj elitnih jedinki, koje direktno prolaze u narednu generaciju, bez primene genetskih operatora selekcije, ukrštanja i mutacije. Vrednosti elitnih jedinki se izračunavaju samo u početnoj generaciji, a zatim se samo prenose iz svake generacije u narednu, bez ponovnog izračunavanja. Zbog toga elitne jedinke u maloj meri utiču na vreme izvršavanja GA u odnosu na ostale jedinke. Mogućnost neopravdane dominacije elitnih nad ostalim jedinkama u populaciji je efikasno sprečena smanjivanjem njihovih prilagođenosti po formuli (2.9). Ove jedinke donekle smanjuju gradijent pretrage GA, ali se njihovom primenom uspešno čuvaju dobre jedinke i njihovi geni od ”nesretne” slučajne primene nekog od genetskih operatora. Imajući u vidu te njihove osobine, preporučuje se uključivanje elitnih jedinki u populaciju u većem broju, tako da čine i više od 1/2 populacije.

RefreshmentPeriod = n

Ova vrednost se za sada ne koristi, ali je ostavljena da bi se eventualno koristila u proširenju ovog sistema nekim specifičnim potrebama.

Type = fp

Osim vrednosne funkcije, genetskih operatora, funkcije prilagođenosti i izveštaja o toku GA, potrebno je izvršavanje i nekih drugih funkcija u svakoj

generaciji genetskog algoritma. One mogu uključivati: uređivanje (sortiranje) jedinki u populaciji, markiranje višestrkuih pojava neke jedinke za ukljanjanje u narednoj generaciji, izračunavanje sličnosti jedinki u populaciji i drugo.

Datim parametrom se zadaju dve funkcije ove namene, jedna se izvršava pre, a druga posle primene genetskih operatora. Može se zadati jedna od nekoliko varijanti:

None	Nema nikakvih dodatnih radnji.
QSortGA	Uređivanje populacije primenom *quick_sort*() funkcije (.
QSortGARemoveEqual	Uređivanje populacije i markiranje višestruke pojave jedinki.

Kao što je već rečeno, uklanjanje višestruke pojave jedinki u populaciji u velikoj meri poboljšava performanse GA, i efektivno smanjuje mogućnost gubljenja genetskog materijala i preuranjene konvergencije. Zbog toga se, u praksi, preporučuje primena poslednje varijante (QSortGARemoveEqual).

SimilarityFunction = fp

Na ovom mestu se zadaje funkcija za merenje sličnosti jedinki u populaciji. Implementirane su sledeće funkcije:

None	Data funkcija se ne koristi.
Equality	Računanje sličnosti preko broja istih jedinki u populaciji.
Remove Equal	Takođe se nalazi broj istih jedinki u populaciji, samo se svaka njihova višestruka pojava markira za izbacivanje u narednoj generaciji.

A.2.9 Kriterijum završetka

[FinishingCriterium]

Bitan deo za izvršavanje genetskog algoritma je i kriterijum za njegov završetak. Svi parametri bitni za kriterijum završetka su zadati u ovoj sekciji. Od izbora odgovarajuće funkcije zavisi i to da li se i kako primenjuju neki od parametara.

UserQuitPosibility = ON/OFF

Ovaj parametar pokazuje da li je dozvoljeno korisniku da prekine izvršavanje GA uz štampanje izveštaja sa rezultatima genetskog algoritma koji su do tada ostvareni.

MaxNumberOfUnchangedBest = n

Maksimalan dozvoljen broj generacija tokom kojih se nije promenila najbolja jedinka. Preporučuje se zadavanje ovog parametra imajući u vidu da li se dati kriterijum primenjuje samostalno ili u kombinaciji sa drugim kriterijumima, kao i prirodu instance problema koji se rešava. Pri tome nije toliko bitno kolika je dimenzija same instance, već njena "težina" pri izvršavanju pomoću GA. Zbog

toga se primenjuju vrlo različite vrednosti ovog parametra počev od 20 za "lakše" instance u samostalnoj primeni ovog kriterijuma, pa do 10 000 za vrlo "teške" instance ako se primenjuje kombinacija svih kriterijuma završetka GA.

MaxEquality = x

Najveća dopuštena sličnost jedinki u populaciji, a data vrednost je iz opsega [0,1]. Iako se može zadati proizvoljna vrednost iz tog intervala, u praktičnoj primeni se najčešće primenjuju vrednosti u opsegu 0.6 - 0.95 .

MaxNumberOfGenerations = n

Maksimalan broj generacija pri izvršavanju GA. U praksi se ovaj broj kreće u opsegu od 20-100 za instance manje dimenzije i ako se ovaj kriterijum primenjuje samostalno, do 50 000 za instance problema vrlo velike dimenzije i ako se primenjuje kombinacija svih kriterijuma.

StartGenerationToProveOptimal = n

Ukoliko se primenjuje kriterijum verifikacije optimalnog rešenja (ispitivanje njegove optimalnosti), nije svrsishodno primenjivati dati postupak u početnim generacijama GA, jer rešenje sigurno nije optimalno. Tek kasnije, svojim izvršavanjem, genetski algoritam ima šansu da dostigne rešenja boljeg kvaliteta, pa i optimalno. Zbog toga se ovim parametrom zadaje redni broj generacije GA u kojoj počinje dato ispitivanje optimalnosti, ukoliko je ono realizovano.

Type = fp

Zadaje se odgovarajući kriterijum završetka, i on može biti:

NumberOfGeneration	GA se izvršava do određenog broja generacija.
UnchangedBest	Izvršavanje traje sve dok, broj generacija tokom kojih se najbolja jedinka nije promenila, nije dostigao unapred zadatu vrednost.
UnchangedBestOr NumberOfGeneration	Kombinacija prethodna dva kriterijuma.
AllConditions	Kombinacija svih (dole pomenutih) kriterijuma. Po ispunjavanju bar jednog od kriterijuma prekida se izvršavanje GA.

Imajući u vidu rezultate u praksi, u najvećem broju slučajeva se preporučuje primena poslednjeg kriterijuma, koji sadrži kombinaciju svih postojećih kriterijuma. On sadrži sledeće elemente:

- Prekid od strane korisnika;
- Broj generacija je dostigao granicu;
- Najbolja jedinka se nije promenila određen broj generacija;
- Sličnost jedinki u generaciji veća od granične;

- Rešenje je verifikovano (dokazano) kao optimalno. Ovaj kriterijum se koristi samo ako je realizovana odgovarajuća funkcija za verifikaciju optimalnosti rešenja;
- Optimalno rešenje je unapred poznato za datu instancu problema i dostignuto je pri izvršavanju GA.

ProveOptimalSolution = fp

Dati parametar zadaje funkciju za verifikaciju optimalnosti rešenja, ukoliko je ona realizovana. Ova funkcija zavisi od prirode samog problema, pa je njena eventualna realizacija data u odgovarajućem delu programskog koda.

Napomenimo da je ova opcija predviđena u budućoj primeni, jer za date probleme rešavane u ovom radu (SPLP, UNDP i ISP), ovakva funkcija nije još uspešno realizovana.

A.3 Konfigurisanje paralelnih aspekata

Pomoću datoteke PARAL.CFG se postavljaju svi parametri vezani za paralelno izvršavanje genetskog algoritma. Na osnovu toga se može fleksibilno vršiti izbor raznih varijanti pri paralelizaciji: centralizovani ili distribuirani model, razne virtuelne arhitekture, način međuprocesorske komunikacije, i drugo.

Napomenimo da je u ovom trenutku dostupan samo distribuirani model, i da je on korišćen za rešavanje problema opisanih u ovom radu (SPLP, UNDP i ISP), dok je centralizovani model još u fazi testiranja.

A.3.1 Zajednički deo

[Common]

U ovoj sekciji se bira model paralelizacije GA i na osnovu toga se postavljaju odgovarajuće vrednosti parametara bitnih za njegovo izvršavanje.

Implementation = fp

Dati model može biti jedan od:

CentralizedPGA	Centralizovani model, gde osnovni proces izvršava GA, a ostali procesi mu pomažu u računanju vrednosti jedinki koje im on šalje.
DistributedPGA	Distribuirani model, gde svaki proces izvršava GA na svojoj potpopulaciji, uz povremenu razmenu dobrih jedinki između njih.

A.3.2 Paralelna arhitektura

[Arhitecture]

Ova sekcija se koristi za postavljanje virtuelne arhitekture po kojoj komuniciraju pojedinačni procesori. Poželjno je, ali nije neophodno, da data virtuelna arhitektura prati konkretne fizičke veze između datih procesora. Napomenimo da je ona mnogo značajnija u slučaju distribuiranog modela jer se njome zadaje i šema prosleđivanja jedinki između potpopulacija, nego kod

centralizovanog modela, kod koga se ona koristi samo pri prosleđivanju početne konfiguracije svim procesima.

CreatingFunction = fp

Data virtuelna arhitektura može biti:

CPGA	Koristi se kod inicijalizacije u slučaju centralizovanog modela.
LinearArray	Procesori su raspoređeni u jednodimenzionalni niz.
LinearRing	Jednodimenzionalni niz povezan na krajevima.
HyperCube	Definisana je arhitektura hiperkocke.

Sending&ReceivingGAStrFunction = fp

Funkcije koje se ovde zadaju se koriste pri inicijalizaciji genetskog algoritma i služe za slanje odnosno prijem podataka iz osnovne GA strukture. Za sada je implemetirana samo jedna funkcija, koja se koristi kod oba modela paralelizacije (*DPGABase*).

Sending&ReceivingProblemStrFunction = fp

Ovde se zadaju funkcije za prosleđivanje (slanje i prijem) problem strukture, koje zavise od prirode problema koji rešavamo, pa su date u odgovarajućem delu programskog koda.

A.3.3 Kriterijum završetka PGA

[GlobalFinishingCriterium]

Kod sekvencijalnog GA i centralizovanog modela paralelizacije postoji jedinstvena populacija, pa njen lokalni kriterijum završetka označava ujedno i globalni završetak izvršavanja genetskog algoritma. Međutim kod distribuiranog modela nije dovoljan samo lokalni kriterijum završetka GA koji se odnosi na svaku lokalnu potpopulaciju, već je potreban i globalni kriterijum koji važi na celoj populaciji i za sve procese.

SendingFunction = fp
ReceivingFunction = fp

Ove opcije služe za zadavanje načina na koji se generišu i šalju, odnosno primaju, poruke o globalnom završetku GA, u slučaju distribuiranog modela. One takođe, u oba modela, služe i za prosleđivanje poruka o lokalnom završetku GA na nekom procesu ka drugim procesima. Moguće su sledeće varijante:

CPGABase	Osnovni izbor u slučaju centralizovanog modela.
DPGABase	Osnovni izbor u slučaju distribuiranog modela.

A.3.4 Prosleđivanje rešenja

[Solution]

U ovoj sekciji se za oba modela paralelizacije zadaje način izračunavanja vrednosti jedinki u toku izvršavanja GA, a u distribuiranom modelu takođe i način prosleđivanja konačnih rešenja po završetku lokalnih GA.

ComputeValues = fp

Način na koji se izračunavaju vrednosti jedinki u populaciji direktno zavisi od modela paralelizacije, pa su moguće sledeće varijante:

CPGA U slučaju centralizovanog modela osnovni proces raspodeljuje jedinke, i one koje se ne nalaze u keš-memoriji, šalje ostalim procesima na izračunavanje. Pošto ostali procesi izračunaju vrednosti jedinki, prosleđuju ih osnovnom procesu.

DPGA U distribuiranom modelu svaki proces izračunava vrednosti jedinki u svojoj potpopulaciji. Izuzetak su samo jedinke koje su primljene od susednih procesa i one koje su memorisane u keš memoriji.

GASending&ReceivingFunction = fp

Zbog potpuno različitih pristupa pri paralelizaciji pomoću centralizovanog odnosno distribuiranog modela, dati funkcijski pokazivač se primenjuje zavisno od konteksta. Moguće su sledeće varijante:

CPGAComputedValue Služi osnovnom procesu za prijem vrednosti jedinki koje su prethodno poslate na računanje ostalim procesima (centralizovani model).

DPGAOnlyBest Vrši se prosleđivanje konačnih rešenja lokalnih GA, gde se dalje prosleđuje samo najbolje rešenje, koje se na kraju štampa (distribuirani model).

ProblemSending&ReceivingFunction = fp

Ovom opcijom se zadaje način prosleđivanja podataka o konačnom rešenju koje zavise od prirode rešavanog problema.

A.3.5 Razmena jedinki

[ExchangingItems]

Osnovna namena ove sekcije je postavljanje parametara za prosleđivanje jedinki. U slučaju distribuiranog modela vrši se razmena jedinki između potpopulacija, a kod centralizovanog modela osnovni proces šalje jedinke ostalim procesima na izračunavanje vrednosti.

GenerationFrequency = n

Učestanost razmene jedinki između potpopulacija je važan aspekt pri izvršavanju distribuiranog PGA. Ukoliko je to suviše često ili suviše retko gube se prednosti paralelnog izvršavanja, kao što je rečeno u odeljku 3.2.1.2 . Najčešće se preporučuju vrednosti u opsegu 5-30, zavisno od prirode problema koji se rešava.

NumberOfSendTournamentItems = n

Ova opcija se koristi u distribuiranom modelu ako se izbor jedinki za slanje susednim procesima vrši pomoću turnira. Ona u tom slučaju predstavlja veličinu datog turnira, a šalje se najbolja jedinka na turniru.

NumberOfOutTournamentItems = n

Jedinke poslate od susednih procesa u distribuiranom modelu se primaju i smeštaju u potpopulaciju. Na taj način je potrebno izbaciti neke jedinke iz nje. Ukoliko se izbacivanje vrši turnirom, ovaj parametar predstavlja veličinu datog turnira, gde se izbacuje najlošija jedinka.

SendingChoiceFunction = fp

Način izbora jedinke za slanje susednim procesima može biti:

BestInPopulation	Bira se najbolja jedinka u potpopulaciji.
TournamentSelect	Izbor se vrši turnirom, a bira se najbolja jedinka na turniru.

OutChoiceFunction = fp

Način izbora jedinke za izbacivanje iz potpopulacije može biti:

WorstInPopulation	Izbacuje se najlošija jedinka u potpopulaciji.
TournamentSelect	Izbor se vrši turnirom, a izbacuje se najlošija jedinka na turniru.

Sending&ReceivingFunction = fp

Ovde se bira konkretan način prosleđivanja (slanja i prijema) jedinki, koji može biti:

CPGABase	Osnovni način za slanje jedinki na izračunavanje od strane osnovnog procesa, i njihov prijem od strane ostalih procesa (centralizovani model).
DPGAWorkingNeighboors	Realizovane su funkcije za slanje i prijem jedinki između potpopulacija. Svaki proces pri tome šalje jedinke samo susednim procesima gde izvršavanje lokalnog GA još uvek traje. Onim susednim procesima koji su završili izvršavanje lokalnog GA se ne šalju jedinke (distribuirani model).
DPGAAllNeighboors	Slično prethodnom, samo se jedinke šalju svim susednim procesima. Oni susedni procesi koji su završili izvršavanje lokalnog GA te jedinke zatim prosleđuju dalje svojim susedima (distribuirani model).

CPGAFirstNumberOfItems = n

U centralizovanom modelu osnovni proces izvršava GA na svojoj populaciji, pri čemu se jedinke šalju ostalim procesima za izračunavanje njihovih vrednosti. Dati parametar predstavlja broj jedinki koji se u prvoj fazi šalje svakom od ostalih procesa.

CPGALaterNumberOfItems = n

Ukoliko nisu sve jedinke poslate na računanje u prvoj fazi, druga faza nastupa po prijemu prvih izračunatih vrednosti od nekog procesa. U toj fazi se svakom od datih procesa šalje novih *n* jedinki za izračunavanje vrednosti.

A.4 Konfigurisanje konkretnih problema

U ovom odeljku će biti opisano konfigurisanje parametara koji zavise od prirode konkretnih problema (SPLP, UNDP i ISP). Osim detaljnijeg opisa takvih aspekata u konfiguracionim datotekama GENET.CFG i PARAL.CFG, biće dato i kompletno objašenjenje za odgovarajuću datoteku PROBLEM.CFG .

A.4.1 SPLP

U datoteci GENET.CFG se zadaju sledeći parametri:

Functions	Input	UWLInput
	Init	UWLInit
	ReportProblem	None *
		UWLReport
	ArgumentsToGenCode	UWLSimpleCode
	GenCodeToArguments	UWLSimpleCode *
		UWLSimpleCode2
		UWLSimpleCode3
		UWLSimpleCodeFast
		UWLSimpleCodeSlow
		UWLFirst
		UWLSecond
	ArgumentsToF	None
		UWLArgsToF *
	OutputProblem	UWLOutput
	NBit	None
	ImprovingHeuristic	None *
		TwoInterchange
		ChangeOneBit
		TwoInterchange2
		ChangeOneBit2
FinishingCriterium	ProveOptimalSolution	None

Napomena: U slučaju više ponuđenih opcija, one koje su standardne obeležene su zvezdicom.

U datoteci PARAL.CFG se zadaje sledeći parametar:

Arhitecture	Sending&ReceivingProblemStrFunction	UWLP

Dalje će biti data struktura datoteke PROBLEM.CFG, i ona sadrži sledeće elemente u slučaju prostog lokacijskog problema.

[Common]

Zajednička sekcija.

FindMinTreshold = x

Dati parametar služi za nalaženje graničnog broja uspostavljenih skladišta, koji se koristi kod određivanja načina za izračunavanje vrednosne funkcije.

[FileMask]

U ovoj sekciji se zadaju maske na osnovu kojih se formiraju puna imena datoteka.

Input = s

Na osnovu teksta *s* i parametra komandne linije formira se puno ime ulazne datoteke. Ukoliko ovaj parametar nije zadat, kao ime ulazne datoteke uzima se direktno sadržaj komandne linije.

Ako na disku ne postoji ulazna datoteka sa datim imenom, izvršavanje se prekida, i štampa se odgovarajuća poruka.

Optimal = s

Tekst *s* se koristi ukoliko je optimalno rešenje unapred poznato, za formiranje punog imena datoteke koja ga sadrži, da bi ono moglo da se učita.

Ako na disku ne postoji datoteka sa datim imenom, izvršavanje se nastavlja normalno, uz odgovarajuću poruku.

[Heuristic]

Ova sekcija služi za zadavanje heuristike koja generiše početnu populaciju. Za razliku od ostalih sekcija, koje se mogu pojaviti samo jednom, ova sekcija se ponavlja onoliko puta koliko je različitih heuristika definisano.

Function = fp

Dati funkcijski pokazivač definiše datu heuristiku.

NumberOfVariants = n

Broj *n* predstavlja broj različitih varijanti date heuristike. Pri dobijanju početne populacije se generiše taj broj jedinki uz pomoć date heuristike.

A.4.2 UNDP

U datoteci GENET.CFG se zadaju sledeći parametri:

Functions	Input	UNDPInput
	Init	UNDPInit
	ReportProblem	None
		UNDPReport *
	ArgumentsToGenCode	UNDPSimpleCode
		UNDPFastCode *
	GenCodeToArguments	UNDPSimpleCode
		UNDPFastCode *
	ArgumentsToF	None
		UNDPArgsToF *
	OutputProblem	UNDPOutput
	NBit	None
	ImprovingHeuristic	None
FinishingCriterium	ProveOptimalSolution	None

U datoteci PARAL.CFG se zadaje sledeći parametar:

Arhitecture	Sending&ReceivingProblemStrFunction	UNDP

Dalje će biti data struktura datoteke PROBLEM.CFG, i ona sadrži sledeće elemente u slučaju problema dizajniranja mreže neograničenog kapaciteta.

[FileMask]

U ovoj sekciji se kao i kod prethodnog problema (SPLP) zadaju maske na osnovu kojih se formiraju puna imena datoteka.

Input = s
Optimal = s

Na osnovu teksta *s* i parametra komandne linije formira se puno ime ulazne datoteke i datoteke sa eventualnim unapred datim optimalnim rešenjem.

[Heuristic]
Function = fp
NumberOfVariants = n

Ova sekcija služi za zadavanje heuristike koja generiše početnu populaciju.

A.4.3 ISP

U datoteci GENET.CFG se zadaju sledeći parametri:

Functions	Input	ISPInput
	Init	ISPInit
		ISPInit2 *
	ReportProblem	None
		ISPReport
	ArgumentsToGenCode	ISPSimpleCode
	GenCodeToArguments	ISPSimpleCode
		ISPSimpleCode2 *
	ArgumentsToF	None
		ISPArgsToF *
	OutputProblem	ISPOutput
	NBit	None
	ImprovingHeuristic	None
FinishingCriterium	ProveOptimalSolution	None

U datoteci PARAL.CFG je zadat sledeći parametar:

Arhitecture	Sending&ReceivingProblemStrFunction	ISP
Solution	ProblemSending&ReceivingFunction	ISPSolution

Dalje će biti data struktura datoteke PROBLEM.CFG, i ona sadrži sledeće elemente u slučaju problema izbora indeksa.

[FileMask]

U ovoj sekciji se kao i kod prethodna dva problema (SPLP i UNDP) zadaju maske na osnovu kojih se formiraju puna imena datoteka.

Input = s
Optimal = s

Na osnovu teksta *s* i parametra komandne linije formira se puno ime ulazne datoteke i datoteke sa eventualnim unapred datim optimalnim rešenjem.

