Univerzitet u Beogradu
Matematički fakultet

Vladimir Filipović

OPERATORI SELEKCIJE I MIGRACIJE I WEB SERVISI KOD PARALELNIH EVOLUTIVNIH ALGORITAMA

doktorska disertacija

Beograd, 2006.

Komisija:

_____

prof. dr Dušan Tošić
(predsednik)

_____

prof. dr Milan Tuba
(član)

_____

dr Jozef Kratica
(član)

Datum odbrane

Univerzitet u Beogradu
Matematički fakultet

Rezime

OPERATORI SELEKCIJE I MIGRACIJE I WEB SERVISI KOD PARALELNIH EVOLUTIVNIH ALGORITAMA

Vladimir Filipović

U radu se razmatraju operatori selekcije i migracije kod paralelnih evolutivnih algoritama i predlaže implementacija paralelnih evolutivnih algoritama pomoću web servisa.
Selekcija je jedan od najvažnijih operatora kod evolutivnih algoritama. U ovom radu, posebna pažnja se poklanja operatorima turnirske selekcije i fino gradirane turnirske selekcije. Popularnost turnirske selekcije je porasla kada je utvrđeno da se ovaj operator veoma dobro uklapa u paralelne evolutivne algoritme. Fino gradirana turnirska selekcija predstavlja poboljšanje turnirske selekcije, uz zadržavanje svih dobrih osobina koje karakterišu turnirsku selekciju. U radu se predlažu razne varijacije fino gradirane turnirske selekcije. Za predložene varijacije operatora selekcije izvode se odgovarajuće teorijske procene. Potom se, u seriji testova, nove varijacije operatora porede međusobno i sa drugim selekcijama na De Jongovom skupu test-funkcija (kao i na nekim novijim test-funkcijama), kako bi se empirijski utvrdile performanse koje karakterišu ovakav algoritam (on-line performansa, off-line performansa, broj izgubljenih gena, selekcioni intenzitet, disperzija selekcije i gubitak raznovrsnosti). Pored toga, poređenje kvaliteta se vrši i na skupu instanci NP-teških problema, kao i na instancama problema vezanih za genetsko programiranje.
Operator migracije se javlja samo kod grubo usitnjenog paralelnog evolutivnog algoritma (odnosno ostrvskog evolutivnog algoritma, ili distribuiranog paralelnog evolutivnog algoritma). Kod grubo usitnjenog paralelnog evolutivnog algoritma svaki proces izvršava lokalni evolutivni algoritam na svojoj sopstvenoj potpopulaciji, uz povremenu razmenu jedinki između susednih potpopulacija. Međutim, kako je ovaj tip paralelnog evolutivnog algoritma najviše korišćen (i jedini koji se jednostavno i efikasno može implementirati u Internet okruženju), to je značaj migracije izuzetno veliki. U radu se daje teorijska procena osobina migracije i njenog doprinosa paralelnom evolutivnom algoritmu, te opisuju i analiziraju različite varijante migracije.
Detaljnije je opisan dizajn web servisa i aplikacija razvijenih radi izvršavanja paralelnih evolutivnih algoritama. Motivacija za razvoj web servisa je korišćenje računarskih resursa raspoloživih na Internetu. Pri izvršavanju, aplikacija šalje web servisu XML nisku sa informacijama o parametrima evolutivnog algoritma koji će biti izvršen i informacijama o instanci problema koji se rešava. Neki od parametara algoritma (kao što su zamena populacije, selekcija, kriterijum završetka itd.) ne zavise od reprezentacije jedinke, dok neki drugi (npr. tip populacije, inicijalizacija, ukrštanje, mutacija itd.) zavise. Web servis podržava tri tipa reprezentacije jedinki: binarna niska, broj u pokretnom zarezu i drvoidna reprezentacija. Web servis vraća (u XML formatu) nađeno rešenje i zahtevane dodatne rezultate potrebne za kreiranje raznovrsnih izveštaja. Web servis dopušta aplikaciji da prekine tekuće izvršavanje i vrati dotadašnji rezultat u ma kom momentu izvršavanja.
Ovaj servis i prateće aplikacije su razvijene pod .NET platformom, na programskom jeziku C#. I pored toga što je servis kompleksan, on je vrlo efikasan i može lako da se nadograđuje. Lakoća nadogradnje proizilazi iz činjenice da je tokom dizajna i razvoja dosledno korišćena objektno orjentisana paradigma i da su korišćeni obrasci dizajna.
Izvršena testiranja ukazuju na širinu polja primene razvijenog softvera. Web servis i prateće aplikacije se uspešno mogu primenjivati pri rešavanju raznorodnih problema – sve poznate evolutivno inspirisane tehnike se veoma jednostavno uklapaju u razvijen programski model. Rezultati izvršavanja na instancama izabranih problema pokazuju superiornost operatora fino gradirane turnirske selekcije nad alternativama i daju jasnu preporuku za tip i veličinu parametara migracije.

Ključne reči: evolutivni algoritmi, genetski algoritmi, paralelni algoritmi, operator selekcije, operator migracije, web servis, fino gradirana turnirska selekcija.