[Heuristic]
Function = fp
NumberOfVariants = n

Ova sekcija služi za zadavanje heuristike koja generiše početnu populaciju.

Dodatak B. DETALJAN OPIS IMPLEMENTACIJE

Opis detalja svih mogućih aspekata GANP i PGANP implementacija bi bio vrlo opširan, i sadržao bi i neke relativno manje važne delove. S druge strane, neki delovi su već opisani vrlo detaljno (keširanje GA), pa nema potrebe za ponovnim opisivanjem. Zbog toga će u ovom dodatku biti prikazani samo ostali delovi implementacije koji su najvažniji za izvršavanje GA, a koji nisu prethodno detaljno opisani: funkcija prilagođenosti i genetski operatori selekcije, ukrštanja i mutacije.

B.1 Funkcije prilagođenosti

U datoteci *fitness.c* se nalazi veći broj funkcija prilagođenosti, koje izračunavaju prilagođenost jedinki u populaciji na osnovu njihovih vrednosti. Pri učitavanju iz konfiguracione datoteke ("GENET.CFG"), funkcijskom pokazivaču *ga->fitness.f* se dodeljuje pokazivač na izabranu funkciju prilagođenosti. Implementirane su sledeće funkcije, čije se objašnjenje može videti u odeljku 2.3.2.

NoScaling()

Primenom ove funkcije, vrednost svake jedinke postaje njena prilagođenost, što odgovara shemi direktnog preuzimanja.

LinearScaling()

Ova funkcija realizuje linearno skaliranje, gde su koeficijenti *A* i *B* konstantni unapred zadati (učitavaju se iz konfiguracione datoteke). Dati koeficijenti se redom memorišu u promenljivima *ga->fitness.a* i *ga->fitness.b* .

DirectTo01Scaling()

Ova funkcija realizuje skaliraje vrednosti u jedinični interval, u slučaju problema maksimizacije. Funkcija je implementirana u dva ciklusa, gde se u prvom ciklusu računaju minimalna i maksimalna vrednost jedinki u populaciji, a u drugom vrši dodeljivanje prilagođenosti jedinkama po formuli (2.6). Vrednost tekuće jedinke *i* je data sa *ga->pop[i]->funvalue,* a minimalna i maksimalna vrednost se memorišu u lokalnim promenljivima *min* i *max*.

Pre skaliranja se proverava da li postoje bar dve različite jedinke, da bi ono moglo da se izvrši. U suprotnom dalje izvršavanje nema smisla, jer je populacija konvergirala, pa se prekida rad GA, i štampa izveštaj da su sve jedinke iste.

Na sličan način se pre samog skaliranja proverava i korektnost pojedinačnih jedinki u populaciji, i u razmatranje se uzimaju samo korektne jedinke. Ukoliko su sve jedinke nekorektne, dalje izvršavanje nema smisla, pa se prekida izvršavanje GA, i štampa odgovarajući izveštaj.

InverseTo01Scaling()

Formula (2.7) je primenjena kod ove funkcije prilagođenosti, pa se vrednosti inverzno skaliraju u jedinični interval, što se primenjuje u problemima minimizacije. Ako su sve jedinke iste ili sve nekorektne, štampa se izveštaj o tome. Data funkcija je implementirana analogno direktnom skaliranju u jedinični interval.

DirectToStretchScaling()

U ovom slučaju se vrši skaliranje po formuli f(x)=Ax+B, gde *A* i *B* nisu konstantni već se određuju prema formulama (2.4) i (2.5). Parametar *C* u tim formulama se učitava iz konfiguracione datoteke, a memorisan je u promenljivoj *ga->fitness.c,* dok se ostale promenljive memorišu u lokalnim promenljivima. Računanje se vrši u tri faze: nalaženje minimalne i maksimalne vrednosti, nalaženje ostalih parametara i dodeljivanje prilagođenosti jedinkama. I u ovom slučaju se proverava korektnost jedinki i to da li su sve jedinke iste.

DirectSigmaTruncationScaling()

Funkcija prilagođenosti po shemi sigma-odsecanja, koja je data formulom (2.8). Kao i u prethodnim slučajevima proverava se korektnost jedinki i da li su sve jedinke iste. Računanje se, kao i u prethodnom slučaju, vrši u tri faze: nalaženje minimalne i maksimalne vrednosti, nalaženje ostalih parametara i dodeljivanje prilagođenosti jedinkama.

B.2 Selekcija

Više implementiranih varijanti selekcije se nalazi u datoteci *select.c*, gde odgovarajuće funkcije vrše selekciju jedinki za narednu generaciju. Funkcijski pokazivač *ga->select.f* sadrži varijantu selekcije koja se primenjuje u izvršavanju GA, a informacija o tome se učitava iz konfiguracione datoteke.

Ukoliko je jedinka *i* izabrana na mesto *j* u sledećoj generaciji, vrši se kopiranje genetskog koda. U prvoj fazi se kopira dužina genetskog koda, pa se vrši dodeljivanje *ga->pop[i]->tmpcodelen = ga->pop[j]->codelen.* Zatim se kopira ceo genetski kod, odnosno genetski kod jedinke *j* prelazi u privremeni genetski kod jedinke *i*. Time se niz *ga->pop[j]->gencode* kopira na mesto niza *ga->pop[i]->tmpgencode.*

Jedina globalna promenljiva je

FILE **rankfile*;

koja služi u slučaju selekcije zasnovane na rangu kao pokazivač na datoteku gde je smešten niz rangova.

Sledeće varijante selekcije su implementirane:

SimpleSelect()

Proste rulet selekcija, koja je detaljno opisana u odeljku 2.2.2.1, je realizovana simuliranjem bacanje kuglice na ruletu, gde su širine polja na ruletu direktno proporcionalne prilagođenosti jedinke. To se realizuje na sledeći način:

- Za svaku jedinku *i* koju biramo za prolaz u sledeću generaciju, se slučajno bira broj *rndf* iz intervala [0, *sumf*], gde je *sumf* zbir prilagođenosti svih jedinki u populaciji;
- Traži se jedinka *j* čiji opseg prilagođenosti (polje na ruletu) sadrži dati broj *rndf* ;
- Kopira genetski kod jedinke *j* u privremeni genetski kod jedinke *i.*

U narednu generaciju se direktno selektuje najboljih N_{elite} + N_{pass} (*ga->newgener.nelite* + *ga->select.npass*) jedinki. Da date jedinke ne bi bile neopravdano favorizovane u odnosu na ostale jedinke u populaciji, jer je jedna kopija svake od njih direktno prošla proces selekcije, vrši se popravka prilagođenosti po formuli (2.9).

InitRankBasedSelect()

Vrši inicijalizaciju niza rangova kod selekcije zasnovane na rangu, učitavanjem iz datoteke određene pokazivačem *rankfile*, gde je naziv odgovarajuće datoteke na disku "RANK.DAT". Ukoliko u datoteci ima više vrednosti od broja jedinki u populaciji N_{pop} (ga->nitem), učitava se prvih N_{pop} vrednosti. U suprotnom, ako u datoteci ima manje od N_{pop} vrednosti, niz rangova se popunjava sve dok ima vrednosti, a sva nepopunjena mesta se popunjavaju vrednošću poslednjeg učitanog elementa. Niz rangova na koji pokazuje odgovarajući pokazivač (*ga->select.rank*) i koji se alocira dinamički, memoriše učitane vrednosti i kasnije ih prosleđuje u populaciju.

RankBasedDirectSelect()

Primenjena je selekcija zasnovana na rangu, koja je već opisana u odeljku 2.2.2.2, pri čemu se preporučuje da jedinke u populaciji budu uređene u nerastuđem poretku po prilagođenosti. Pri tome se insistira da niz rangova bude uređen u nerastućem poretku. Prilagođenost jedinki se direktno očitava iz niza rangova (**ga->select.rank*). Broj kopija date jedinke u narednoj generaciji se određuje na osnovu prilagođenosti date jedinke. Nedostatak ove metode je potpuno deterministički način izbora jedinki u narednu generaciju, što ne doprinosi u dovoljnoj meri raznovrsnosti genetskog materijala.

RankBasedRouletteSelectR()

Ova funkcija realizuje selekciju zasnovanu na rangu, za izbor jedinki u sledećoj generaciji. Prva faza dodele niza rangova za prilagođenost jedinki je

istovetna kao i u slučaju *RankBasedDirectSelect().* Međutim za razliku od prethodne funkcije za selekciju, sam izbor jedinki za narednu generaciju se vrši na ruletu, isto kao u slučaju proste selekcije. Pri tome se na isti način kao i kod proste rulet selekcije tretira prvih N_{elite} najboljih i sledećih N_{pass} jedinki, koje se direktno biraju, uz istovremenu popravku njihovih prilagođenosti po formuli (2.9).

FinishRankBasedSelect()

Ova funkcija obezbeđuje, na kraju rada GA, oslobađanje memorijskog prostora koji je koristio niz rangova, u slučaju selekcije zasnovane na rangu.

StohasticReminderSelectLR()

Pri ovom načinu selekcije (videti odeljak 2.2.2.4) ceo deo očekivanog broja potomaka neke jedinke direktno se nasleđuje, a razlomljeni delovi učestvuju na ruletu za izbor preostalih, nepopunjenih, elemenata populacije u narednoj generaciji. Pre primene se prilagođenost svih jedinki u populaciji linearno skalira množenjem faktorom N_{pop} / *sumf,* pa se tada direktno dobija očekivani broj potomaka svake jedinke. Pri izboru jedinki na osnovu razlomljenih delova očekivanog broja potomaka, primenjuje se potpuno isti postupak kao i u slučaju proste selekcije.

TournamentSelect()

Turnirska selekcija se, kao što je to već rečeno u odeljku 2.2.2.3, implementira simulacijom N_{pop} - N_{nelite} (*ga->nitem - ga->newgener.nelite*) turnira. Na svakom turniru učestvuje N_{tourn} (*ga->select.ncompet*) jedinki, koje se biraju na slučajan način, a pobednik svakog turnira biva izabran u narednu generaciju. Svaki član koji učestvuje na turniru ima indeks *intItem*, indeks tekuće 32-bitne reči u okviru jedinke je *intLong,* a indeks jedinke-pobednika je *intChamp.* Pobednik turnira je jedinka sa najboljom prilagođenošću, od svih jedinki koje učestvuju na turniru.

FineTournamentSelect()

U nekim slučajevima nije pogodan izbor klasičnog oblika turnirske selekcije, jer su svi turniri iste veličine, gde veličina turnira N_{tourn} (*ga->select.ncompet*) mora biti ceo broj. Na primer u nekim slučajevima veličina turnira N_{tourn} daje suviše sporu konvergenciju, a već za veličinu turnira N_{tourn} + 1 se dobija preuranjena konvergencija. U radu [Fil98] se efektivno uvodi novi oblik turnirske selekcije, nazvan *fino gradirana turnirska selekcija*, gde srednja veličina turnira može biti proizvoljni realan (racionalan) broj *ga->select.fcompet*. Izbor jedinki u narednu generaciju se vrši organizovanjem:

- *intNM* turnira veličine *intTFM* = (*int*) *ga->select.fcompet;*
- *intNP* turnira veličine *intTFP* = *intTFM+1.*

Turniri su ili veličine *intTFM* koji predstavlja prvi ceo broj manji od *ga->select.fcompet*, ili *intTFP* koji je prvi ceo broj veći od njega. Takođe važi jednakost *intNM* + *intNP* = *intN*, odnosno ukupan broj turnira je jednak broju

jedinki koji se biraju za narednu generaciju, pri čemu treba imati u vidu da je *ga->newgener.nelite* jedinki direktno izabrano u narednu generaciju. Brojeve *intNM* i *intNP* nalazimo po formuli (B.1).

```
intN =  ga->nitem - ga->newgener.nelite;
intNM = (int) (intN * (1 + intTFM - ga->select.fcompet));
intNP = intN - intNM;                                          (B.1)
```

Detaljan opis fino gradirane turnirske selekcije, uporedna analiza sa običnom turnirskom selekcijom kao i ostalim načinima selekcije se može videti u [Fil98].

B.3 Ukrštanje

Datoteka *cross.c* sadrži realizacije nekoliko operatora ukrštanja, gde se pokazivač na izabrani operator dodeljuje promenljivoj *ga->cross.f*, na osnovu podataka učitanih iz konfiguracione datoteke "GENET.CFG".

Prilikom primene svakog od operatora ukrštanja, formiraju se parovi jedinki za ukrštanje, kojih ima $(N_{pop} - N_{elite})/2$ (u programu je to (*ga->nitem - ga->newgener.nelite*) / 2). Za svaki par jedinki se sa verovatnoćom p_{cross} (*ga->cross.prob*) vrši ukrštanje njihovih genetskih kodova, a u suprotnom oni ostaju nepromenjeni. Sve implementirane funkcije ukrštanja podrazumevaju da je dužina genetskog koda (*ga->pop[i]->codelen*) ista za sve jedinke u populaciji.

Operator ukrštanja u ovoj implementaciji, privremene genetske kodove jedinki ukršta ili ostavlja iste, što zavisi od nivoa ukrštanja *ga->cross.prob,* i zatim ih smešta kao genetske kodove datih jedinki. Ako se ukrštaju jedinke *i* i *j* tada se postupak može šematski prikazati kao:

```
ga->pop[i]->tmpgencode     ---------->     ga->pop[i]->gencode
ga->pop[j]->tmpgencode      (ukrštanje)    ga->pop[j]->gencode
```

Vrednost lokalne promenljive *nbits* je broj bitova u genetskim kodovima jedinki, a ta vrednost se čuva i globalno u *ga->cross.bits.* Zbog toga se na početku svake funkcije koja realizuje neki operator ukrštanja ispituje da li je globalna promenljiva *ga->cross.bits* postavljena, inače u suprotnom prekida rad GA, i štampa odgovarajući izveštaj.

Implementirani su sledeći operatori ukrštanja:

OnePointCrossover()

Na početku se kopira privremeni genetski kod *ga->pop[i]->tmpgencode* svake jedinke *i* u genetski kod te jedinke *ga->pop[i]->gencode*. Kasnije se eventualno samo razmenjuju delovi genetskog koda između jedinki.

U svakom koraku se na slučajan način iz populacije biraju po dve jedinke za ukrštanje, čiji su indeksi *itm1* i *itm2*, i slučajan broj iz intervala [0, 1]. Ukoliko je dobijeni slučajni broj manji od nivoa mutacije *ga->cross.prob,* vrši se ukrštanje između datih jedinki, a pozicija za ukrštanje *site* se bira kao slučajan ceo broj iz intervala [0, *nbits*]. U suprotnom promenljiva *site* uzima vrednost *nbits*, i nema ukrštanja između datih jedinki. Posle sledećih operacija *l = site/32*; *site = site%32*; promenljiva *l* predstavlja redni broj 32-bitne reči koja sadrži poziciju ukrštanja, a *site* je redni broj bita unutar 32-bitne reči. Sve 32-bitne reči pre

rednog broja *l* ostaju na svojim mestima, sve one koje su posle rednog broja *l* se kompletno razmenjuju, dok se 32-bitna reč na mestu *l* delimično razmenjuje. Pri delimičnoj razmeni, se formira maska, koja sadrži vrednost 0 na mestima bitova koji ostaju nepromenjeni, a sadrži vrednost 1 na mestima bitova koje treba razmeniti. Zbog efikasnosti sve operacije se izvode kao bitovske.

TwoPointCrossover()

Kao i kod prethodnog jednopozicionog ukrštanja privremeni genetski kodovi svih jedinki *ga->pop[i]->tmpgencode* se kopiraju u genetske kodove *ga->pop[i]->gencode*. Takođe se na slučajan način biraju jedinke *itm1* i *itm2* i slučajan broj iz intervala [0, 1]. I u ovom slučaju se vrši poređenje sa nivoom ukrštanja *ga->cross.prob,* što određuje, da li se jedinke ukrštaju, ili ne.

Ako se date jedinke ukrštaju, dve pozicije za ukrštanje *site1* i *site2* se biraju kao slučajani celi brojevi iz intervala [0, *nbits*]. Ako je *site1* > *site2*, oni razmenjuju mesta, pa *site1* dobija vrednost *site2*, i obratno. Na taj način u daljem postupku uvek važi da je prva pozicija za ukrštanje *site1* pre druge pozicije *site2* u genetskom kodu (*site1* ≤ *site2*). Slično kao i u slučaju jednopozicionog ukrštanja vrši se celobrojno deljenje pozicija za ukrštanje *site1* i site2. Količnici *l1* i *l2* predstavljaju redne brojeve 32-bitnih reči koje sadrže pozicije ukrštanja, a ostaci *site1* i *site2* je redne brojeve bitova unutar datih 32-bitnih reči.

Sve 32-bitne reči pre rednog broja *l1* i posle rednog broja *l2* ostaju na svojim mestima, sve one koje su između *l1* i *l2* se kompletno razmenjuju, dok se 32-bitne reči *l1* i *l2* delimično razmenjuju. Pri delimičnoj razmeni na mestu *l1*, se na isti način kao u slučaju jednopozicionog ukrštanja, formira maska, koja sadrži vrednost 0 na mestima bitova koji ostaju nepromenjeni, a sadrži vrednost 1 na mestima bitova koje treba razmeniti. Kao i u prethodnim slučajevima, se zbog efikasnosti sve operacije izvode kao bitovske.

UniformCrossover()

Kao i kod prethodnih operatora ukrštanja privremeni genetski kodovi *ga->pop[i]->tmpgencode* se kopiraju u genetske kodove *ga->pop[i]->gencode*. Takođe se na slučajan način biraju jedinke (*itm1* i *itm2*) i slučajan broj iz intervala [0, 1]. Ukoliko je taj slučajan broj manji od nivoa ukrštanja *ga->cross.prob* jedinke se ukrštaju*,* inače ostaju nepromenjene.

Ako se date jedinke ukrštaju za svaku 32-bitnu reč se generiše odgovarajuća maska ukrštanja. Ona se generiše na slučajan način, tako da je verovatnoća pojavljivanja bita čija je vrednost 1 jednaka p_{unif} (*ga->cross.probunif*), a 0 je $1 - p_{unif}$.

U ovom slučaju se ukrštanje mora efektivno primeniti na svaku 32-bitnu reč, za razliku od prethodnih operatora ukrštanja (jednopozicionog i dvopozicionog), gde se efektivno primenjuje samo na jednu (odnosno dve) 32-bitne reči u genetskom kodu, dok su se ostale direktno kopirale. Zbog toga je operator uniformnog ukrštanja nešto sporiji od prethodnih, ali je to uglavnom zanemarljivo u odnosu na vreme izvršavanja vrednosne funkcije.

B.4 Mutacija

Datoteka *mut.c* sadrži realizacije nekoliko operatora mutacije, gde se pokazivač na izabrani operator dodeljuje promenljivoj *ga->mut.f*, na osnovu podataka učitanih iz odgovarajuće sekcije konfiguracione datoteke "GENET.CFG". Za razliku od prethodnih genetskih operatora (selekcije i ukrštanja) koji su vršili kopiranje genetskog koda jedinke u privremeni genetski kod, ili obratno, operator mutacije se direktno primenjuje na genetski kod jedinke (*ga->pop[i]->gencode*).

Svi operatori ukrštanja primenjuju prostu mutaciju, ali su realizovani različito imajući u vidu efikasno izvršavanje, u slučaju raznih dužina genetskog koda.

Implementirane su sledeće funkcije:

SimpleSlow()

Implementirana je u skladu sa definicijom proste mutacije. Za sve jedinke osim elitnih, kojih ima N_{pop} - N_{elite} (*ga->nitem* - *ga->newgener.nelite*), obrađuje se celokupni genetski kod i za svaku 32-bitnu reč se formira maska mutacije. Verovatnoća pojavljivanja bita sa vrednošću 1 je jednaka nivou mutacije p_{mut} (*ga->mut.prob*). Za svaki bit u masci se primenjuje funkcija *Flip()*, koja određuje da li je vrednost 1 (dati bit se mutira), ili 0 (ostaje nepromenjen).

Pošto se za svaku 32-bitnu reč maska formira bit-po-bit, broj izvršenih operacija je veliki, što donekle usporava GA. Međutim, u velikom broju slučajeva je vreme izvršavanja genetskih operatora višestruko manje u odnosu na vreme izvršavanja vrednosne funkcije. U takvim slučajevima vreme izvršavanja operatora mutacije vrlo malo utiče na ukupno vreme izvršavanja GA, pa se može primeniti i ovaj operator. Prednost ovakvog načina je u tome što je egzaktan, i može se primeniti za proizvoljan broj bitova u genetskom kodu jedinke.

SimpleMutation()

Pošto je nivo mutacije obično mali, najčešće je broj bitova u genetskom kodu jedinke vrlo mali. Ovaj pristup unapred izračunava mogući broj mutiranih bitova i njihove pozicije, a zatim se obrađuju samo takvi segmenti genetskog koda, što bitno ubrzava operator mutacije. Broj mutiranih bitova se određuje korišćenjem binomne raspodele, kao što je dato u (2.11) u odeljku 2.4.3.2 ovog rada.

Dati način je tamo vrlo detaljno opisan, pa samo napomenimo da je ovaj način primenljiv samo za populacije sa genetskim kodom jedinke veličine najviše do 1000 bitova ($N_{bits} \leq 1000$).

SimpleNorm()

Kao i kod prethodne funkcije obrađuju se samo segmenti genetskog koda jedinke za koje su prethodno određene pozicije bitova (gena) koji treba da budu mutirani. Pri određivanju njihovog broja se koristi normalna raspodela kao aproksimacija binomne raspodele.

Pri inicijalizaciji GANP se iz odgovarajuće datoteke ("NORM.DAT") učitava niz koeficijenata inverznih vrednosti normalne raspodele $\Phi^{-1}(x)$, gde funkcija $\Phi(x)$ definiše normalnu raspodelu $N(0, 1)$. Ove vrednosti se smeštaju u

memoriju na lokaciji *ga->mut.norm*, a eksperimentalno je utvrđeno da je 100 memorisanih vrednosti garantuje dovoljnu tačnost pri izračunavanju.

Na osnovu vrednosti N_{bits} (*CODEBITS* = *ga->pop[...]->codelen * 32*) i p_{mut} (*ga->mut.prob*) određuju se očekivanje i devijacija date raspodele ($ex = N_{bits} * p_{mut}$; $sigmax = N_{bits} * p_{mut} * (1-p_{mut})$). Zatim se na slučajan način bira broj ceo broj *num* iz intervala [0, 99] i znak $sign \in \{-1,1\}$. Očitavanjem $s = \Phi^{-1}(num)$ iz tabele normalne raspodele dobija se broj bita (gena) *m* koje treba mutirati na sledeći način:

$$m = \mathbf{int}(ex + sign * s * sigmax + 0.3) \qquad \text{(B.2)}$$

Funkcija int() označava funkciju za računanje celobrojne vrednosti izraza, a vrednost 0.3 se dodaje zbog odsecanja razlomljenog dela izraza. Kao što se može videti ovaj korak se izvršava u konstantnom vremenu izvršavanja.

Posle ovog koraka, vrši se *m* mutacija sa slučajnim izborom mutiranih bitova (gena). Pošto se data aproksimacija može primeniti u praksi ako je ispunjen uslov $N_{bits} * p_{mut} \geq 5$, u takvim slučajevima je pogodno primeniti i ovaj operator.

SimpleNormAll()

I kod ove funkcije se koristi normalna raspodela za izračunavanje broja bitova (gena) koje treba mutirati. Pošto gorepomenuta aproksimacija zahteva da broj bitova bude što veći, u ovoj funkciji se računa broj bitova koje treba mutirati u celoj populaciji (a ne u pojedinačnoj jedinki). Zbog toga je prethodni uslov sada promenjen ($N_{pop} * N_{bits} * p_{mut} \geq 5$) i skoro uvek ispunjen, osim u eventualno nekim ekstremnim slučajevima vrlo male populacije, sa vrlo malim brojem bitova i malim nivoom mutacije.

Dodatak C. POJEDINAČNI REZULTATI IZVRŠAVANJA

U osnovnom delu ovog rada (poglavljima 4. - 7.) su radi preglednosti rezultati izvršavanja grupisani i prikazani zbirno za svaku grupu instanci. Na taj način se vrlo lako mogu uočiti globalne karakteristike koje se javljaju pri izvršavanju GA za određeni problem ili pri testiranju performansi keširanja GA. Međutim i rezultati izvršavanja na instancama iz iste grupe se razlikuju, ponekad čak i jedan red veličine. Da bi se imao uvid i u takve pojedinačne karakteristike, u ovom dodatku su dati i svi pojedinačni rezultati izvršavanja.

C.1 SPLP: Eksperimentalni rezultati

U ovom odeljku su dati rezultati izvršavanja GA, obe verzije DUALOC algoritma i optimalne vrednosti (ili NDR ako optimalna vrednost nije verifikovana) za svaku od instanci ovog problema (41-134, A-C, MO1-MT5).

C.1.1 GA

Tabela C.1 Instance 41-74

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
41	20	16.4	0.23	20	-
42	20	19.6	0.27	20	-
43	20	20.3	0.27	20	-
44	20	15.0	0.21	20	-
51	20	18.8	0.27	20	-
61	20	19.8	0.28	20	-
62	20	20.0	0.27	20	-
63	20	17.8	0.24	20	-
64	20	15.8	0.21	20	-
71	20	16.4	0.21	20	-
72	20	17.0	0.23	20	-
73	20	16.4	0.23	20	-
74	20	15.8	0.22	20	-

Tabela C.2 Instance 81-104

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
81	20	33.8	0.42	20	-
82	20	39.3	0.48	20	-
83	20	42.2	0.51	20	-
84	20	25.4	0.32	20	-
91	20	31.2	0.39	20	-
92	20	39.8	0.48	20	-
93	20	35.4	0.43	20	-
94	20	24.4	0.30	20	-
101	20	32.2	0.40	20	-
102	20	39.2	0.48	20	-
103	20	40.6	0.49	20	-
104	20	27.6	0.34	20	-

Tabela C.3 Instance 111-134 i A-C

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
111	20	106.4	1.30	20	-
112	20	128.6	1.56	20	-
113	20	102.6	1.27	20	-
114	20	73.6	0.92	20	-
121	20	90.2	1.11	20	-
122	20	125.2	1.53	20	-
123	20	113.3	1.40	20	-
124	20	68.6	0.86	20	-
131	20	77.5	0.97	20	-
132	20	152.4	1.84	20	-
133	20	166.1	2.00	20	-
134	20	70.6	0.89	20	-
A	20	229.0	18.80	20	-
B	20	733.8	48.86	20	-
C	20	2 698	154.59	20	-

Tabela C.4 Instance MO, MP i MQ

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
MO1	20	194.7	4.44	20	-
MO2	20	101.4	2.35	20	-
MO3	20	128.9	2.92	20	-
MO4	20	100.4	2.32	20	-
MO5	20	93.3	2.16	20	-
MP1	20	182.1	7.22	20	-
MP2	20	163.7	6.47	20	-
MP3	20	195.7	7.41	20	-
MP4	20	205.4	7.74	20	-
MP5	20	194.3	7.60	20	-
MQ1	20	259.5	14.84	20	-
MQ2	20	253.6	14.55	20	-
MQ3	20	291.6	16.63	20	-
MQ4	20	272.7	15.63	20	-
MQ5	20	245.2	14.23	20	-

Tabela C.5 Instance MR, MS i MT

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
MR1	20	417.6	41.38	20	-
MR2	20	440.3	43.36	20	-
MR3	20	454.0	45.50	20	-
MR4	20	407.4	39.86	20	-
MR5	20	398.8	39.66	20	-
MS1	20	842.5	187.47	20	-
MS2	20	912.2	197.75	20	-
MS3	20	856.8	187.73	20	-
MS4	20	945.4	205.23	20	-
MS5	20	1 069	217.51	20	-
MT1	20	1 864	936.50	20	-
MT2	20	1 904	943.62	20	-
MT3	20	1 750	911.30	20	-
MT4	20	1 793	914.84	20	-
MT5	20	1 719	892.64	20	-

C.1.2 DUALOC

Tabela C.6 Instance 41-74

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
41	1	<0.01	Opt
42	1	<0.01	Opt
43	1	0.05	Opt
44	1	<0.01	Opt
51	1	<0.01	Opt
61	1	<0.01	Opt
62	1	<0.01	Opt
63	1	<0.01	Opt
64	1	<0.01	Opt
71	1	<0.01	Opt
72	1	<0.01	Opt
73	1	<0.01	Opt
74	1	<0.01	Opt

Tabela C.7 Instance 81-104

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
81	5	<0.01	Opt
82	7	<0.01	Opt
83	3	<0.01	Opt
84	1	<0.01	Opt
91	5	<0.01	Opt
92	7	<0.01	Opt
93	3	<0.01	Opt
94	1	<0.01	Opt
101	5	<0.01	Opt
102	7	<0.01	Opt
103	3	<0.01	Opt
104	1	<0.01	Opt

Tabela C.8 Instance 111-134 i A-C

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
111	2	<0.01	Opt
112	8	<0.01	Opt
113	1	<0.01	Opt
114	1	<0.01	Opt
121	2	<0.01	Opt
122	6	<0.01	Opt
123	1	<0.01	Opt
124	1	<0.01	Opt
131	2	<0.01	Opt
132	8	<0.01	Opt
133	1	<0.01	Opt
134	1	<0.01	Opt
A	166	2.30	Opt
B	516	10.19	Opt
C	2 688	63.03	Opt

Tabela C.9 Instance MO, MP i MQ

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
MO1	66 772	65.86	Opt
MO2	8 303	8.57	Opt
MO3	23 125	22.85	Opt
MO4	29 715	28.67	Opt
MO5	6 003	5.76	Opt
MP1	173 051	401.01	Opt
MP2	53 503	117.32	Opt
MP3	193 096	426.33	Opt
MP4	117 198	265.24	Opt
MP5	97 825	219.70	Opt
MQ1	665 310	2814.0	Opt
MQ2	404 701	3543.7	Opt
MQ3	189 110	3476.8	Opt
MQ4	560 003	3472.5	Opt
MQ5	728 703	2977.3	Opt

Tabela C.10 MR instance

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
MR1	10 307 844	99 424	9.4% od NDR
MR2	7 050 800	68 008	6.7% od NDR
MR3	5 522 898	54 167	Opt
MR4	8 121 349	79 779	Opt
MR5	8 375 395	78 444	Opt

C.1.3 DUALOC bez grananja

Tabela C.11 Instance 41-74

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
41	1	<0.01	Opt
42	1	<0.01	Opt
43	1	0.05	Opt
44	1	<0.01	Opt
51	1	<0.01	Opt
61	1	<0.01	Opt
62	1	<0.01	Opt
63	1	<0.01	Opt
64	1	<0.01	Opt
71	1	<0.01	Opt
72	1	<0.01	Opt
73	1	<0.01	Opt
74	1	<0.01	Opt

Tabela C.12 Instance 81-104

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
81	5	<0.01	Opt
82	7	<0.01	Opt
83	3	<0.01	Opt
84	1	<0.01	Opt
91	5	<0.01	Opt
92	7	<0.01	Opt
93	3	<0.01	Opt
94	1	<0.01	Opt
101	5	<0.01	Opt
102	7	<0.01	Opt
103	3	<0.01	Opt
104	1	<0.01	Opt

Tabela C.13 Instance 111-134 i A-C

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
111	2	<0.01	Opt
112	8	<0.01	Opt
113	1	<0.01	Opt
114	1	<0.01	Opt
121	2	<0.01	Opt
122	6	<0.01	Opt
123	1	<0.01	Opt
124	1	<0.01	Opt
131	2	<0.01	Opt
132	8	<0.01	Opt
133	1	<0.01	Opt
134	1	<0.01	Opt
A	166	2.09	9.10% od Opt
B	382	6.97	6.86% od Opt
C	459	10.44	1.25% od Opt

Tabela C.14 Instance MO, MP i MQ

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
MO1	230	0.38	11.02% od Opt
MO2	164	0.28	6.00% od Opt
MO3	243	0.38	17.04% od Opt
MO4	236	0.39	10.83% od Opt
MO5	199	0.33	3.79% od Opt
MP1	509	2.14	6.25% od Opt
MP2	797	3.08	8.96% od Opt
MP3	817	3.18	13.13% od Opt
MP4	619	2.52	13.47% od Opt
MP5	735	2.80	6.34% od Opt
MQ1	1 682	10.76	9.09% od Opt
MQ2	1 249	8.73	11.18% od Opt
MQ3	1 555	10.82	14.51% od Opt
MQ4	1 792	11.59	8.29% od Opt
MQ5	973	7.08	7.47% od Opt

Tabela C.15 Instance MR, MS i MT

Instanca	Sred. broj iteracija	Sred. vreme izvrš. (s)	Rešenje
MR1	2 422	37.57	13.75% od NDR
MR2	3 373	48.01	13.36% od NDR
MR3	2 790	41.19	12.90% od Opt
MR4	2 676	40.21	17.09% od Opt
MR5	3 170	46.08	9.64% od Opt
MS1	10 718	377.62	24.63% od NDR
MS2	10 495	375.47	10.42% od NDR
MS3	9 722	346.09	10.20% od NDR
MS4	8 906	351.85	13.22% od NDR
MS5	8 286	293.03	17.99% od NDR

C.1.4 Optimalna i najbolja dobijena rešenja

Tabela C.16 Instance 41-134 i A-C

Instanca	Rešenje	Instanca	Rešenje	Instanca	Rešenje
41	932615.75	82	854704.2	114	928941.75
42	977799.4	83	893782.11	121	793439.56
43	1010641.45	84	928941.75	122	851495.32
44	1034976.97	91	796648.44	123	893076.71
51	1010641.45	92	854704.2	124	928941.75
61	932615.75	93	893782.11	131	793439.562
62	977799.4	94	928941.75	132	851495.325
63	1010641.45	101	796648.437	133	893076.712
64	1034976.97	102	854704.2	134	928941.75
71	932615.75	103	893782.112	A	17156454.478
72	977799.4	104	928941.75	B	12979071.582
73	1010641.45	111	793439.56	C	11505594.329
74	1034976.975	112	851495.32		
81	796648.44	113	893076.71		

Tabela C.17 Instance MO-MT

Instanca	Rešenje	Instanca	Rešenje	Instanca	Rešenje
MO1	1156.909	MQ1	3591.271	MS1	4378.632
MO2	1227.666	MQ2	3543.662	MS2	4658.349
MO3	1286.369	MQ3	3476.806	MS3	4659.156
MO4	1177.880	MQ4	3742.474	MS4	4536.002
MO5	1147.595	MQ5	3751.326	MS5	4888.911
MP1	2460.101	MR1	2349.856	MT1	9176.509
MP2	2419.324	MR2	2344.757	MT2	9618.855
MP3	2498.151	MR3	2183.237	MT3	8781.115
MP4	2633.561	MR4	2433.110	MT4	9225.497
MP5	2290.164	MR5	2344.353	MT5	9540.667

C.2 UNDP: Eksperimentalni rezultati

Kao i u prethodnom odeljku i ovde su prikazani pojedinačni rezultati izvršavanja GA i odgovarajuće NDR vrednosti za svaku od UNDP instanci (MA1-ME5).

C.2.1 GA

Tabela C.18 Instance MA, MB i MC

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
MA1	20	27.0	0.48	20	-
MA2	20	19.8	0.40	20	-
MA3	20	32.0	0.58	20	-
MA4	20	28.8	0.52	20	-
MA5	20	21.2	0.39	20	-
MB1	20	136.2	3.41	20	-
MB2	20	84.6	2.25	20	-
MB3	20	99.8	2.45	20	-
MB4	20	104.2	2.76	20	-
MB5	20	73.4	2.03	20	-
MC1	20	121.4	4.41	20	-
MC2	20	132.4	4.90	20	-
MC3	20	116.4	3.84	20	-
MC4	20	139.4	4.59	20	-
MC5	20	196.7	6.48	20	-

Tabela C.19 Instance MD i ME

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
MD1	20	387.6	14.20	20	-
MD2	20	170.1	7.81	20	-
MD3	20	582.9	19.10	20	-
MD4	20	375.8	12.93	20	-
MD5	20	109.6	5.87	20	-
ME1	20	1 100	53.14	15	5 (0.22%)
ME2	20	2 556	107.74	5	15 (2.45%)
ME3	20	990.2	45.16	20	-
ME4	20	3 151	116.69	4	16 (1.73%)
ME5	20	3 539	134.63	5	15 (0.83%)

C.2.2 Najbolja dobijena rešenja

Tabela C.20 Sve instance

Instanca	Rešenje	Instanca	Rešenje	Instanca	Rešenje
MA1	55.193	MC1	87.974	ME1	137.823
MA2	53.221	MC2	107.735	ME2	129.454
MA3	57.684	MC3	85.750	ME3	127.715
MA4	56.047	MC4	87.560	ME4	136.312
MA5	55.215	MC5	110.498	ME5	140.869
MB1	81.755	MD1	105.877		
MB2	85.945	MD2	133.368		
MB3	81.445	MD3	114.454		
MB4	94.471	MD4	105.776		
MB5	90.216	MD5	116.365		

C.3 ISP: Eksperimentalni rezultati

Kao i u prethodnom odeljku i ovde su dati pojedinačni rezultati izvršavanja GA i NDR vrednosti svih ISP instanci (AA1-AH5). Pojedinačni rezultati izvršavanja BnC se mogu naći u radu [Cap95b].

C.3.1 GA

Tabela C.21 Instance AA, AB i AC

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
AA1	20	98.0	4.01	20	-
AA2	20	75.5	3.52	20	-
AA3	20	67.2	3.16	20	-
AA4	20	59.2	2.92	20	-
AA5	20	78.4	3.38	20	-
AB1	20	63.1	3.02	20	-
AB2	20	70.0	3.19	20	-
AB3	20	59.3	2.89	20	-
AB4	20	61.0	2.96	20	-
AB5	20	61.4	2.93	20	-
AC1	20	69.4	3.07	20	-
AC2	20	78.4	3.57	20	-
AC3	20	58.2	2.84	20	-
AC4	20	103.3	4.17	20	-
AC5	20	157.7	5.71	20	-

Tabela C.22 Instance AD, AE i AF

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
AD1	20	82.6	3.72	20	-
AD2	20	105.4	4.41	20	-
AD3	20	108.8	4.37	20	-
AD4	20	249.6	8.80	20	-
AD5	20	188.2	6.58	20	-
AE1	20	124.1	4.99	20	-
AE2	20	145.4	5.72	20	-
AE3	20	97.8	4.19	20	-
AE4	20	271.0	8.89	20	-
AE5	20	471.2	16.04	20	-
AF1	20	129.4	5.17	20	-
AF2	20	718.4	23.30	20	-
AF3	20	354.4	11.85	20	-
AF4	20	310.0	10.88	19	1 (0.22%)
AF5	20	116.8	4.71	20	-

Tabela C.23 Instance AG i AH

Instanca	Broj izvrš.	Sred. br. gener.	Sred. vreme izvrš. (s)	Opt. (NDR)	Ostala rešenja
AG1	20	429.6	14.2	20	-
AG2	20	221.0	7.85	20	-
AG3	20	666.2	22.88	18	2 (0.25%)
AG4	20	111.6	4.92	20	-
AG5	20	683.8	21.27	20	-
AH1	20	463.2	16.00	20	-
AH2	20	217.1	8.13	20	-
AH3	20	195.9	7.62	20	-
AH4	20	1 118	37.97	9	11 (3.31%)
AH5	20	800.6	24.77	20	-

C.3.2 Najbolja dobijena rešenja

Tabela C.24 Sve instance

Instanca	Rešenje	Instanca	Rešenje	Instanca	Rešenje
AA1	40 660	AC5	29 721	AF4	13 204
AA2	41 068	AD1	24 359	AF5	13 410
AA3	40 968	AD2	24 766	AG1	7 709
AA4	41 903	AD3	24 503	AG2	7 479
AA5	41 273	AD4	24 211	AG3	7 497
AB1	36 176	AD5	24 634	AG4	7 733
AB2	35 531	AE1	18 310	AG5	7 275
AB3	35 535	AE2	18 179	AH1	2 496
AB4	36 317	AE3	18 431	AH2	2 273
AB5	35 408	AE4	18 944	AH3	2 440
AC1	30 095	AE5	19 004	AH4	2 359
AC2	30 211	AF1	12 586	AH5	2 366
AC3	30 041	AF2	13 161		
AC4	30 086	AF3	12 755		

C.4 Performanse keširanja GA

I pri testiranju performansi keširanja dobijeni rezultati za neku instancu mogu se prilično razlikovati od prosečnih rezultata za tu grupu. Na primer ušteda vremena korišćenjem keširanja GA pri izvršavanju instance MC5 daleko prevazilazi uštedu u slučaju ostalih MC instanci.

C.4.1 Rezultati na SPLP

Tabela C.25 Instance 41-74

Instanca	GA bez keširanja				CRCHashQueue		CRCHashQueue2	
	rešenje	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
41	Opt.	15	0.17	23.88%	0.22	0.773	0.16	1.063
42	Opt.	31	0.39	47.81%	0.39	1.000	0.38	1.026
43	Opt.	15	0.22	23.88%	0.16	1.375	0.22	1.000
44	Opt.	5	0.06	14.00%	0.11	0.545	0.11	0.545
51	Opt.	15	0.17	23.88%	0.22	0.773	0.22	0.773
61	Opt.	15	0.22	23.88%	0.16	1.375	0.22	1.000
62	Opt.	31	0.39	47.81%	0.39	1.000	0.38	1.026
63	Opt.	15	0.22	23.88%	0.17	1.294	0.22	1.000
64	Opt.	5	0.06	14.00%	0.11	0.545	0.11	0.545
71	Opt.	15	0.22	23.88%	0.22	1.000	0.22	1.000
72	Opt.	31	0.39	47.81%	0.39	1.000	0.38	1.026
73	Opt.	15	0.22	23.88%	0.22	1.000	0.16	1.375
74	Opt.	5	0.06	14.00%	0.06	1.000	0.05	1.200

Tabela C.26 Instance 81-104

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
81	42	0.49	34.56%	0.55	0.891	0.54	0.907
82	56	0.71	40.35%	0.71	1.000	0.71	1.000
83	30	0.44	24.65%	0.38	1.158	0.38	1.158
84	23	0.33	18.17%	0.33	1.000	0.33	1.000
91	42	0.55	34.56%	0.49	1.122	0.55	1.000
92	56	0.72	40.35%	0.71	1.000	0.72	1.000
93	30	0.39	24.65%	0.38	1.026	0.39	1.000
94	23	0.33	18.17%	0.33	1.000	0.33	1.000
101	42	0.55	34.56%	0.55	1.000	0.50	1.100
102	56	0.72	40.35%	0.71	1.014	0.66	1.091
103	30	0.44	24.65%	0.38	1.158	0.38	1.158
104	23	0.28	18.17%	0.28	1.000	0.27	1.037

Tabela C.27 Instance 111-134

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
111	69	1.10	19.34%	1.05	1.048	1.04	1.058
112	99	1.54	27.18%	1.53	1.007	1.53	1.007
113	198	3.08	41.99%	2.91	1.058	2.91	1.058
114	151	2.36	43.75%	2.20	1.073	2.20	1.073
121	69	1.04	19.34%	1.04	1.000	1.10	0.945
122	99	1.54	27.18%	1.54	1.000	1.48	1.041
123	198	3.08	41.99%	2.91	1.058	2.91	1.058
124	151	2.36	43.75%	2.20	1.073	2.25	1.049
131	69	1.10	19.34%	1.09	1.009	1.10	1.000
132	99	1.54	27.18%	1.54	1.000	1.49	1.034
133	198	3.07	41.99%	2.91	1.055	2.92	1.051
134	151	2.36	43.75%	2.25	1.049	2.20	1.073

Tabela C.28 Instance A-C, MO, MP i MQ

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
A	222	30.81	31.87%	22.35	1.379	22.35	1.379
B	748	94.31	37.17%	64.37	1.465	64.31	1.466
C	572	65.91	37.31%	45.54	1.447	45.53	1.448
MO1	180	5.17	30.96%	4.61	1.121	4.56	1.134
MO2	110	3.07	23.73%	2.80	1.096	2.80	1.096
MO3	67	1.70	10.82%	1.76	0.966	1.76	0.966
MO4	95	2.58	17.23%	2.53	1.020	2.47	1.045
MO5	94	2.53	17.26%	2.42	1.045	2.47	1.024
MP1	168	8.51	20.27%	7.85	1.084	7.86	1.083
MP2	213	10.66	26.42%	9.45	1.128	9.45	1.128
MP3	178	8.68	23.18%	7.86	1.104	7.85	1.106
MP4	158	7.75	22.21%	7.08	1.095	7.08	1.095
MP5	184	9.45	24.43%	8.41	1.124	8.35	1.132
MQ1	228	17.79	20.81%	15.65	1.137	15.60	1.140
MQ2	287	21.97	24.90%	19.05	1.153	19.00	1.156
MQ3	270	21.09	28.11%	17.68	1.193	17.63	1.196
MQ4	294	23.67	29.13%	19.44	1.218	19.45	1.217
MQ5	207	15.44	19.13%	14.01	1.102	13.95	1.107

Tabela C.29 Instance MR i MS

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
MR1	460	63.27	27.57%	50.91	1.243	50.86	1.244
MR2	546	75.64	32.60%	58.61	1.291	58.50	1.293
MR3	470	65.69	27.35%	53.06	1.238	53.00	1.239
MR4	394	51.96	23.11%	43.89	1.184	43.88	1.184
MR5	356	48.01	23.51%	40.32	1.191	40.26	1.192
MS1	786	239.53	25.83%	189.77	1.262	184.93	1.295
MS2	837	251.34	28.23%	195.70	1.284	195.48	1.286
MS3	644	199.27	21.61%	163.74	1.217	163.56	1.218
MS4	1010	303.62	30.21%	232.61	1.305	232.39	1.307
MS5	1012	294.84	32.78%	223.05	1.322	222.84	1.323

C.4.2 Rezultati na UNDP

Tabela C.30 Instance MA, MB i MC

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
MA1	16	0.28	23.88%	0.33	0.848	0.33	0.848
MA2	17	0.38	27.44%	0.33	1.152	0.33	1.152
MA3	26	0.49	23.56%	0.49	1.000	0.44	1.114
MA4	18	0.33	19.47%	0.33	1.000	0.32	1.031
MA5	18	0.33	18.32%	0.33	1.000	0.33	1.000
MB1	45	1.49	11.48%	1.37	1.088	1.42	1.049
MB2	191	5.55	39.50%	4.62	1.201	4.56	1.217
MB3	107	3.13	35.67%	2.58	1.213	2.58	1.213
MB4	47	1.54	15.92%	1.43	1.077	1.43	1.077
MB5	288	8.52	53.60%	6.31	1.350	6.32	1.348
MC1	82	3.46	15.61%	3.40	1.018	3.40	1.018
MC2	70	3.19	12.37%	3.18	1.003	3.18	1.003
MC3	64	2.36	15.51%	2.37	0.996	2.30	1.026
MC4	74	2.86	14.03%	2.80	1.021	2.80	1.021
MC5	468	19.61	62.53%	12.25	1.601	12.19	1.609

Tabela C.31 Instance MD i ME

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
MD1	79	5.11	11.28%	4.78	1.069	4.78	1.069
MD2	85	5.55	12.58%	5.22	1.063	5.22	1.063
MD3	129	6.70	23.17%	5.77	1.161	5.77	1.161
MD4	125	6.70	21.78%	5.82	1.151	5.82	1.151
MD5	88	5.60	14.56%	5.11	1.096	5.11	1.096
ME1	1283	111.44	59.64%	60.97	1.828	60.97	1.828
ME2	1760	133.64	52.27%	81.57	1.638	81.68	1.636
ME3	184	16.59	26.82%	13.29	1.248	13.29	1.248
ME4	129	13.95	11.48%	12.74	1.095	12.74	1.095
ME5	1452	99.25	59.20%	58.71	1.691	58.77	1.689

C.4.3 Rezultati na ISP

Tabela C.32 Instance AA, AB i AC

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
AA1	69	3.02	33.66%	2.42	1.248	2.47	1.223
AA2	59	2.58	21.93%	2.36	1.093	2.30	1.122
AA3	114	5.00	48.43%	3.46	1.445	3.46	1.445
AA4	50	2.26	21.02%	2.03	1.113	2.03	1.113
AA5	85	3.68	40.07%	2.80	1.314	2.74	1.343
AB1	60	2.64	26.79%	2.25	1.173	2.26	1.168
AB2	52	2.31	19.32%	2.09	1.105	2.14	1.079
AB3	52	2.31	22.72%	2.04	1.132	2.09	1.105
AB4	54	2.41	25.60%	2.15	1.121	2.08	1.159
AB5	122	5.22	48.21%	3.62	1.442	3.63	1.438
AC1	58	2.53	22.81%	2.26	1.119	2.25	1.124
AC2	68	2.92	25.86%	2.58	1.132	2.58	1.132
AC3	64	2.81	26.55%	2.42	1.161	2.41	1.166
AC4	96	4.23	44.25%	3.07	1.378	3.02	1.401
AC5	181	7.64	56.64%	4.84	1.579	4.84	1.579

Tabela C.33 Instance AD, AE i AF

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
AD1	77	3.35	30.36%	2.81	1.192	2.80	1.196
AD2	167	7.14	52.14%	4.78	1.494	4.77	1.497
AD3	66	2.85	24.78%	2.58	1.105	2.53	1.126
AD4	169	7.36	53.98%	4.78	1.540	4.72	1.559
AD5	69	2.92	24.09%	2.64	1.106	2.58	1.132
AE1	115	4.83	43.72%	3.63	1.331	3.62	1.334
AE2	107	4.56	39.52%	3.51	1.299	3.52	1.295
AE3	198	8.46	54.11%	5.55	1.524	5.54	1.527
AE4	103	4.50	33.42%	3.68	1.223	3.68	1.223
AE5	71	3.03	27.86%	2.58	1.174	2.58	1.174
AF1	110	4.67	41.30%	3.57	1.308	3.51	1.330
AF2	597	23.78	63.05%	14.17	1.678	14.12	1.684
AF3	231	9.67	53.54%	6.43	1.504	6.37	1.518
AF4	889	36.97	59.39%	22.52	1.642	22.41	1.650
AF5	136	5.55	40.32%	4.28	1.297	4.23	1.312

Tabela C.34 Instance AG i AH

Instanca	GA bez keširanja			CRCHashQueue		CRCHashQueue2	
	gener.	vreme (s)	ponov. jedinke	vreme (s)	faktor ubrzanja	vreme (s)	faktor ubrzanja
AG1	88	3.57	32.85%	3.02	1.182	3.02	1.182
AG2	1712	67.62	55.76%	44.05	1.535	43.83	1.543
AG3	1166	46.19	62.96%	27.63	1.672	27.46	1.682
AG4	96	3.96	37.01%	3.13	1.265	3.13	1.265
AG5	750	29.16	63.43%	17.36	1.680	17.36	1.680
AH1	3 143	119.9	56.73%	78.70	1.524	78.32	1.532
AH2	1 955	75.74	49.69%	53.22	1.423	53.00	1.429
AH3	313	12.14	46.92%	8.95	1.356	8.84	1.373
AH4	3 205	123.5	58.71%	78.65	1.571	78.38	1.577
AH5	1 060	39.99	61.87%	24.72	1.618	24.72	1.618

Dodatak D. KARAKTERISTIKE MPI STANDARDA

U ovom dodatku će biti prikazana C verzija MPI standarda verzije 1.1 iz 1995. godine. Većina MPI funkcija vraća celobrojnu vrednost (int), tako da to nećemo navoditi, osim ako vraćaju vrednost nekog drugog tipa. Zbog ograničenog prostora detaljnije su opisane samo konstrukcije koje su korišćene za realizaciju PGANP implementacije, a ostale funkcije su samo navedene. Detaljan opis svih konstrukcija MPI standarda i nekih saveta za njegovo uspešno korišćenje pri paralelizaciji se može naći u knjigama [Gro94] i [MPI95].

D.1 Opis MPI standarda

Pri definisanju MPI standarda korišćene su već postojeće konstrukcije raznih paralelnih programskih sistema koje su se dokazale u praktičnoj primeni. One su uklopljene u jednu celinu, tako da je postignuta maksimalna izražajnost paralelnog programiranja, uz očuvanje efikasnosti. Neki od najvažnijih aspekata MPI standarda su:

- Grupisanje procesa u jednu celinu i pojam komunikatora koji fleksibilno opisuje grupu procesa i njihov zajednički komunikacioni kontekst;
- Mogućnost zadavanja virtuelne topologije višeprocesorskog računara od strane korisnika;
- Tipovi podataka koji se koriste za međuprocesorsku komunikaciju mogu biti sistemski ili ih može definisati sam korisnik;
- Efikasne pojedinačne i kolektivne komunikacije;
- Globalne operacije koje se izvršavaju na većem broju procesa.

D.1.1 Grupe procesa i komunikatori

Grupisanje procesa u jednu celinu je važna karakteristika koja obezbeđuje efikasno paralelno izvršavanje. Grupe procesa se mogu koristiti na dva osnovna načina:

- Označavanje procesa koji učestvuju u kolektivnoj komunikaciji;
- Pomoć u podeli poslova na pojedinačne procese, gde procesi u različitim grupama izvršavaju različite zadatke.

Grupe mogu biti kreirane i brisane u svakom trenutku, a svaki proces može pripadati istovremeno u više grupa. Članovi grupe se ne mogu menjati u toku rada programa, već se u tom slučaju mora formirati nova grupa. Ova operacija može biti izvršena lokalno ili kolektivno podelom neke postojeće grupe.

Ovako definisan pojam grupe nije dovoljan, pa je u takvom kontekstu moguće da poruka poslata u jednoj fazi programa bude pogrešno primljena u

nekoj drugoj fazi. Zbog toga je definisan kontekst komunikacije, koji obezbeđuje mogućnost različitih vrsta poruka između istih procesa.

Komunikator predstavlja grupu procesa sa definisanim kontekstom komunikacije, i on je osnovna konstrukcija MPI standarda. Po oblasti dejstva se komunikatori mogu podeliti na lokalne i globalne. Lokalni komunikator vrši komunikaciju unutar grupe, dok globalni komunikatori mogu vršiti komunikaciju između procesa koji pripadaju različitim grupama. Osim oblasti delovanja, ista su sva ostala svojstva lokalnih i globalnih komunikatora, pa se oni mogu potpuno ravnopravno primenjivati u svim primenama.

Pojam grupe procesa je eksplicitno definisan, pa su dozvoljene manipulacije sadržajem i ostalim osobinama postojećih grupa. Nasuprot tome, kontekst komunikacije nije spolja dostupan, pa se ne mogu vršiti nikakve eksplicitne operacije nad njim, već samo implicitno preko odgovarajućeg komunikatora. Zbog toga poruka poslata preko nekog komunikatora, može biti primljena samo od procesa koji ima komunikator sa odgovarajućim kontekstom.

D.1.2 Virtuelne topologije

Osnovni model paralelizacije MPI standarda podrazumeva mogućnost komunikacije svakog procesora sa svim ostalim. Međutim ovakav model nije uvek pogodan, naročito u slučajevima kada je paralelna aplikacija posebno projektovana za određeni tip arhitekture. U tim slučajevima MPI standard omogućuje korišćenje virtuelne topologije definisane od strane korisnika. Virtuelna topologija koja dobro prati fizičke veze između procesora je važan preduslov za postizanje dobrih performansi cele paralelne aplikacije, naročito u slučaju velike međuprocesorske komunikacije.

MPI podržava zadavanje virtuelnih topologija na dva načina: grafom veza između procesa (procesora) ili n-dimenziona mreža u Decartes-ovim koordinatama. Prvi način je opštiji, i može se definisati proizvoljna topologija, ali zaheva veliki trud oko opisa veza između procesa. Drugi način je specifičan samo za date mrežne arhitekture, ali se lako opisuje.

D.1.3 Tipovi podataka za međuprocesorsku komunikaciju

MPI sistemski podržava međuprocesorsku komunikaciju nad većim brojem osnovnih tipova podataka. Pošto se svi izvedeni tipovi podataka sastoje od osnovnih tipova, na taj način se mogu obezbediti sve potrebe za međuprocesorskom komunikacijom.

Međutim iako je ovaj način funkcionalan, u praksi se pokazao kao relativno spor. To je posledica korišćenja raznih protokola za prenošenje poruka, koji obezbeđuju sigurnost i fleksibilnost međuprocesorske komunikacije. Pošto svaka poruka ima svoje zaglavlje određene fiksne dužine i zahteva neko konstantno vreme za prijem, mnogo je efikasnije slati manji broj poruka veće dužine, nego obratno.

Zbog toga je MPI standard predvideo slanje celokupnih nizova osnovnih tipova podataka u jednoj poruci, umesto slanja član po član. Međutim, ne mogu se svi složeniji podaci opisati nizom, pa je bilo nužno efikasno realizovati i njihov prenos. U ovom slučaju nije moguće direktno slanje cele strukture, jer operativni sistem rezerviše memorijski prostor onako kako mu trenutno najviše odgovara, pa bi prijemni proces smestio promenljive na pogrešna mesta. Dodatni problem predstavlja to što podaci u istoj strukturi ne moraju biti

smešteni u neprekidnom memorijskom bloku, već mogu biti raštrkani po memoriji.

Imajući u vidu prethodne činjenice, MPI standard je predvideo u međuprocesorskoj komunikaciji i mogućnost korišćenja složenih tipova podataka definisanih od strane korisnika. Time se mogu na siguran način slati čitave složene strukture u jednoj poruci, gde će prijemni proces svaku promenljivu smestiti na pravo mesto, nezavisno od načina alokacije memorijskog prostora.

I ovakav pristup bi imao neke nedostatke, pošto prijemni proces mora znati unapred strukturu podataka u poruci koju prima. Na taj način ne bi bilo moguće, u jednoj poruci, slati nizove promenljive dužine, jer pri prijemu datom procesu mora biti unapred poznat broj članova niza koga prima. Ova anomalija je prevaziđena mogućnošću pakovanja u neprekidne uzastopne memorijske lokacije pre slanja, a po prijemu se vrši postepeno raspakivanje. Pri tome nije neophodno zadavanje unapred dužine cele poruke, već samo gornje granice te dužine.

D.1.4 Pojedinačne komunikacije

MPI podržava pojedinačne komunikacije gde je prosleđivanje i selekcija poruka bazirana na rangovima procesa, tagu poruke i komunikacionom kontekstu. Pri zadavanju rangova i taga poruke moguće su i opšte vrednosti (wildcards) koje omogućuju prijem poruke od svih procesa ili sa proizvoljnim tagom. Komunikacioni kontekst nije moguće zadati kao opštu vrednost, već on zavisi samo od konkretnog komunikatora. Pri svakoj komunikaciji su procesi za slanje odnosno prijem poruke jednoznačno određeni grupom procesa i rangom datog procesa unutar te grupe. Lokalni komunikator vrši komunikaciju unutar iste grupe, dok globalni komunikator vrši prosleđivanje poruke između procesa koji pripadaju različitim grupama.

Postoje četiri načina pri pojedinačnim komunikacijama: standardni, baferisani, po zahtevu prijemnog procesa i sinhroni. Svi ovi načini komunikacije se mogu podeliti dalje po načinu izvršavanja, gde mogu biti blokirajući ili neblokirajući. To ukupno daje osam različitih mogućnosti za međuprocesorsku komunikaciju što može da zadovolji i najzahtevnije potrebe.

Standardni način komunikacije vrši slanje i prijem poruka asinhrono, odnosno oni se mogu izvršavati nezavisno i u različitim vremenskim momentima. Na taj način po završetku slanja poruke pošiljalac nastavlja dalje sa radom, nezavisno od toga kada će poruka biti primljena od strane prijemnog procesa.

Baferisani mod podrazumeva da se data komunikacija ne odvija direktno, već preko pomoćnog bafera. I u standardnom modu je moguće korišćenje baferisanja, ali sistem sam bira način na koji se poruka prosleđuje, kako mu najviše odgovara u datom momentu. U tom slučaju sistem sam vodi računa o svemu, pa i o alokaciji memorije potrebne za bafer. Za razliku od toga, u baferisanom modu, programer mora sam eksplicitno da odvoji memorijski prostor za bafer.

Ako je poruka poslata u sinhronom modu, slanje poruke počinje tek u momentu kada su oba procesa (pošiljalac i primalac) spemna za komunikaciju. Ukoliko je samo jedan od njih spreman, on čeka na spremnost drugog procesa.

Za razliku od prethodnih načina komunikacije, gde pošiljalac započinje komunikaciju, u komunikaciji po zahtevu, primalac prvi šalje zahtev za prijem

podataka, a zatim čeka pošiljaoca da mu ih pošalje. Pošiljalac izvršava datu naredbu slanja podataka samo ako je prethodno primio odgovarajući zahtev.

Ako je komunikacija blokirajuća, pri slanju odgovarajući proces nastavlja sa radom tek po završetku cele operacije, kada su odgovarajući podaci poslati, pa je moguće ponovo koristi njihove memorijske lokacije. Slično tome, pri prijemu poruke, dati proces nastavlja sa radom tek pošto je primio celu poruku.

Neblokirajući način inicira komunikaciju, a odgovarajući proces odmah zatim nastavlja sa daljim radom, uporedo sa slanjem ili prijemom date poruke. Pri tome je postavljen poseban objekat, na osnovu koga je moguće kontrolisati tok i završetak tekuće komunikacije. Ovaj način može u nekim slučajevima biti pogodan, iako zahteva veću kontrolu od strane programera.

D.1.5 Kolektivne komunikacije

Kolektivne komunikacije obezbeđuju fleksibilan način za razmenu podataka između procesora koji pripadaju istoj grupi. Za razliku od pojedinačnih komunikacija poruke u kolektivnim komunikacijama nemaju tag i mogu se koristiti samo lokalni komunikatori, pošto to jedino ima smisla. Odgovarajuća operacija za kolektivnu komunikaciju mora biti pozvana od svih procesa u datoj grupi, tako da se argumenti odgovarajuće funkcije moraju slagati po tipu i vrednosti.

Sve operacije za kolektivnu komunikaciju su realizovane isključivo kao blokirajuće. Svaki proces po završetku svog udela u celoj komunikaciji može da nastavi dalje sa radom. U nekim MPI implementacijama su funkcije za kolektivne komunikacije realizovane korišćenjem funkcija za pojedinačnu komunikaciju, ali se takav pristup izbegava u novijim MPI implementacijama, koje teže vrhunskim performansama.

D.1.6 Globalno izračunavanje

MPI osim lokalnog izračunavanja na datom procesoru (procesu) omogućuje i globalno izračunavanje neke operacije na svim procesima u datoj grupi. Pored nekih osnovnih operacija obezbeđenih od sistema, moguće je i zadavanje operacija koje definiše sam korisnik. Globalne operacije se mogu izvršavati kako na pojedinačnim, tako i na vektorskim podacima. Iako se globalno izračunavanje može emulirati lokalnim izračunavanjima na datim procesima uz potrebnu međuprocesorsku komunikaciju, ovakav pristup ima vrlo kratak zapis uz maksimalno očuvanje efikasnosti izvršavanja.

D.2 MPI konstrukcije korišćene u implementiranju PGANP

Najveća pažnje će biti poklonjena delovima koji su potrebni za paralelnu GA implementaciju, koju je autor razvio koristeći MPI standard. Ove konstrukcije će biti nešto detaljnije opisane, a pregled ostalih se može videti u narednom odeljku. Zbog kraćeg zapisa, u prikazu funkcija će biti dati samo odgovarajući argumenti, dok se tipovi svih argumenata MPI funkcija mogu naći na kraju ovog dodatka.

D.2.1 Inicijalizacija

Inicijalizacija u MPI standardu se vrši funkcijom *MPI_Init*(*argc*, *argv*). Ona se mora pozvati pre korišćenja bilo koje druge MPI funkcije, i to najviše jednom. Verzija za ANSI C prihvata *&argc* i *&argv* koje su dobijene od *main*() funkcije.

MPI_Finalize(*void*) je poslednja MPI funkcija u programu, i posle nje nije moguće korišćenje MPI funkcija, čak ni *MPI_Init*(). Korisnik mora da bude siguran da su sve komunikacije gotove pre nego što se pozove *MPI_Finalize*().

MPI_Initialized(*flag*) određuje da li je *MPI_Init*() već ranije pozvana, ili ne, što se vidi iz vrednosti parametra *flag*.

MPI_Abort(*comm, errorcode*) prekida sve procese u grupi komunikatora *comm*, a *MPI_Comm_free*(*comm*) markira komunikator *comm* za oslobađanje (dealokaciju). Posle oslobađanja pokazivaču se dodeljuje vrednost *MPI_COMM_NULL*, a fizički se oslobađa tak kada nema aktivnih referenci na njega. Primenjuje se i na lokalne i na globalne komunikatore.

MPI_Comm_size(*comm, size*) parametrom *size* vraća broj procesa u komunikatoru *comm.*

MPI_Comm_rank(*comm, rank*) parametrom *rank* vraća indeks (rang) tekućeg procesa u komunikatoru *comm*.

MPI_COMM_WORLD predstavlja početni komunikator obezbeđen od samog MPI sistema koji sadrži sve procese, i dozvoljava proizvoljnu komunikaciju između njih. Broj procesa tokom rada MPI programa ostaje konstantan, i unapred je zadat konfiguracionom datotekom.

D.2.2 Korisnički tipovi podataka za komunikaciju

Za međuprocesorsku komunikaciju se osim osnovnih tipova podataka, mogu se koristiti i tipovi koje je definisao korisnik.

Posle definisanja, a pre samog korišćenja novog korisničkog tipa potrebno ga je ozvaničiti (committed), funkcijom *MPI_Type_commit*(*datatype*). Posle toga je spreman za upotrebu u komunikaciji. I pre kao i posle ozvaničavanja novog tipa moguće ga je koristiti pri pravljenju novih tipova. Osnovne tipove nije potrebno ozvaničavati.

MPI_Type_free(*datatype*) oslobađa objekat *datatype* i dodeljuje mu vrednost *MPI_DATATYPE_NULL*. Svaka komunikacija koja koristi dati tip će biti normalno završena. Izvedeni tipovi datog tipa će ostati sačuvani.

Neke od osnovnih informacija o korisničkim tipovima su adresa početka odnosno kraja objekta datog tipa (bottom), njegova dužina (size) i struktura podataka unutar njega (extent). Prva se može dobiti preko *MPI_BOTTOM, MPI_Type_lb*(*datatype*, *lbound*) ili *MPI_Type_ub*(*datatype*, *ubound*). Veličina datog objekta izražena u bajtovima može se dobiti preko funkcije *MPI_Type_size*(*datatype*, *size*), a struktura podataka unutar korisničkog tipa pomoću *MPI_Type_extent*(*datatype*, *extent*). Pomoću funkcije *MPI_Address*(*location*, *address*) se može na osnovu lokacije u memoriji naći odgovarajuća MPI adresa.

Funkcija *MPI_Type_contiguous*(*count, old_type, new_type*) pravi novi tip *new_type* kopiranjem i spajanjem *count* tipova *old_type*. Dužina svake kopije je određena sa strukturom podataka unutar starog tipa (extent).

MPI_Type_vector(*count*, *blocklen*, *stride*, *old_type*, *new_type*) pravi novi tip spajanjem *count*blocklen* delova tipa *old_type*, sa razmakom *stride* elemenata između glavnih delova. *MPI_Type_hvector*() radi isto samo je razmak *stride* u bajtovima.

MPI_Type_struct(*count*, *blocklens*, *inds*, *old_types*, *new_type*) pravi složeniju strukturu od *count* postojećih tipova. Broj kopija svakog od tipova u

nizu *old_types*[i] je dat u nizu *blocklens*[i], a smešta se na poziciju *inds*[i]. Na kraju se sve smešta u tip *new_type*.

Primer D.1. Neka je tip *type1* = {(double,0), (char,8)} sa *extent* = 16. Ako su T = (*MPI_FLOAT*, *type1*, *MPI_CHAR*), B = (2,1,3) i D = (0, 16, 26), tada poziv *MPI_Type_struct*(3,B,D,T,*new_type*) vraća vrednost:

{(float,0), (float,4), (double,16), (char,24), (char,26), (char,27), char,28)}.

Ponekad je potrebno pakovati podatke ako nisu zadati u uzastopnim memorijskim lokacijama. Funkcija *MPI_Pack*(*inbuf, incount, datatype, outbuf, outsize, position, comm*) pakuje poruku u baferu za slanje *inbuf*, tipa *datatype* čija je dužina *incount,* u bafer *outbuf*, dužine *outsize* izražene u bajtovima. Promenljiva *position* na početku predstavlja lokaciju na koju se u izlaznom baferu smešta pakovana poruka, a na kraju se vraća prva slobodna lokacija. Parametar *comm* predstavlja komunikator u kome će se slati data poruka. Funkcija vraća vrednosti preko *outbuf* i *position*.

MPI_Unpack(*inbuf, insize, position, outbuf, outcount, datatype, comm*) raspakuje poruku *inbuf* tipa *datatype* veličine *insize* bajtova u bafer *outbuf* veličine *outcount*. Ulazna vrednost *position* je početak pakovane poruke, a izlazna je prva slobodna pozicija iza pakovane poruke.

MPI_Pack_size(*incount*, *datatype*, *comm*, *size*) vraća promenljivom *size* dužinu pakovane poruke, odnosno povećanje promenljive *position* koje bi nastalo primenom funkcije *MPI_Pack*().

MPI_Type_indexed(*count, blocklens, inds, old_type, new_type*) kopira stari tip *old_type* u niz od *count* blokova, gde je svaki blok višestruka kopija starog tipa. *MPI_Type_hindexed*() radi isto samo su dužine blokova *blocklens* zadate u bajtovima.

D.2.3 Pojedinačne komunikacije

Funkcija *MPI_Send*(*buffer, count, datatype, dest, tag, comm*) vrši blokirajuće slanje poruke na standardan način. Poruka se uzima sa memorijske lokacije *buffer*, i sadrži *count* delova tipa *datatype*. Poruka sa datim tagom se šalje procesu sa rangom *dest* u tekućoj grupi koja je zadata komunikatorom *comm*. Odgovarajuće funkcije za baferisano, sinhrono i slanje po zahtevu su *MPI_Bsend*(), *MPI_Ssend*() i *MPI_Rsend*(). Dodavanjem prefiksa *I* zadaju se odgovarajuće varijante neblokirajućeg slanja poruka (*MPI_Isend*(), *MPI_Issend*() i *MPI_Irsend*()), sa dodatnim parametrom *prequest* koji sadrži pokazivač na strukturu za praćenje toka date komunikacije.

Standardni način prijema poruke od procesa *source* se vrši funkcijom *MPI_Recv*(*buffer, count, datatype, source, tag, comm, status*), i to samo u slučaju ako parametri *comm* i *tag* odgovaraju poslatoj poruci. Ona se smešta u memoriju na lokaciju *buffer*, i sastoji se od *count* podataka tipa *datatype.* Status date komunikacije se može videti preko parametra *status.* Ostali načini prijema poruke se ostvaruju funkcijama sa istim prefiksima, ako i kod slanja poruke. Ukoliko ne znamo tačno od kog procesa možemo očekivati prijem poruke i sa kakvim tagom, na mesta odgovarajućih parametara možemo zadati vrednosti *MPI_ANY_SOURCE* ili *MPI_ANY_TAG*. Na taj način se ne vrši selekcija poruka po datom parametru, pa će dati proces primiti poruku od svakog procesa u datoj grupi ili poruku sa proizvoljnim tagom.

Pri neblokirajućem prijemu poruke mogu poslužiti funkcije *MPI_Wait*(*prequest, status*) i *MPI_Test*(*prequest, flag, status*). Prva zaustavlja izvršavanje sve dok zahtev *prequest* za prijemom poruke koja je u toku, ne bude završen. Druga testira kako teče prijem date poruke i koliko je elemenata već primljeno.

MPI_Get_elements(*status, datatype, count*) vraća koliko je elemenata *count* primljeno nekom neblokirajućom komandom prijema poruke tipa *datatype* u tom trenutku. Pri tome se čita promenljiva *status*. Slična je funkciji *MPI_Get_count*(), sa time što za razliku od nje vraća broj primeljenih elemenata, a ne broj primljenih blokova.

D.2.4 Kolektivne komunikacije

MPI_Bcast(*buffer, count, type, root, comm*) šalje poruku od procesa sa rangom *root* do svih ostalih procesa u grupi. Svi argumenti moraju biti isti za sve procese.

MPI_Gather(*sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm*) prosleđuje poruke od svakog procesa ka *root* procesu. On prima sve poruke i smešta ih u poretku rangova. Za procese različite od *root* vrednost *recvbuf* se ignoriše. Promenljive *sendbuf* i *sendcount* označavaju bafer za slanje za sve procese, a *recvbuf* i *recvcount* prijemni bafer procesa *root*, i moraju da budu isti. Nasuprot tome, parametri *sendtype* i *recvtype* mogu i da se razlikuju. Funkcija *MPI_Gatherv*() je vrlo slična, samo poruke koje se primaju od svakog procesa ne moraju biti iste dužine

MPI_Scatter(*sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm*) je inverzna operacija od *MPI_Gather* i šalje *sendcount* elemenata iz *sendbuf* bafera svakom od procesa. Tip podataka za slanje je *sendtype* a svaki od procesa prima *recvcount* elemenata tipa *recvtype*. Parametri *root* i *comm* moraju imati iste vrednosti za sve procese. Funkcija *MPI_Scatterv*() je vrlo slična, samo poruke koje se šalju svakom od procesa ne moraju biti iste dužine.

U nekim slučajevima je potrebno sinhronizovati rad procesora u paralelnom programu. Jedan od načina za sinhronizaciju je funkcija *MPI_Barrier*(*comm*) koja blokira tekući proces sve dok svi članovi grupe ne pozovu datu funkciju, i tek tada procesi u datoj grupi nastavljaju izvršavanje.

D.2.5 Globalno izračunavanje

Kao što smo već napomenuli MPI standard omogućuje globalno izračunavanje za sve procese u datoj grupi. Pri tome svaki proces poseduje neke skalarne ili vektorske veličine, koje su argumenti date operacije, a rezultat se prosleđuje nekom od procesa ili svim procesima.

MPI_Reduce(*sendbuf, recvbuf, count, datatype, op, root, comm*) kombinuje elemente date u *sendbuf* svakog procesa u grupi operacijom *op* i izračunatu vrednost vraća u *recvbuf* bafer procesa u datoj grupi sa rangom *root.* Ako *sendbuf* sadrži više članova (vektorski podaci) tada se operacija primenjuje na svaki od njih pojedinačno.

MPI_Allreduce(*sendbuf, recvbuf, count, datatype, op, comm*) na isti način globalno izračunava vrednosti, ali ih prosleđuje svim procesima u datoj grupi. Na sličan način *MPI_Reduce_scatter*(*sendbuf, recvbuf, recvcounts, datatype, op, comm*) vrši operaciju po elementima vektora na *sendbuf* memorijskoj lokaciji

svakog procesa, a zatim elemente rezultujućeg vektora deli na delove, i šalje redom procesima od 0, 1, ... n-1.

Osim osnovnih sistemskih operatora, mogu se koristiti i operatori koje je definisao korisnik. Oni se prethodno moraju kreirati naredbom *MPI_Op_create*(*opfunc, commute, pop*), gde je *opfunc* definicija date korisničke funkcije, a *pop* je pokazivač na kreirani operator. Ukoliko nam neki od korisnički definisanih operatora više nije potreban, može se ukloniti naredbom *MPI_Op_free*(*pop*).

LITERATURA

[Abr95] **Abramson D., De Silva A.**, "A Parallel Interior Point Algorithm and its Application to Facility Location Problems", *Research Report CIT:95-12*, School of Computing and Information Technology, Griffith University, Nathan (1995).

[Ack87] **Ackley D.H.**, "An empirical study of bit vector function optimization", In: *Genetic algorithms and Simulated Annealing*, Davis L. (ed), chapter 13, pp. 170-204 (1987).

[Aik85] **Aikens C.H.**, "Facility location models for distribution planning", *European Journal of Operational Research,* Vol. 22, pp.263-279 (1985).

[Akl89] **Akl S.**, "Design and Analysis of Parallel Algorithms", Prentice-Hall International, Englewoods Cliffs, NJ (1989).

[Aln95] **Alander J.T.**, "Indexed bibliography of genetic algorithms in operations research", Report 94-1-OR, University of Vassa, Department of Information Technology and Production Economics (1995).
ftp://ftp.uwasa.fi/cs/report94-1/gaORbib.ps.Z

[AlS96] **Al-Sultan K., Hussain M., Nizami J**, "A genetic algorithm for the set covering problem", *Journal of the Operational Research Society*, Vol. 47, pp. 702-709 (1996).

[Alv92] **Alves M.L., Almeida M.T.**, "Simulated Annealing Algorithm for the Simple Plant Location Problem: A Computational Study", *Revista Investigação*, Vol. 12 (1992).

[Ant89] **Antonisse J.**, "A new interpretation of schema that overturns the binary encoding constraint", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 86-91 (1989).

[Aqe90] **Aqeev A.A., Beresnev V.S.**, "Polynomially Solvable Cases of the Simple Plant Location Problem", In: *Proceeding of the First Integer Programming and Combinatorial Optimization Conference*, Kannan R. and Pulleyblank W.R. (eds), University of Waterloo Press, ON, Canada, pp. 1-6 (1990)

[Ard95] **Aardal K., van Hoesel S.**, "Polyhedral Techniques in Combinatorial Optimization II: Computations", *Technical Report UU-CS-1995-42*, Department of Computer Science, Utrecht University, 44 p. (1995).
http://www.cs.ruu.nl/~aardal/survey1.ps

[Ard96a] **Aardal K., Labbe M., Leung J., Queyranne M.**, "On the Two-Level Uncapacitated Facility Location Problem", *Informs Journal of Computing,* Vol. 8, pp. 289-301 (1996).
http://smg.ulb.ac.be/Preprints/Aardal94_15.html

[Ard96b] **Aardal K., van Hoesel S.**, "Polyhedral Techniques in Combinatorial Optimization I: Theory", *Statistica Neerlandica*, Vol. 50, No. 1, pp. 3-26 (1996).
http://www.cs.ruu.nl/~aardal/survey2.ps

[Art97a] **Aarts E., Lenstra J.K.**, "Local Search in Combinatorial Optimization", John Wiley & Sons, Baffins Lane, Chichester, UK (1997).

[Art97b] **Aarts E.H.L., Korst J.H.M., van Laarhoven P.J.M.**, "Simulated annealing", In: *Local Search in Combinatorial* Optimization, Aarts E.H.L. and Lenstra J.K. (eds.), John Wiley & Sons Ltd., pp. 91-120 (1997).

[Bak98] **Baker M.A., Fox G.C.**, "MPI on NT: A Preliminary Evaluation of the Available Environments", *Lecture Notes in Computer Science*, Jose Rolim (Ed.), Parallel and Distributed Computing, Springer Verlag, Heidelberg, Germany, (1998).
ISBN 3-540 64359-1 (12th IPPS & 9th SPDP Conference, Orlando, USA).
http://www.sis.port.ac.uk/~mab/Papers/PC-NOW/

[Bar90] **Barcucci E., Pinzani R., Sprugnoli R.**, "Optimal Selection of Secondary Indexes", *IEEE Transactions on Software Engineering*, Vol. 16, pp. 32-38 (1990).

[Bas87] **Bastay G., Turesson B.O., Hellstrand J.**, "A comparison between decomposition methods for the discrete network design problem", LiU-MAT-EX-87-62, Linköping University, Sweden (1987).

[Bat93] **Battle D., Vose M.D.**, "Isomorphisms of genetic algorithms", *Artificial Intelligence*, Vol. 60, pp. 155-165 (1993).

[Bay91] **Bayer S.E., Wang L.**, "A genetic algorithm programming environment: Splicer". In: *Proceedings of the 1991 IEEE International Conference on Tools with Artificial Intelligence - TAI'91*, pp. 138-144, IEEE Computer Society Press, Los Alamitos (1991).

[BeD93a] **Beasley D.**, **Bull D.R.**, **Martin R.R.**, "An Overview of Genetic Algorithms, Part 1, Fundamentals", *University Computing*, Vol. 15, No. 2, pp. 58-69 (1993).
ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview1.ps

[BeD93b] **Beasley D.**, **Bull D.R.**, **Martin R.R.**, "An Overview of Genetic Algorithms, Part 2, Research Topics", *University Computing*, Vol. 15, No. 4, pp. 170-181 (1993).
ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview2.ps

[BeJ88] **Beasley J.E.**, "An algorithm for solving large capacitated warehouse location problems", *European Journal of Operational Research*, Vol. 33, No. 3, pp. 314-325 (1988).

[BeJ90a] **Beasley J.E.**, "OR Library: distributing test problems by electronic mail", *Journal of the Operational Research Society*, Vol. 41, pp. 1069-1072 (1990).

[BeJ90b] **Beasley J.E.**, "A Lagrangean heuristic for set-covering problems", *Naval Research Logistics*, Vol. 37, pp. 151-164 (1990).

[BeJ93] **Beasley J.E.**, "Lagrangean heuristic for location problems", *European Journal of Operational Research*, Vol. 65, No. 3, pp. 383-399 (1993).

[BeJ95] **Beasley J.E.**, "Lagrangean Relaxation", In: *Modern Heuristic Techniques for Combinatorial Problems,* Reeves C.R.(ed), McGraw-Hill, pp. 243-303 (1995).

[BeJ96a] **Beasley J.E.**, "Obtaining test problems via internet", *Journal of Global Optimization*, Vol. 8, pp. 429-433 (1996)
http://mscmga.ms.ic.ac.uk/info.html

[BeJ96b] **Beasley J.E., Chu P.C.**, "A genetic algorithm for the set covering problem", *European Journal of Operational Research,* Vol. 94, pp. 392-404 (1996).
http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Beasley.Chu.A_Genetic_Algorithm_for_the_Set_Covering_Problem.ps.gz

[Ber89] **Bertsekas D.P., Tsitsiklis J.N.**, "Parallel and Distributed Computation: Numerical Methods", *Prentice-Hall International,* 715 p (1989).

[Bëc91a] **Bëck T., Hoffmeister F., Schwefel H.P.**, "A Survey of Evolution Strategies", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 2-9 (1991).
ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/icga91.ps.gz

[Bëc91b] **Bëck T., Hoffmeister F.**, "Extended selection mechanisms in genetic algorithms", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 92-99 (1991).
ftp://lumpi.informatik.uni-dortmund.de/pub/ES/papers/icga91-2.ps.gz

[Bëc92a] **Bëck T.**, "Self-adaptation in Genetic Algorithms", In: *Proceedings of the First European Conference on Artificial Life*, MIT Press (1992).
ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/ecal92.ps.gz

[Bëc92b] **Bëck T.**, "A User Guide to GENEsYs 1.0", *Technical Report*, University of Dortmund, Department of Computer Science, Germany (1992).

[Bëc93] **Bëck T.**, "Optimal Mutation Rates in Genetic Search", In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 2-8 (1993).
ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/icga93.ps.gz

[Bla87] **Balakrishnan A.**, "LP Extreme Points and Cuts for the Fixed-Charge Network Design Problem", *Mathematical Programming*, Vol. 39, pp. 263-284, (1987).

[Bla89] **Balakrishnan A., Magnati T.L., Wong R.T.**, "A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design", *Operations Research*, Vol. 37, pp. 716-740 (1989).

[Bld95] **Belding T.C.**, "The Distributed Genetic Algorithm Revisited", In: *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, Morgan Kaufmann, San Mateo, Calif., pp. 114-121 (1995).
ftp://ftp.Germany.EU.net/pub/research/softcomp/EC/GA/papers/dga95.ps.gz

[Blm58] **Bellman R.E.**, "On a Routing Problem", Quarterly Applied Mathematics, Vol. 16, pp. 87-90 (1958).

[Blu97] **Baluja S., Davies S.**, "Using Optimal Dependency Trees for Combinatorial Optimization: Learning the Structure of the Search Space", *Report CMU-CS-97-107*, Carnegie Mellon University (1997).

[Bok87] **Booker L.**, "Improving Search in Genetic Algorithms", In: *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London, pp. 61-73 (1987).

[Bon85] **Bonanno R., Maio D., Tiberio P.**, "An approximation algorithm for secondary index selection in relational database physical design", *The Computer Journal*, Vol. 28, pp. 398-405 (1985).

[Brme95] **Bramel J., Simchi-Levi D.**, "A Location Based Heuristic for General Routing Problems", *Operations Research,* Vol. 43, No. 4, pp. 649-660 (1995).

[Brml91] **Bramlette M.F.**, "Initialisation, mutation and selection methods in genetic algorithms for function optimization", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 100-107 (1991).

[Brs88] **Brassard G., Bratley P.**, "Algorithms: Theory and Practice", Prentice-Hall Int., Englewoods Cliffs NJ (1988).

[Bum89] **Bum-Il L.**, "On the Uncapacitated Facility Location Problem", *PhD thesis*, Dept. of Business Management, Dan Kook University, South Corea (1989).

[Bur94] **Burns G.D., Daoud R.B., Vaigl J.R.**, "LAM: An Open Cluster Environment for MPI", Supercomputing Symposium '94, Toronto, Canada, June 1994.
http://www.mpi.nd.edu/lam/lam-mpi.paper.ps

[But94] **Butler R., Lusk E.**, "Monitors, Messages, and Clusters: The **p4** Parallel Programming System", *Parallel Computing,* Vol. 20 (1994).

ftp://info.mcs.anl.gov/pub/p4/p4-1.4.tar.Z

[Cal94] **Calkin R., Hempel R., Hoppe H., Wypior P.**, ”Portable Programming with the PARMACS Message-Passing Library”, *Parallel Computing*, Vol. 20 - *Special issue on message-passing interfaces*, pp. 615-632 (April 1994).

[Cap95a] **Caprara A., Fischetti M., Maio D.**, "Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design", *IEEE Transactions on Knowledge and Data Engineering,* Vol. 7, No. 6, pp. 955-967 (December 1995).

[Cap95b] **Caprara A., Salazar J.J.**, "A Branch-and-Cut Algorithm for the Index Selection Problem", *Technical Report OR-95-9*, DEIS Operations Research Group, University of Bologna, Italy (1995).
http://promet4.deis.unibo.it/~alberto/isp96.ps

[Cap96] **Caprara A., Salazar J.J.**, "A Branch-and-Cut Algorithm for a Generalization of the Uncapacitated Facility Location Problem", *TOP*, Vol. 4, pp. 135-163 (1996).

[Chd97] **Chudak F.A.**, "Improved approximation algorithms for uncapacitated facility location", In: *Lecture Notes in Computer Science,* Vol. 1412 (1997)
ftp://ftp.hpc.uh.edu/pub/ipco98/chudak.ps

[Che93] **Cherkassky B.V., Goldberg A.V., Radzik T.**, "Shortest paths algorithms: Theory and experimental evaluation", *Technical Report STAN-CS-93-1480*, Stanford University, Computer Science Department, Stanford, CA (1993).
ftp://theory.stanford.edu/pub/goldberg/stan-CS-93-1480.ps.gz

[CHI91] "CHIMP Concepts", *Edinburgh Parallel Computing Centre*, University of Edinburgh, UK, June 1991.

[CHI92] "CHIMP Version 1.0 Interface", *Edinburgh Parallel Computing Centre*, University of Edinburgh, UK, May 1992.

[Chu97a] **Chu P.C.**, "A genetic algorithm approach for combinatorial optimisation problems", *PhD thesis*, University of London, Imperial College of Science - The Management School (1997).
http://mscmga.ms.ic.ac.uk/pchu/ps/thesis.ps.gz

[Chu97b] **Chu P.C., Beasley J.E.**, "A genetic algorithm for the generalised assignment problem", *Computers and Operations Research*, Vol. 24, pp. 17-23 (1997).

[Coh91] **Cohoon J., Martin W., Richards D.**, "Genetic algorithms and punctuated equilibria in VLSI", In: *Parallel Problem Solving from Nature - PPSN,* Schwefel H.P. and Manner R. (eds.), Springer-Verlag, Berlin, pp. 134-144 (1991).

[Col91] **Collins R.J., Jefferson D.R.**, "Selection in massively parallel genetic algorithms", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 249-256 (1991).

[Com78] **Comer D.**, "The difficulty of optimum index selection", *ACM Transactions on Database Systems*, Vol. 3, pp. 440-445 (1978).

[Con90] **Conn A.R.**, **Cornuejols G.,** "A projection method for the uncapacitated facility location problem", *Mathematical Programming*, Vol. 46, pp. 273-298 (1990).

[Cook71] **Cook S.A.**, "The complexity of theorem-proving procedures", In: *Proceedings of Third Annual ACM Symposium on the Theory of Computing*, pp. 151-158 (1971).

[Cox91] **Cox L.A., Davis L., Qiu Y.**, "Dynamic anticipatory routing in circuit-switched telecommunication networks", In: *Handbook of Genetic Algorithms*, Davis L. (ed), pp. 124-143, Van Nostrad Reinhold, New York, NY (1991).

[Cre97] **Crescenci P., Kann V.,** "A compendium of NP optimization problems" (1997). *http://www.nada.kth.se/theory/problemlist.html*

[Crm90] **Cormen T.H., Leiserson C.E., Rivest D.L.**, "Introduction to Algorithms", MIT Press, Cambridge MA (1990).

[Crn82] **Cornuejols G., Thizy J.M.**, "Some facets of the Simple Plant Location Polytope", *Mathematical Programming*, Vol. 23, pp. 50-74 (1982).

[Crn90] **Cornuejols G., Nemhauser G.L., Wolsey L.A.**, "The uncapacitated facility location problem", In: *Discrete Location Theory*, eds: Mirchandani P.B. and Francis R.L., John Wiley & Sons, chap. 3, pp. 120-171 (1990).

[Cve96] **Cvetković D., Čangalović M., Dugošija Đ., Kovačević-Vujčić V., Simić S., Vuleta J.**, "Kombinatorna optimizacija: Matematička teorija i algoritmi", Društvo operacionih istraživača Jugoslavije, Beograd (1996).

[Čan96] **Čangalović M.**, "Opšte heuristike za rešavanje problema kombinatorne optimizacije", U: *Kombinatorna optimizacija: Matematička teorija i algoritmi*, str. 320-350 (1996).

[Dah94] **Dahlin M.D., Wang R.Y., Anderson T.E., Patterson D.A.**, "Cooperative Caching: Using Remote Client Memory to improve File System Performance", In: *Proceedings of the First Symposium on Operating System Design and Implementation*, pp. 267-280 (November 1994).
http://now.CS.Berkeley.EDU/~dahlin/papers/osdiDraft.ps

[Dar85] **Darvin Č.**, "Postanak vrsta", Nolit, Beograd (1985).

[Dav89] **Davis L.**, "Adapting operator probabilities in genetic algorithms", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 61-69 (1989).

[Dav91] **Davis L.**, "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York (1991).

[Dea85] **Dearing P.M.**, "Location problems", *Operations Research Letters*, Vol. 4, pp. 95-98 (1985).

[Dea92] **Dearing P.M., Hammer P., Simeone B.**, "Boolean and graph theoretic formulations of the simple plant location problem", *Transportation Science*, Vol. 26, pp. 138-148 (1992).

[Deb91] **Deb K., Goldberg D.E.**, "Analyzing deception in trap function", Technical Report 91009, IlliGal, December 1991.

[Dij59] **Dijkstra E.W.**, "A note on two problems in connexion with graphs", *Numerische Mathematik*, Vol. 1, pp. 269-271 (1959).

[DJo75] **De Jong K.E.**, "An analysis of the behavior of a class of genetic adaptive systems", *PhD thesis*, University of Michigan (1975).

[DJo89] **De Jong K.E.**, **Spears W.M.**, "Using Genetic Algorithms to Solve NP-Complete Problems", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 124-132 (1989).
ftp://ftp.aic.nrl.navy.mil/pub/spears/icga89.ps.Z

[DSi95] **De Simone C., Mannino C., Sassano A.**, "The Simple Plant Location Problem", *Informs*, New Orleans, TD4.3 (1995).
http://mjmi.engin.umich.edu/Conf/NO95/TALKS/TD4.3.html

[DSi96] **De Simone C., Mannino C.**, "Easy Instances of the Plant Location Problem", *Technical Report R-427*, Gennaio 1996, University of Roma, Italy (1996).
ftp://151.100.16.20/pub/OR/mannino/plant.ps.gz

[Don93] **Dongarra J.J., Hempel R., Hey A.J.G., Walker D.W.**, "A Proposal for a user-level, message passing interface in a distributed memory environment", *Technical Report TM-12231*, Oak Ridge National Laboratory (1993).

[Don95] **Dongara J.J., Otto S.W., Snir M., Walker D.**, "An Introduction to the MPI Standard", *Techical Report CS-95-274* (1995).

[Don97] **Dongara J., Dunigan T.**, "Message-Passing Performance of Various Computers", *Technical* Report, Oak Ridge National Laboratory (January 1997).
http://www.netlib.org/utk/papers/commperf.ps

[Dor96] **Dorigo M., Maniezzo V., Colorni A.**, "Ant System: Optimizing by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man and Cybernetics - Part B*, Vol. 26, No. 1, pp. 29-41 (February 1996).

[Dow96] **Dowsland K.A.**, "Genetic algorithms - a tool for OR?", *Journal of Operational Research Society,* Vol. 47, pp. 550-561 (1996).

[Dzu94] **Dzubera J.**, **Whitley D.**, "Advanced Corelation Analysis of Operators for the Traveling Salesman Problem", In: *Parallel Problem Solving from Nature - PPSN III*, eds: Davidor Y., Schwefel H.P., Manner R., Springer-Verlag, pp. 68-77 (1994).

[Erl78] **Erlenkotter D.**, "A dual-based procedure for uncapacitated facility location", *Operations Research*, Vol. 26, pp. 992-1009 (1978).

[Ers92] **Ersoy C.**, "Topological Design of Interconnected Local and Metropolitan Area Networks", *PhD Thesis*, Polytechnic University, New York (1992).

[Exp92] *Express User's Guide*, ver. 3.2.5, Parasoft Corporation, Monrovia, CA (1992). E-mail: *parasoft@Parasoft.com*

[Fan90] **Fang L., Li T.**, "Design of competition based neural networks for combinatorial optimization", *International Journal on Neural System,* Vol. 3, pp. 221-235 (1990).

[Fil96a] **Filipović V.**, **Kratica J.**, **Radojević S.**, **Vugdelija M.**, "Uticaj binarnog kodiranja na genetske algoritme za nalaženje ekstremnih vrednosti", *X Međunarodna konferencija Industrijski Sistemi - IS '96*, Novi Sad, Zbornik radova, str. 193-198 (1996).

[Fil96b] **Filipović V., Tošić D.**, "Comparasion of Selection Operators in Genetic Algorithms for the Function Optimization", In: *Abstratcts of XI Conference on Applied Mathematics - PRIM '96*, pp. 94, Budva (1996).

[Fil97] **Filipović V.**, "Određivanje performansi genetskih algoritama u teoriji i praksi", *Prolećna škola o programskim jezicima,* Institut za Matematiku PMF, Novi Sad, str. 131-141 (1997).

[Fil98] **Filipović V.** "Predlog poboljšanja operatora turnirske selekcije kod genetskih algoritama", *Magistarski rad*, Univerzitet u Beogradu, Matematički fakultet (1998).

[Fin88] **Finkelstein S., Schkolnick M., Tiberio P.**, "Physical Database Design for Relational Databases", *ACM Transactions on Database Systems*, Vol. 13, pp. 91-128 (1988).

[Fly66] **Flynn M.J.**, "Very High-Speed Computing Systems", *Proc. IEEE*, Vol. 54, pp. 1901-1909 (1966).

[Fly72] **Flynn M**., "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, Vol. 21, pp. 948-960 (1972).

[Fog89] **Fogarty T.C.**, "Varying the probability of mutation in the genetic algorithms", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 61-69 (1989).

[Fog91] **Fogarty T.C., Huang R.**, "Implementing the genetic algorithm on transputer based parallel processing systems", In: *Parallel Problem Solving from Nature - PPSN,* Schwefel H.P. and Manner R. (eds.), Springer-Verlag, Berlin, pp. 145-149 (1991).

[Fon93] **Fonseca C.M., Fleming P.J.**, "Genetic algorithms for multiobjective optimization: formulation, discusion and generalization", In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 416-423 (1993).

[Fra83] **Francis R.L., McGinnis L.F., White J.A.**, "Locational analysis", *European Journal of Operational Research*, Vol. 12, pp. 220-252 (1983).

[Fra92] **Francis R.L., McGinnis L.F., White J.A.**, "Facility Layout and Location: An Analytical Approach", Second Edition, Prentice-Hall International, Englewood Cliffs, NJ (1992).

[Fre84] **Freedman M.L., Tarjan R.E.**, "Fibonacci heaps and their uses in improved network optimization algorithms", In: *Proceedings of the 25th Annual IEEE Symposium on the Foundations of Computer Science,* pp. 338-346 (1984).

[Fry92] **Frye D., Bryant R., Ho H., Lawrence R., Snir M.**, "An external user interface for scalable parallel systems", *Techical Report*, IBM, May 1992.

[Gao94] **Gao L.L, Robinson E., Powell Jr.**, "Uncapacitated facility location: General solution procedure and computational experience", *European Journal of Operational Research*, Vol. 76, No. 3, pp. 410-427 (1994).

[Gar79] **Garey M.R.**, **Johnson D.S.**, "Computers and Intractibility: A Guide to the Theory of NP Completness", W.H. Freeman and Co. (1979).

[Gei94] **Geist A., Beguelin A., Dongarra J., Manchek R., Jaing W., Sundreem V.**, "PVM: A Users' Guide and Tutorial for Networked Parallel Computing", *MIT Press,* 1994.
http://netlib2.cs.utk.edu/pvm3/book/pvm-book.html

[Geo74] **Geoffrion A., McBride R.**, "Lagrangean Relaxation for Integer Programming", *Mathematical Programming Study*, Vol. 2, pp. 82-114 (1974).

[Gla90] **Galarce C.E.**, "Adaptive systems and the search for optimal index selection", *MSc Thesis*, Wayne State University, Detroit (1990).

[Gll88] **Gallo G., Pallotino S.**, "Fortran codes for network optimization: Shortest path algorithms", *Annals of Operations Research*, Vol. 13, pp. 3-79 (1988).

[Gln98] **Goldengorin B., Sierksma G., Tijssen G.A., Tso M.**, "The Data-Correcting Algorithm for Supermodular Functions, with Applications to Quadratic Cost Partition and Simple Plant Location Problems", *Technical Report 98A08*, University of Groningen, Netherland (February 1998).
http://www.ub.rug.nl/eldoc/som/a/98A08/98a08.pdf

[Glo86] **Glover F.**, "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research*, Vol. 5, pp. 533-549 (1986).

[Glo90] **Glover F.**, "Tabu search: A Tutorial", *Interfaces*, Vol 20, pp. 74-94 (1990).

[Glv93] **Galvao R.D.**, "The use of Lagrangean Relaxation in the solution of uncapacitated facility location problems", *Location* Science, Vol 1, No. 1, pp. 57-79 (1993).

[Gol85] **Goldberg D.E.**, "Alleles, loci and the TSP", In: *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pp. 154-159 (1985).

[Gol87] **Goldberg D.E.**, "Simple genetic algorithms and the minimal deceptive problem", In: *Genetic Algorithms and Simulated Annealing*, L. Davis (ed.), chapter 6, pp. 74-88 (1987).

[Gol89] **Goldberg D.E.**, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley Publ. Comp., Reading, Mass., 412 pp (1989).

[Gol90] **Goldberg D.E.**, "A Note on Boltzmann Tounament Selection for Genetic Algorithms and Population-oriented Simulated Annealing", *Complex Systems,* pp. 445-460 (1990).

[Gol91] **Goldberg D.E., Deb K.**, "A comparative analysis of selection schemes used in genetic algorithms", In: *Foundations of Genetic Algorithms*, Rawlins G. (ed.), Morgan Kaufmann, San Mateo, pp. 69-93 (1991).

[GolA93] **Goldberg A. V.,** "Scaling Algorithms for the Shortest Paths Problem", In: *Proceedings of the 4nd ACM-SIAM Symposium on Discrete Algorithms*, pp. 222-231 (1993).

[Grd93] **Gordon S., Whitley D.**, "Serial and parallel genetic algorithms as function optimizers", In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 177-183 (1993).
http://www.cs.colostate.edu/~ftppub/TechReports/1993/tr-114.ps.Z

[Grg89] **Gorges-Schleuter M.**, Asparagos: An Asynchronous Parallel Genetic Optimisation Strategy", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 416-421 (1989).

[Grg91] **Gorges-Schleuter M.**, "Explicit parallelism of genetic algorithms through population structures", In: *Parallel Problem Solving from Nature - PPSN ,* Schwefel H.P. and Manner R. (eds.), Springer-Verlag, Berlin, pp. 150-159 (1991).

[Gre84] **Grefenstette J.J.**, "Genesis: A system for Using Genetic Search Procedures", In: *Proceedings of the Conference on Intelligent Systems and Machines,* pp. 161-165 (1984).

[Gre85] **Grefenstette J.J.**, **Gopal R.**, **Rosmaita R.**, **Gucht D.V.**, "Genetic Algorithms for the traveling salesman problem", In: *Proceedings of an International Conference of Genetic Algorithms and Their Applications*, Texas Instrument and U.S. Navy Center for Applied Research and Artificial Intelligence, pp. 154-159 (1985).

[Gre86] **Grefenstette J.J.**, "Optimization of control parameters for genetic algorithms", *IEEE Transactions on Systems, Man and Cybernetics*, Vol 16, pp. 122-128 (1986).

[Gre87] **Grefenstette J.J.**, "Incorporating problem specific knowledge into genetic algorithms", In: *Genetic Algorithms and Simulated Annealing*, L.Davis, (ed.),chap. 4, pp. 42-60, Pitman (1987).

[Gre93] **Grefenstette J.J.**, "Deception considered harmful", In: *Foundations Of Genetic Algorithms - FOGA 2*, D. Whitley (ed.), pp. 75-91, Morgan Kaufmann (1993).

[Grf87] **Griffiths G., Carlyle Stones G.**, "The Tea-Leaf Reader Algorithm: An Efficient Implementation of CRC-16 and CRC-32", *Communications of the ACM*, Vol. 30, No. 7, pp. 617-620 (1987).

[Grn72] **Garfinkel R., Nemhauser G.**, "Integer Programming", John Wiley & Sons Inc., New York (1972).

[Gro94] **Gropp W., Lusk E., Skjellum A.**, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, Cambridge (1994).
http://www.mcs.anl.gov/mpi/usingmpi/index.html

[Gro96a] **Gropp W., Doss N.**, "Instalation Guide to ***mpich***, an Portable Implementation of MPI", *Technical Report ANL/MCS-TM-ANL-96/5*, Argonne National Laboratory, Department of Computer Science, University of Chicago, 45 p. (1996).
ftp://ftp.mcs.anl.gov/pub/mpi/install.ps.Z

[Gro96b] **Gropp W., Lusk E.**, "User's Guide for ***mpich***, a Portable Implementation of MPI", *Technical Report ANL/MCS-TM-ANL-96/6*, Argonne National Laboratory, Department of Computer Science, University of Chicago, 51 p. (1996).
ftp://ftp.mcs.anl.gov/pub/mpi/userguide.ps.Z

[Gro96c] **Gropp W., Lusk E., Doss N., Skjellum A.**, "A high-performance portable implementation of the MPI message-passing interface standard", *Parallel Computing*, Vol. 22, pp. 789-828 (1996).

http://www.mcs.anl.gov/mpi/mpich/

[Gro97] **Gropp W., Lusk E., Doss N., Skjellum A.**, "MPICH Model MPI Implementation: Reference Manual", *Technical Report ANL-00*, Argonne National Laboratory, Department of Computer Science, University of Chicago, 176 p. (1997).

[Grö95] **Grötschel M., Monma C.L., Stoer M.**, "Polyhedral and Computational Investigations for Designing Communication Networks with High Survivability Requirements", *Operations Research*, Vol. 43, No. 6, pp. 1012-1024 (1995).

[Grs94] **Grishukhin V.P.**, "On polynomial solvability conditions for the simplest plant location problem", In: *Selected topics in discrete mathematics*, Kelmans A.K. and Ivanov S. (eds), American Mathematical Society, Providence, RI, pp. 37-46 (1994)

[Gui88] **Guignard M.**, "A Lagrangean dual ascent algorithm for simple plant location problems", *European Journal of Operational* Research, Vol. 35, pp. 193-200 (1988).

[Han99] **Hansen P., Mladenović N.**, "An Introduction to Variable Neighborhood Search", In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Voss S., Martello S., Osman I.H., Roucairol C. (eds.), Kluwer Academic Publishers, pp. 433-458 (1999).

[Hat85] **Hatzopoulos M., Kollias J.G.**, "On the selection of a reduced set of indexes", *The Computer Journal*, Vol. 28, pp. 406-408 (1985).

[Hei93] **Heitkoetter J., Beasley D.**, "The Hitch-Hiker's Guide to Evolutionary computation", FAQ in *comp.ai.genetic ,* (1993). *ftp://rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/*

[Hel92] **Hellstrand J., Larsson T., Migdalas A.**, "A Characterization of the Uncapacitated Network Design Polytope", *Operations Research Letters*, Vol. 12, pp. 159-163 (1992).

[Hel94] **Helstrand J., Holmberg K.**, "A Lagrangean Heuristic Applied to the Uncapacitated Network Design Problem", *Technical Report LiTH-MAT/OPT-WP-1994-04*, Department of Mathematics, Linköping Institute of Technology, Sweden (1994).

[Her97] **Hertz A., Taillard E., de Werra D.**, "Tabu search", In: *Local Search in Combinatorial* Optimization, Aarts E.H.L. and Lenstra J.K. (eds.), John Wiley & Sons Ltd., pp. 121-136 (1997).

[Hil98] **Hill J., Warren M., Goda P.**, ”Loki”, *LINUX Journal*, pp. 56-60, January 1998.

[Hll75] **Holland J.H.**, "Adaptation in Natural and Artificial Systems", The University of Michigan Press, Ann Arbor (1975).

[Hlm86] **Holmberg K., Jörnsten K., Migdalas A.**, "Decomposition methods applied to discrete network design", *Technical Report LiTH-MAT/OPT-WP-1986-07*, Department of Mathematics, Linköping Institute of Technology, Sweden (1986).

[Hlm90] **Holmberg K.**, "On the Convergence of Cross Decomposition", *Mathematical Programming*, Vol. 47, pp. 269-296 (1990).

[Hlm91] **Holmberg K., Migdalas A.**, "Solution Methods for the Discrete Choice Network Design Problem Combining Lagrangean Relaxation and Decomposition with Generation of Valid Inequalities", *Technical Report LiTH-MAT/OPT-WP-1991-07*, Department of Mathematics, Linköping Institute of Technology, Sweden (1991).

[Hlm95] **Holmberg K.**, "Experiments with primal-dual decomposition and subgradient methods for the uncapacitated facility location problem", *Research Report LiTH-MAT/OPT-WP-1995-08*, Optimization Group, Department of Mathematics, Linkoping Institute of Technology, Sweden (1995).

http://www.mai.liu.se/~kahol/abstracts/Abstr-ULoc.txt

[Hlm97a] **Holmberg K., Ling J.,** "A Lagrangean Heuristic for the Facility Location Problem with Staircase Costs", *European Journal of Operational* Research, Vol. 97, pp. 63-74 (1997).
http://www.mai.liu.se/~kahol/papers/Rep-ScLoc.ps.gz

[Hlm97b] **Holmberg K., Hellstrand J.,** "Solving the uncapacitated network design problem by a Lagrangean heuristic and branch-and-bound", *Operations* Research (1997).
http://www.mai.liu.se/~kahol/papers/Rep-NDbabsub.ps.gz

[Hmf93] **Homaifar A.**, **Guan S.**, **Liepins G.E.**, "A New Approach on the Traveling Salesman Problem by Genetic Algorithms", In: *Proceedings of the Fifth International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, Calif., pp. 460-466 (1993).

[Hmf94] **Homaifar A., Lai S.H., Qi X.**, "Constrained optimization via genetic algorithms, *Simulation*, Vol. 62, pp. 242-254 (1994).

[Hoc91] **Hockney R.**, ”Performance parameters and benchmarking of supercomputers”, *Parallel Computing*, Vol. 17, pp. 1111-1130 (1991).

[Hoc94] **Hockney R., Berry M.**, ”Public international benchmarks for parallel computers”, parkbench committee report, *Scientific Programming*, Vol. 3, No. 2, pp. 101-146 (1994).

[Hof91] **Hoffmeister F.**, "The User's Guide to Escapade 1.2: A Runtime Environment for Evolution Strategies", Department of Computer Science, University of Dortmund, Germany (1991).

[Hou95] **Houck C.R., Joines J.A., Kay M.G.**, "Comparison of Genetic Algorithms, Random Restart, and Two-Opt Switching for Solving Large Location-Allocation Problems", *Techical Report*, Department of Industrial Engineering, North Carolina State University, Raleigh, NC (1995).
ftp://ftp.eos.ncsu.edu/pub/simul/GAOT/papers/gala.ps.gz

[Hug89] **Hughes M.**, "Genetic Algorithm Workbench Documentation", Cambridge Consultants, Cambridge, UK (1989).

[Ip83] **Ip M.Y.L., Saxton L.V., Raghavan V.V.**, "On the Selection of an Optimal Set of Indexes", *IEEE Transactions on Software Engineering*, Vol. 9, pp. 135-143 (1983).

[Jan91] **Janikow C.Z., Michalewicz Z.**, "An Experimental Comparasion of Binary and Floating Point Representations in Genetic Algorithms", In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, Morgan Kaufmann, San Mateo, Calif., pp. 37-44 (1991).

[Jog89] **Jog P.**, **Suh J.Y.**, **Van Gucht D.**, "The effects of Population size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 110-115 (1989).

[Jog90] **Jog P.**, **Suh J.Y.**, **Van Gucht D.**, "Parallel Genetic Algorithms Applied to the Traveling Salesman Problem", *Technical Report 314*, Indiana University (1990).

[Joh77] **Johnson D.S.**, "Efficient algorithms for shortest paths in sparse networks", *Journal of the ACM*, Vol. 24, No. 1, pp. 1-13 (1977).

[Joh78] **Johnson D.S., Lenstra J.K., Rinnooy Kan A.H.G.**, "The Complexity of the Network Design Problem", *Networks*, Vol. 8, pp. 279-285 (1978).

[Joh97] **Johnson D.S., McGeoch L.A.**, "The Traveling Salesman Problem: A Case Study in Local Optimization", *Local Search in Combinatorial* Optimization, eds. Aarts E.H.L., Lenstra J.K., John Wiley & Sons Ltd., pp. 215-310 (1997).

[Jon95] **Jones P.C., Lowe T.J., Muller G., Xu N., Yinyu Y., Zydiak J.L.**, "Specially Structured Uncapacitated Facility Location Problems", *Operations Research*, Vol. 43., No. 4, pp. 661-669 (1995).

[Jun98] **Jungnickel D.**, "Graphs, Networks and Algorithms", Springer Verlag, Berlin (1998).

[Kam92] **Kamel N., King R.**, "Intelligent Database Caching Through the Use of Page-Answers and Page-Traces", *ACM Transactions on Database Systems*, Vol. 17, No. 4, pp. 601-646 (December 1992).

[Khu94] **Khuri S.**, **Back T.**, **Heitkotter J.**, "An Evolutionary Approach to Combinatorial Optimization Problems", In: *Proceedings of CSC '94*, Phoenix, Arizona (1994).
ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/csc94.ps.gz

[Kid93] **Kido T., Kitano H., Nakanishi M.**, "A hybrid search for genetic algorithms: Combining genetic algorithms, tabu search, and simulated annealing", In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 614 (1993).

[Kir83] **Kirkpatrick S., Gellat C., Vecchi M.**, "Optimization by simulated annealing", *Science*, Vol. 220, pp. 671-680 (1983).

[Kno89] **Knox J.**, "The application of TABU search to the symmetric traveling salesman problem", PhD dissertation, University of Colorado (1989).

[Knu73] **Knuth D.E.**, "The Art of Computer Programming", second edition, Addison-Wesley, Reading, MA (1973).

[Koe89] **Koerkel M.**, "On the exact solution of large-scale simple plant location problems", *European Journal of Operational* Research, Vol. 39, pp. 157-173 (1989).

[Kon98] **Konchady M.**, "Parallel Computing Using LINUX", *LINUX Journal*, pp. 78-81, January 1998.

[Kor65] **Kornai J., Liptak T.**, "Two-level planning", *Econometrica*, Vol. 33, pp. 141-169 (1965).

[Koz93] **Koza J.R.**, "Genetic programming: On the programming of computers by means of natural selection", MIT Press, Cambridge, MA (1993).

[Kra94] **Kratica J.,** "Paralelizacija funkcionalnih programskih jezika i implementacija na transpjuterskim sistemima", *Magistarski rad*, Univerzitet u Beogradu, Matematički fakultet (1994).
http://alas.matf.bg.ac.yu/~kratica/msc.pdf

[Kra96a] **Kratica J.**, **Filipović V.**, **Šešum V.**, **Tošić D.**, "Solving of the uncapacitated warehouse location problem using a simple genetic algorithm", In: *Proceedings of the XIV International conference on material handling and warehousing,* Faculty of Mechanical Engineering, Belgrade, pp. 3.33 - 3.37 (1996).
http://alas.matf.bg.ac.yu/~kratica/cmhw96.pdf

[Kra96b] **Kratica J., Radojević S., Filipović V., Šćepanović A.**, "Primena epsilon-transformacije u problemu pretrage drveta", Međunarodni naučno-razvojni simpozijum: Stvaralaštvo kao uslov privrednog razvoja - Nove tehnologije i tehnike u službi čoveka, u štampi, Beograd (1996).
http://alas.matf.bg.ac.yu/~kratica/ntt96.pdf

[Kra97a] **Kratica J.**, "Napredne tehnike genetskih algoritama i njihova implementacija", *Prolećna škola o programskim jezicima,* Institut za Matematiku PMF, Novi Sad, str. 123-130 (1997).
http://alas.matf.bg.ac.yu/~kratica/pspj97.pdf

[Kra97b] **Kratica J.**, **Radojević S.**, **Šešum V.**, "Jedna metoda poboljšanja vremena izvršavanja prostog genetskog algoritma", *XXIII Jupiter konferencija,* Zbornik radova, Mašinski Fakultet, Beograd, str. 457 - 462 (1997).

[Kra97c] **Kratica J., Stojanović Z., Stojanović Ž.**, ”Primena genetskih algoritama za nalaženje optimalne trase cevovoda”, *Seminar iz genetskih algoritama*, Matematički fakultet, Beograd, 25.12.1997.

[Kra98a] **Kratica J., Ljubić I., Šešum V., Filipović V.**, "Neke metode za rešavanje problema trgovačkog putnika pomoću genetskih algoritama", *Zbornik radova sa drugog međunarodnog simpozijuma iz industrijskog inženjerstva,* Mašinski fakultet, Beograd, str. 281-284 (1998).

[Kra98b] **Kratica J., Filipović V., Tošić D.**, "Solving the Uncapacitated Warehouse Location Problem by SGA with Add-Heuristic", In: *Proceedings of the XV International conference on material handling and warehousing,* Faculty of Mechanical Engineering, Belgrade, pp. 2.28-2.32 (1998).

[Kra99a] **Kratica J.**, "Improvement of Simple Genetic Algorithm for Solving the Uncapacitated Warehouse Location Problem", In: *Advances in Soft Computing - Engineering Design and Manufacturing*, Roy R., Furuhashi T. and Chawdhry (eds.), Springer-Verlag London Ltd., pp. 390-402 (1999).

[Kra99b] **Kratica J.**, "Improving Performances of the Genetic Algorithm by Caching", *Computers and Artificial Intelligence*, Vol. 18, No. 3, pp. 271-283 (1999).

[Kra99c] **Kratica J.**, "Solving the simple plant location problem by genetic algorithms", Submitted to *Operations Research Spektrum* (1999).

[Krl94] **Karrels E., Lusk E.**, "Performance analysis of MPI programs", In: *Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing*, Dongarra J. and Tourancheau B. (eds.), SIAM Publications (1994).

[Krp72] **Karp R.**, "Reducibility among combinatorial problems", In: *Complexity of Computer Computations*, Miller R.E and Thatcher J.W. (eds.), Plenum Press, New York, NY, pp. 85-104 (1972).

[Krr83] **Krarup J.**, **Pruzan P.M.**, "The simple plant location problem: Survey and synthesis", *European Journal of Operational* Research, Vol. 12, pp. 36-81 (1983).

[Krr90] **Krarup J., Pruzan P.M.**, "Ingredients of Locational Analysis", In: *Discrete Location Theory*, Mirchandani P.B. and Francis R.L. (eds), John Wiley & Sons, chap. 3, pp. 120-171 (1990).

[Lam88] **Lam J.**, "An Efficient Simulated Annealing Schedule", PhD dissertation, Yale University (1988).

[Las91] **Laszevski G., Mühlenbien H.**, "Partitioning a graph with a parallel genetic algorithm", In: *Parallel Problem Solving from Nature - PPSN ,* Schwefel H.P. and Manner R. (eds.), Springer-Verlag, Berlin, pp. 165-169 (1991).

[Les93] **Lester B.**, ”The Art of Parallel Programming”, Prentice-Hall Inter., Englewood Cliffs, NJ (1993).

[Lew92] **Lewis T., El-Rewini H.**, "Introduction to Parallel Computing", Prentice-Hall Inter., Englewood Cliffs, NJ (1992).

[Lin82] **Lin S.**, "Effective use of heuristic algorithms in network design", *Proceedings of Symposia in Applied Mathematics*, Vol. 26, pp. 63-84 (1982).

[Lju98] **Ljubić I., Kratica J., Filipović V.**, "Primena genetskih algoritama u nalaženju minimalnog Steinerovog stabla", *Zbornik radova sa drugog međunarodnog simpozijuma iz industrijskog inženjerstva,* Mašinski fakultet, Beograd, str. 277-280 (1998).

[Lou91] **Louis S.J., Rawlins G.J.E.**, ”Designer genetic algorithms: Genetic algorithms in structure design”, In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91,* Morgan Kaufmann, San Mateo, Calif., pp. 53-60 (1991).

[Lov88] **Love R.F., Morris J.G., Wesolowshy G.O.**, *Facilities Location: Models & Methods*, North-Holland, New York (1988).

[Lve91] **Levenick J.**, ”Inserting introns improves genetic algorithms success rate: taking a cue from biology”, In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91,* Morgan Kaufmann, San Mateo, Calif., pp. 123-127 (1991).

[Lvi93a] **Levine D.**, "A Parallel Genetic Algorithm for the Set Partitioning Problem, *PhD thesis*, Argonne National Laboratory, ANL-94/23, Illinois Institute of Technology, (1993).
ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9423.ps.Z

[Lvi93b] **Levine D.**, "A genetic algorithm for the set partitioning problem", In: *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93,* Morgan Kaufmann, San Mateo, Calif., pp. 65-69 (1993).

[Lvi95a] **Levine D.**, "PGAPack Parallel Genetic Algorithm Library", Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL (1995).
http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Levine.PGAPack_Parallel_Genetic_Algorithm_Library.pgapack.tar.gz

[Lvi95b] **Levine D.**, "Users Guide to the PGAPack Parallel Genetic Algorithm Library", *Technical Report ANL-95/18*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, p. 77 (1995).
ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9518.ps.Z

[Lvi96] **Levine D.**, "A parallel genetic algorithm for the set partitioning problem", In: I.H. Osman and J.P. Kelly, editors, *Meta-Heuristic: Theory & Applications*, Kluwer Academic Publishers, pp. 23-35 (1996).
http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Levine.A_Parallel_Genetic_Algorithm_for_the_Set_Partitioning_Problem.ps.gz

[Mag84] **Magnanti T.L., Wong R.T.**, "Network Design and Transportation Planning: Models and algorithms", *Transportation Science*, Vol. 18, pp. 1-55 (1984).

[Mag86] **Magnanti T.L., Mireault P., Wong R.T.**, "Tailoring Benders Decomposition for Uncapacitated Network Design", *Mathematical Programming Study*, Vol. 29, pp. 464-484 (1986).

[Man91] **Manber U.**, "Introduction to Algorithms: A Creative Approach", Addison Wesley Publ. Comp., Reading, MA (1991).

[Mar96] **Markatos E.P.**, "Main Memory Caching of Web Documents", In: *Proceedings of the Fifth International World Wide Web Conference*, PS1 - Caching, Paris (May 1996).
http://www5conf.inria.fr/fich_html/papers/P1/Overview.html

[Mat91] **Matsumoto A.,** et. al. "Locally Parallel Cache Design Based on KL1 Memory Access Characteristics", *New Generation Computing*, Vol. 9, pp. 149-169 (1991).

[Mer94a] **Merelo J.J.**, "GAGS 0.94: User's Manual", *Technical Report*, Granada University, Electronic & Computer Technology Department, Spain (1994).
ftp://kal-el.ugr.es/pub/GAGS-0.92.tar.gz

[Mer94b] **Merelo J.J.**, "libGAGS 0.94: Programmer's Manual", *Technical Report*, Granada University, Electronic & Computer Technology Department, Spain (1994).
ftp://kal-el.ugr.es/pub/gagsprogs.ps.gz

[Met53] **Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E.**, "Equation of state calculation by fast computing machines", Journal of Chemical Physics, Vol. 21, pp. 1087-1091 (1953).

[Mey94] **Meyer J.**, ”Message-Passing Interface for Microsoft Windows 3.1”, *MSc thesis*, University of Nebraska, Faculty of the Graduate College, Department of Computer Science (December 1994).

[MGA92] MicroGA*, Genetic Algorithm Digest*, Vol. 6, No. 35, *Emergent Behavior,* Palo Alto, CA (1992).

[Mic91] **Michalewicz Z., Janikow C.**, "Handling constraints in genetic algorithms", In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91,* Morgan Kaufmann, San Mateo, Calif., pp. 151-157 (1991).

[Mic94] **Michalewicz Z., Attia N.**, "In evolutionary optimization of constrained problems", In: *Proceedings of the Third Annual Conference on Evolutionary Programming*, Sebald A., Fogel L. (eds), pp. 98-108, World Scientific Publishing (1994).

[Mic96] **Michalewicz Z.**, "Genetic Algorithms + Data Structures = Evolution Programs", Third Edition, Springer Verlag, Berlin Heideleberg (1996).

[Mig88] **Migdalas A.**, "Mathematical Programming Techniques for Analysis and Design of Communication and Transportation Networks", *PhD Thesis,* Department of Mathematics, Linköping University, Sweden (1988).

[Mir90] **Mirchandani P.B. and Francis R.L.**, "Discrete Location Theory", John Wiley & Sons (1990).

[Mit96] **Mitchell M.,** "An Introduction to Genetic Algorithms", MIT Press (1996).

[Moc89] **Mock J.**, "Processes, Channels, and Semaphores", *Transputer Toolset*, Inmos Corporation (1989).

[Mos89] **Moscato P.**, "On Genetic Crossover Operators for Relative Order Preservation", Caltech Concurent Computation Program, C3P Report 778 (1989).

[MPI94] **MPI Forum**, "The Message Passing Interface", *Interantional Journal of Supercomputing Applications,* Vol. 8, No. 3-4, pp. 165-414, Fall/Winter 1994.
http://www.mcs.anl.gov/mpi/mpi-report/mpi-report.html

[MPI95] **MPI Forum**, "MPI: A Message-Passing Interface Standard", *University of Tennessee*, Knoxville, Tennessee, p. 239 (1995).
http://www.netlib.org/mpi/mpi-1.1-report.ps

[MPI98a] "MPI/PRO"*, MPI Software Technology Inc*
http://www.mpi-softtech.com/products/mpi/mpipro/PDS-MPIProNT-Feb1998-1.html

[MPIa] "Free available and vendor-supplied MPI implementations"
http://www.mcs.anl.gov/mpi/implementations.html
http://www.osc.edu/mpi/implementations.html
http://www.lsc.nd.edu/MPI/

[MPIb] "Message Passing Libraries in MPI"
http://www.mcs.anl.gov/Projects/mpi/libraries.html

[MPICH96] "MPICH/NT 0.92 Beta", Mississippi State University, Engineering Research Center (May 1996).
http://www.erc.msstate.edu/mpi/mpiNT.html
ftp://aurora.cs.msstate.edu/pub/mpi/NTfiles/winMPICHpresent.ps
ftp://aurora.cs.msstate.edu/pub/mpi/NTfiles/winmpich092b.zip

[MPIFM98] "MPI-FM ver. 1.0", University of Illinois at Urbana-Champaign, Department of Computer Science, Concurrent Systems Architecture Group, USA (1998).
http://www-csag.cs.uiuc.edu/projects/comm/mpi-fm.html

[Mrc97] **Marcus K.**, "A new approach to the simple plant location problem" In: *Proceedings of International Conference on Industrial Engineering and Production Management - IEPM*, FUCAM Press, pp. 140-157, Lyon (France) (1997).
http://www.eurecom.fr/~marcus/Artigos-PS/iepm-lyon.ps.gz

[Müh91a] **Mühlenbein H.**, "Evolution in Time and Space - The Parallel Genetic Algorithm", In: *Foundations of Genetic Algorithms,* Rawlins G. (ed), Morgan Kaufmann, San Mateo, Calif., pp. 316-337 (1991).
ftp://borneo.gmd.de/pub/as/ga/gmd_as_ga-91_01.ps

[Müh91b] **Mühlenbein H., Schomisch M., Born J.**, "The Parallel Genetic Algorithm as Function Optimizer", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 271-278 (1991).

[Müh92] **Mühlenbein H.**, "Parallel Genetic ALgorithm in Combinatorial Optimization", In: *Computer Science and Operations Research*, Balci O., Sharda R., Zenios S. (eds), pp. 441-456, Pergamon Press (1992).
ftp://borneo.gmd.de/pub/as/ga/gmd_as_ga-92_01.ps

[Müh94] **Mühlenbein H., Schlierkamp-Voosen D.**, "The science of breeding and its application to the breeder genetic algorithm", *Evolutionary Computation*, Vol. 1, pp. 335-360 (1994).
ftp://borneo.gmd.de/pub/as/ga/gmd_as_ga-94_10.ps

[Müh95] **Mühlenbein H.**, "PeGaSuS: A Programming Environment for Genetic Algorithms", In: *Genetic Algorithms and Evolutionary Strategies in Computational Science & Engineering - EUROGEN'95*, Spain (1995).

[Müh97] **Mühlenbein H.**, "Genetic algorithms", *Local Search in Combinatorial* Optimization, eds. Aarts E.H.L., Lenstra J.K., John Wiley & Sons Ltd., pp. 137-172 (1997).

[Müh98] **Mühlenbein H.**, "The Equation for Response to Selection and its Use for Prediction", *Evolutionary Computation*, Vol. 5, pp. 303-346 (1998).
ftp://borneo.gmd.de/pub/as/ga/gmd_as_ga-98_01.ps

[Nak91] **Nakano R., Takeshi Y.**, "Conventional Genetic Algorithm for Job Shop Problems", In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91,* Morgan Kaufmann, San Mateo, Calif., pp. 474-479 (1991).

[Nel91] **Nelson M.**, "The Data Compression Book", M&T Books, Redwood City, CA, ISBN: 1-55851-214-4 (1991).

[Nem89] **Nemhauser G.L.**, **Wolsey L.A.**, "Integer and combinatorial optimization", John Wiley & Sons (1989).

[Nic96] **Nichols B., Buttlar D., Farrell J.P.**, ”Pthreads Programming”, O’Reilly & Associates, Sebastopol, California (1996).

[Nie92] **Nie X., Plaisted D.A.**, "A semantic backward chaining proof system", *Artificial Intelligence*, Vol. 55, pp. 109-128 (1992).

[Orv93] **Orvosh D., Davis L.**, "Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints", In: *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93,* Morgan Kaufmann, San Mateo, Calif., p. 650 (1993).

[Osm96a] **Osman I.H., Kelly J.P.**, "Metaheuristics: Theory and Applications", Kluwer Academic Publisher, Norwell (1996).

[Osm96b] **Osman I.H., Laporte G.**, "Metaheuristic: A bibliography", *Annals of Operations Research,* Vol. 63, pp. 513-623 (1996).

[Pal94] **Palmer C.C., Kershenbaum A.**, "Representing trees in genetic algorithms", In: *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, FL, June 21 - July 2, 1994.
ftp://ftp.Germany.EU.net/pub/research/softcomp/EC/GA/papers/trees94.ps.gz

[Pal95] **Palmer C.C.**, "An Approach To a Problem in Network Design Using Genetic Algorithms", *PhD Thesis*, Polytechnic University, New York (1995).
ftp://ftp.Germany.EU.net/pub/research/softcomp/EC/thesis/phd/palmer-phd.ps.gz

[Pap82] **Papadimitriou C.D., Steiglitz K.**, "Combinatorial Optimisation: Algorithms and Complexity", Prentice-Hall, Englewood Cliffs, NJ (1982).

[Pau97] **Paunić Đ.**, "Strukture podataka i algoritmi", Univerzitet u Novom Sadu, Prirodno-Matematički fakultet, Novi Sad (1997).

[Pem96] **Pemberton J.C., Zhang W.,** "Epsilon-transformation: exploiting phase transitions to solve combinatorial optimization problems", *Artificial Intelligence*, Vol. 81, pp. 297-325 (1996).

[Pit94] **Pitkow J.E., Recker M.M.**, "A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns", In: *Proceedings of the Second International World Wide Web Conference*, Mosaic and the Web, Chicago (October 1994).
http://www.vuw.ac.nz/~mimi/www/www-caching/caching.html

[Pla88] **Plaisted D.A.**, "Non-Horn Clause Logic Programming Without Contrapositives", *Journal of Automated Reasoning*, Vol. 4, pp. 287-325 (1988).

[Ree95] **Reeves C.R.**, "Modern Heuristic Techniques for Combinatorial Problems", *Advanced Topics in Computer Science Series,* McGraw-Hill Book Company, UK (1995).

[Rib94a] **Ribeiro-Filho J.L., Treleavean P.C., Alippi C.**, "Genetic-Algorithm Programming Environments", *IEEE Computer*, pp. 28-43, June 1994.
ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/ieee94.ps.gz

[Rib94b] **Ribeiro-Filho J.L.**, "Game system". In: *IEE Computing and Control Division Colloquium on Applications of Genetic Algorithms*, 94/067, pp. 2/1-2/4, London, March 1994.

[Ric89] **Richardson J.T., Palmer M.R., Liepins G., Hilliard M.**, "Some guidelines for genetic algorithms with penalty functions", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 191-197 (1989).

[Rob92] **Robbins G.**, "Engineer - The Evolution of Solutions", In: *Proceedings of the Fifth Annual Seminar on Neural Networks and Genetic ALgorithms*, IBC Technical Services Ltd., London, UK, pp. 218-232 (1992).

[Ryu92] **Ryu C., Guignard M.**, "An Exact Algorithm for the Simple Plant Location Problem with an Aggregate Capacity Constraint", *TIMS/ORSA Meeting*, Orlando, FL, 92-04-09 (1992).

[Sar88] **Sarwate D.V.**, "Computation of Cyclic Redundancy Checks via Table Look-Up", *Communications of the ACM*, Vol. 31, No. 8, pp.1008-1013 (1988).

[Sas97] **Sastry T.**, "Algorithms and Complete Formulations for the Network Design Problem", *Technical Report*, Indian Institute of Management, Ahmedabad (November 1997).
ftp://ftp.hpc.uh.edu/pub/ipco98/sastry.ps

[Sev98] **Sevenich R.A.**, "Parallel Processing using PVM", *LINUX Journal*, pp. 14-18, January 1998.

[Shf92] **Schaffer J.D., Whitley D., Eshelman L.J.**, "Combinations of genetic algorithms and neural networks: A survey of the state of the art", In: *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, Whitley D. and Schaffer J.D. (eds.), IEEE Press, Los Alamitos, CA (1992).

[Shm97] **Shmoys D.B., Tardos E., Aardal K.**, "Approximation algorithms for facility location problems", *Techical Report UU-CS-1997-39*, Department of Computer Science, Utrecht University, 21 p. (1997).

[Shn93] **Schoenauer M., Xanthakis S.**, "Constrained GA optimization", In: *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93,* Morgan Kaufmann, San Mateo, Calif., pp. 573-580 (1993).

[Sho93] **Shonkwiler R.**, "Parallel Genetic Algorithm", In: *Proceedings of the Fifth International Conference on Genetic Algorithms- ICGA '93*, Morgan Kaufmann, San Mateo, Calif., pp. 199-205 (1993).

[Shr91] **Schraudolph N.N., Grefenstette J.J.**, "A User's Guide to GAUCSD 1.2", Computer Science and Engineering Department, University of California, San Diego (1991).

[Shw95] **Schwefel H.P.**, "Evolution and Optimum Seeking", A Wiley-Interscience publication, John Wiley & Sons, New York, NY (1995).

[Sim89] **Simao H.P., Thizy J.M.**, "A dual simplex algorithm for the canonical representation of the uncapacitated facility location problem", *Operations Research Letters*, Vol. 8, No. 5, pp. 279-286 (1989).

[Skj90] **Skjellum A., Leung A.**, "Zipcode: A portable multicomputer communication library atop the reactive kernel", In: *Proceedings of the Fifth Distributed Memory Concurent Computing Conference",* Walker D.W. and Stout Q.F. (eds.), pp. 767-776, IEEE Press (1990).

[Skj92] **Skjellum A., Smith S., Still C., Leung A., Morari M.**, "The Zipcode message passing system", *Technical Report*, Lawrence Livermore National Laboratory, September 1992.
http://www.netlib.org/mpi/zipcode.uue

[SmA93] **Smith A.E., Tate D.M.**, "Genetic optimization using a penalty function", In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 499-505 (1993).

[SmR93] **Smith R., Forrest S., Perelson A.S.**, "Searching for diverse, cooperative populations with genetic algorithms", *Evolutionary Computation*, Vol. 1, No. 2, pp. 127-149 (1993).

[Sni95] **Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J.**, "MPI: The Complete Reference", *MIT Press*, MA, p. 350 (1995).
http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

[Spe91] **Spears W., De Jong K.**, "On the virtues of parametrized uniform crossover", In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 230-236 (1991).
ftp://ftp.aic.nrl.navy.mil/pub/papers/1991/AIC-91-022.ps.Z

[Sri94] **Srinivas M., Patnaik L.M.**, "Genetic Algorithms: A Survey", *IEEE Computer*, pp. 17-26 (June 1994).

[Sta91] **Starkweather T., Whitley D., Mathias K.**, "Optimization Using Distributed Genetic Algorithms", In: *Parallel Problem Solving from Nature - PPSN ,* Schwefel H.P. and Manner R. (eds.), Springer-Verlag, Berlin, pp. 176-185 (1991).

[Sys89] **Syswerda G.**, "Uniform Crossover in Genetic Algorithms", In: *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, Morgan Kaufmann, San Mateo, Calif., pp. 2-9 (1989).

[Sys91] **Syswerda G.**, "A study of reproduction in generational and steady-state genetic algorithms", *Foundations of Genetic Algorithms - FOGA*, Rawlins G.J. (ed.), Morgan Kaufmann Publishers, pp. 94-101 (1991).

[Šeš99a] **Šešum V., Kratica J.**, "Neke matematičke metode za rešavanje inverznog geofizičkog problema", Zbornik radova sa 25. Jupiter konferencije, Mašinski fakultet u Beogradu (Februar 1999).

[Šeš99b] **Šešum V.**, "Primena genetskih algoritama u rešavanju geofizičkog inverznog problema", *Magistarski rad*, Univerzitet u Beogradu, Matematički fakultet (1999).

[Šeš00] **Šešum V., Kratica J., Tošić D.**, "Solving of the Geophysical Inversion Problem by Genetic Algorithm", Submitted to *Yugoslav Journal of Operational Research* (2000).

[Tar83] **Tarjan R.E.**, "Data Structures and Network Algorithms", *SIAM*, Philadelphia, PA (1983).

[Tch87] **Tcha D.W., Myung P.**, "Dual-based Add Heuristic for Uncapacitated Facility Location", *IIE Transactions*, Vol. 19, No. 4 (1987).

[Tch88] **Tcha D.W., Ro H.B., Yoo C.B.**, "Dual-based Add Heuristic for Uncapacitated Facility Location", *Journal of Operational Research Society*, Vol. 39, No. 9, (1988).

[Tch95] **Tcha D.W., Paik C.H.**, "Parametric Uncapacitated Facility Location", *International Journal of Production Research*, Vol. 33, No. 3 (1995).

[Tch97] **Tcha D.W., Myung Y.S., Kim H.G.**, "A Bi-objective Model for the Uncapacitated Facility Location Problem", *European Journal of Operational Research,* Vol. 100, No.3, (1997).

[Tnb81] **Tanenbaum A.S.**, "Computer Networks", Prentice Hall, NJ, ISBN: 0-13-164699-0 (1981).

[Tnb92] **Tanenbaum A.S.**, "Modern Operating Systems", Prentice-Hall, NJ (1992).

[Tnl83] **Tansel B.C., Francis R.L., Lowe T.L.**, "Location on Networks: A Survey", *Management Science*, Vol. 29, pp. 482-511 (1983).

[Tns87] **Tanese R.**, "Parallel Genetic Algorithms for a Hypercube", In: *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, pp. 177-183 (1987).

[Tns89] **Tanese R.**, "Distributed Genetic Algorithms", In: *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, Calif., pp. 434-440 (1989).

[Tošˇ97] **Tošić D.**, "Pregled razvoja i opis osnovnih karakteristika evolucionih (genetskih) algoritama", *Prolećna škola o programskim jezicima,* Institut za Matematiku PMF, Novi Sad, str. 115-122 (1997).

[Tra89a] *Transputer C Library Description,* Inmos Corporation (1989).

[Tra89b] *Transputer Toolset,* Inmos Corporation (1989).

[Trm00] **Trmčić (Ljubić) I.**, "Primena genetskih algoritama na probleme povezanosti grafova", *Magistarski rad*, Univerzitet u Beogradu, Matematički fakultet (2000).

[Uck93] **Uckun S., Bagchi S., Kawamura K., Miyabe Y.**, "Managing Genetic Search in Job Shop Scheduling", *IEEE Expert*, pp. 15-24 (October 1993).

[Uro96] **Urošević D.**, "Algoritmi u programskom jeziku C", Mikro knjiga, Beograd (1996).

[Voi91] **Voigt H.M., Born J., Treptow J.**, "The Evolution Machine Manual - V 2.1", Institute for Informatics and Computing Techniques, Berlin (1991).

[VRo83] **Van Roy T. J.**, "Cross Decomposition for Mixed Integer Programming", *Mathematical Programming*, Vol. 25, pp. 46-63 (1983).

[Vse91a] **Vose M.D.**, "Modeling Simple Genetic Algorithms", In: *Foundations of Genetic Algorithms II - FOGA 2*, Whitley D. (ed.), Morgan Kaufmann Publishers, San Mateo, California (1992).

[Vse91b] **Vose M., Liepins G.**, "Schema disruption", In: *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, Morgan Kaufmann, San Mateo, Calif., pp. 237-242 (1991).

[Vss99] **Voss S., Martello S., Osman I.H., Roucairol C.**, "Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization", Kluwer Academic Publishers, Boston, MA (1999).

[Vug96] **Vugdelija M.**, **Kratica J.**, **Filipović V.**, **Radojević S.**, "Mogućnosti genetskih algoritama u mašinskom učenju", *XXII Jupiter konferencija,* Zbornik radova, Mašinski Fakultet, Beograd, str. 4.55 - 4.59 (1996). *http://alas.matf.bg.ac.yu/~kratica/jup96.pdf*

[Wat96] **Watson F.**, "Solving the Uncapacitated Facility Location Problem: A Comparison of Decomposition Methods", *Informs*, Washington, SD16.2 (1996).

[Whi88] **Whitley D., Kauth J.**, "Genitor: A Different Genetic Algorithm", In: *Proceedings of the Rocky Mountain Conference of Artificial Intelligence*, Denver, pp. 118-130 (1988).

[Whi89] **Whitley D.**, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", In: *Proceedings of the*

Third International Conference on Genetic Algorithms - ICGA '89, Morgan Kaufmann, San Mateo, Calif., pp. 116-123 (1989).

[Whi90] **Whitley D., Starkweather T.**, "Genitor-II: A Distributed Genetic Algorithm", *Journal of Experimental Theoretical Artificial Intelligence,* Vol. 2, pp. 189-214 (1990).

[Wlh92] **Wilhelms J., Van Gelder A.**, "Octrees for Faster Isosurface Generation", *ACM Transactions on* Graphics, Vol. 11, No. 3, pp. 201-227 (1992).

[Wlk98] **Wilkinson B., Allen M.**, ”Parallel Programming: Techniques and Application Using Networked Workstations and Parallel Computers”, Prentice-Hall, Upper Saddle River, NJ (1998).

[Wlm93] **Williams R.N.**, ”A Painless Guide To CRC Error Detection Algorithms” (1993). *ftp://ftp.adelaide.edu.au/pub/rocksoft/crc_v3.txt*

[Wls91] **Wilson S.W.**, "GA-Easy Does Not Imply Steepest-Ascent Optimizable", In: *Proceedings of the Fourth International Conference on Genetic Algorithms-ICGA '91*, Morgan Kaufmann, San Mateo, Calif., pp. 85-91 (1991).

[WMPI98a] ”WMPI 1.2 - The full implementation of MPI for Microsoft Win32 platforms”, Instituto Supererior de Engenharia de Coimbra, Portugal (November 1998). *http://dsg.dei.uc.pt/~fafe/w32mpi/*

[WMPI98b] ”PaTENT WMPI”, *WinPar European Project*, Instituto Supererior de Engenharia de Coimbra, Portugal *http://www.genias.de/genias_welcome.html*

[Wol95] **Wolpert D.H., Macready W.G.**, "No Free Lunch Theorems for Search", *Internal Report,* Santa Fe Institute (1995). *http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Wolpert.Macready.No_Free_Lunch_Theorems_for_Search.ps.gz*

[Ynn97] **Yannakakis M.**, "Computational Complexity", In: *Local Search in Combinatorial* Optimization, Aarts E.H.L. and Lenstra J.K. (eds.), John Wiley & Sons Ltd., pp. 19-56 (1997).

[You96] **Young-Soo M., Dong-Wan T.**, "Feasible Region Reduction Cuts for the Simple Plant Location Problem", *Journal of Operations Research Society of Japan*, Vol 39, No. 4, pp. 614-622 (1996).

[Yur94] **Yuret D.**, "From Genetic Algorithms to Efficient Optimization", *MSc Thesis*, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (1994). *http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Yuret.MSc_Thesis.From_Genetic_Algorithms_to_Efficient_Optimization.ps.gz*

[Zan89] **Zanakis S.H., Evans J.R., Vazacopoulos A.A.**, "Heuristic methods and applications: a categorized survey", *European Journal of Operational Research,* Vol. 43, pp. 88-110 (1989).

[Zha96] **Zhang W., Korf R.E.**, "A study of complexity transitions on the asymmetric traveling salesman problem", *Artificial Intelligence*, Vol. 81, pp. 223-239 (1996).

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

Spisak oznaka, skraćenica i stranih izraza korišćenih u radu

Oznake

Oznaka	Promenljiva u programu	Značenje
c_A	ga->fitness.a	Koeficijenti kod skaliranja prilagođenosti
c_B	ga->fitness.b	
c_C	ga->fitness.c	
f(x)	ga->fitness.f	Funkcija prilagođenosti
N_{bits}	CODEBITS	Broj bitova u genetskom kodu
N_{gener}	ga->finish.ngener	Maksimalan broj generacija GA
N_{elite}	ga->newgener.nelite	Broj elitnih jedinki u populaciji
N_{pass}	ga->select.npass	Broj jedinki koji direktno prolaze selekciju
N_{pop}	ga->nitem	Broj jedinki u populaciji
N_{rep}	ga->finish.nunchggen	Maksimalan broj generacija tokom kojih se najbolja jedinka nije promenila
p_{cross}	ga->cross.prob	Nivo ukrštanja
p_{mut}	ga->mut.prob	Nivo mutacije
p_{unif}	ga->cross.probunif	Verovatnoća ukrštanja gena kod uniformnog ukrštanja
x	ga->pop[...]->funvalue	Vrednost jedinke
x_{max}		Maksimalna vrednost jedinki u populaciji
x_{min}		Minimalna vrednost jedinki u populaciji
$\overline{x}$		Srednja vrednost jedinki u populaciji

Oznaka	Značenje
int()	Celobrojni deo izraza
frac()	Razlomljeni deo izraza

Skraćenice

skraćenica	strani izraz	srpski prevod
BnB	branch-and-bound	metoda grananja i ograničenja
BnC	branch-and-cut	metoda grananja i sečenja
CRC	cyclic redundancy code	ciklični redundansni kod
GA	genetic algorithm	genetski algoritam
GP	genetic programming	genetsko programiranje je metod za adaptivno generisanje programa
ISP	index selection problem	problem izbora indeksa
KL1	kernel language 1	jezik jezgra multiprocesorskog sistema
LRU	least recently used	tehnika najstarijeg korišćenog člana

MPI	message passing interface	standard za paralelizaciju prosleđivanjem poruka
NP	nondeterministic polynomial	nedeterministički polinomijalan
P	polynomial	polinomski
RB tree	red-black tree	RB stabla koja su pogodna za neke vrste pretraživanja
SAT problem	Boolean satisfiability problem	problem logičke zadovoljivosti
SCP	set covering problem	problem pokrivanja skupa
SGA	simple genetic algorithm	prost genetski algoritam
SPP	set partitioning problem	problem podele skupa
SPLP	simple plant location problem	prost lokacijski problem
TSP	traveling salesman problem	problem trgovačkog putnika
UNDP	uncapacitated network design problem	problem dizajniranja mreže neograničenog kapaciteta
XOR		isključiva disjunkcija
VNS	variable neighborhood search	metoda promenljivih okolina

Strani izrazi

strani izraz	**srpski prevod**
ant systems	mravlji sistemi
biconnectivity problem	problem 2-povezanosti
crossover	ukrštanje
one-point	jednopoziciono
two-point	dvopoziciono
multipoint	višepoziciono
uniform	uniformno
fitness	prilagođenost
fitness function	funkcija prilagođenosti
linear scaling	linearno skaliranje
direct scaling	direktno skaliranje
inverse scaling	inverzno skaliranje
sigma truncation	sigma odsecanje
fitness ranking	rangiranje
implicit fitness remaping	implicitna funkcija prilagođenosti
generalized assignment problem	uopšteni problem dodele
graph partition problem	problem podele grafa
K-partition problem	problem K-podele grafa
local search heuristics	heuristike lokalnog pretraživanja
maximum cut problem	problem maksimalnog presecanja
multiconstraint knapsack problem	problem ranca sa višestrukim ograničenjima
mutation	mutacija
minimum tardy task problem	problem nalaženja plana sa minimalnim zakašnjenima
optimal dependency trees	optimalna drveta zavisnosti
polyhedral techniques	poliedralne tehnike
propagation	prosleđivanje
real-time control problem	problem kontrole u realnom vremenu
search space	pretraživački prostor
selection	selekcija
simple roulette	prosta-rulet
rank based	zasnovana na rangu
tournament	turnirska
fine grade tournament	fino gradirana turnirska
stohastic reminder	primenom ostatka
simulated annealing	simulirano kaljenje
steady-state GA	stacionarni GA
subset sum problem	problem zbira podskupova
tabu search	tabu pretraživanje

INDEKS POJMOVA

—H—

—I—

—K—

—M—

—N—

—P—

—R—

—S—

—U—

—V—