Belgrade University
Faculty of Mathematics

Abstract

SELECTION AND MIGRATION OPERATORS AND WEB SERVICES IN PARALLEL EVOLUTIONARY ALGORITHMS

Vladimir Filipović

In this paper selection and migration operators in parallel evolutionary algorithms are studied. Also, a parallel evolutionary algorithms implementation using web services is studied.
Selection is one of the most important operators in evolutionary algorithms. In paper special attention is dedicated to the tournament and fine grained tournament selection operators. Popularity of tournament selection grows because that operator extremely well fits to parallel evolutionary algorithms. Fine grained tournament selection is an improvement of tournament selection, which keeps all good features of tournament selection. Several different variations of tournament selection are proposed and adequate theoretical estimations are made. After that, the series of tests are created for comparison among these new variations and other selection operators. Operators are practically compared on De Jong test function suite (and on some newer test functions) and algorithm performances (on-line performance, off-line performance, number of lost genes, selection intensity, dispersion of selection and loss of diversity) are computed. Quality of operators is also empirically compared by solving NP-hard problems and by solving problem that is typical for genetic programming.
Migration operator exists only within coarse grained parallel evolutionary algorithm (also known as island evolutionary algorithm, or distributed parallel evolutionary algorithm). In coarse grained parallel evolutionary algorithm every process executes local evolutionary algorithm on its own subpopulation and from time to time neighbor subpopulations exchange several individuals. However, this type of parallel evolutionary algorithm is the most often used (only that type of computation can be easily and efficiently implemented in Internet environment). Therefore, migration is extremely important evolutionary operator. In this paper, theoretical estimations for migration properties and for its influence to parallel evolutionary algorithm are made. Also, several versions of migration operator are described and analyzed.
Design of web service and applications developed for parallel evolutionary algorithms is described carefully and with many details. Developing of web service is motivated by utilizing computational resources available on Internet. During execution, application sends to web service XML string. XML string contains parameters of evolutionary algorithm that should be executed and information about problem instance. Some of algorithm parameters (like population replacement, selection, finishing criterion etc.) don't depend on individual's representation, while some others (population type, initialization, crossover, mutation, etc.) depend. Web service supports three types of individual's representation: binary string, floating-point number and tree-like representation. Web service returns (in XML format) a solution that is found and additional results, used in various reports. Web service allows application to stop ongoing execution and retrieve current result.
This service and applications are developed under .NET platform, in C# programming language. Although web service is very complex, it is very efficient and can be easily expanded. Easiness of expansion comes from the fact that design and development strictly use object-oriented paradigm and design patterns.
The software that is developed can be used in very wide manner. Web service and accomplishing applications can be successfully used on various problems– all known evolutionary inspired techniques can be extremely easily mapped programming model that is developed. Testing results shows that fine grained tournament selection operator is superior to alternatives. They also give recommendations adequate selection of migration type and migration parameters.

Keywords: evolutionary algorithms, genetic algorithms, parallel algorithms, selection operator, migration operator, web service, fine grained tournament selection.

SADRŽAJ

ZAHVALNICA

Autor želi da se zahvali mentoru, prof. dr Dušanu Tošiću na izuzetnom strpljenju, veoma korisnim sugestijama i stručnim savetima.
Posebnu zahvalnost autor duguje i članovima komisije prof. dr Milanu Tubi na vrlo pažljivom čitanju rukopisa i korisnim savetima, kao i dr Jozefu Kratici na vremenu i energiji koju je potrošio u razmatranju ideja, u sugerisanju puteva za .prevazilaženje problema koji su se pojavljivali u tokom rada na disertaciji.
Kolegama mr Zorici Stanimirović, dr Ivani Ljubić i Milanu Vugdeliji se zahvaljujem na korisnim savetima i sugestijama u raznim periodima našeg zajedničkog rada na evolutivnim algoritmima.
Zahvaljujem se na podršci, pomoći, razumevanju i savetima koju sam dobijao od kolega sa Katedre za Računarstvo Matematičkog fakulteta u Beogradu.
Isto tako, zahvaljujem se na pomoći i kolegama sa Projekta 1583 "Matematički modeli i metode optimizacije sa primenama".
Veliko hvala i kolegama iz Računarske laboratorije Matematičkog fakulteta – mnogo su mi pomogli u dosadašnjem radu.
Svetlana Ćupić je, iako jako zauzeta, uložila mnogo truda i vremena oko lekture za neke od radova koji su objavljeni u inostranim časopisima i ona ima moju veliku zahvalnost.
Hteo bih da iskoristim ovu priliku da se zahvalim Danilu Šćepanoviću, mom profesoru matematike u podgoričkoj gimnaziji. On je, po mom mišljenju, najzaslužniji za odluku da se uputim stazama Matematike i Računarstva – odluku koja mi je donela mnogo sreće i zadovoljstva u životu.
Naročito se zahvaljujem svojoj porodici - supruzi Luciji, roditeljima Jovanu i Milevi i braći Nebojši i Slobodanu što su mi davali bezuslovnu podršku i pomoć u najtežim trenucima, mada je to često od njih zahtevalo dodatna odricanja i žrtve.
Moja ćerkica Sofija mi nije pomogla u radu na disertaciji, ali mi je u protekle dve godine samim svojim postojanjem darivala neslućenu radost, koja blagotvorno utiče na sve aspekte mog života, pa samim tim i na ovaj rad.

TABELA SIMBOLA

n	-	Broj jedinki u populaciji
a	-	Niska
l	-	Dužina niske
H	-	Shema
$o(H)$	-	Red sheme
$\delta(H)$	-	Definišuća dužina sheme
$m(H,t)$	-	Broj pojava sheme H u zadatoj populaciji u trenutku t
f	-	Funkcija prilagođenosti jedinke
$f(H)$	-	Prosek prilagođenosti niski populacije koje se sadrže u shemi H
$\overline{f}$	-	Srednja vrednost prilagođenosti svih niski u populaciji
$\overline{f}$	-	Vrednost prilagođenosti po izvršenju selekcije
p_c	-	Verovatnoća ukrštanja
p_m	-	Verovatnoća mutacije
p_s	-	Verovatnoća da jedinka preživi selekciju
n_s	-	Broj shema koje obrađuje populacija
p_{surv}	-	Verovatnoća preživljavanja za konkretnu shemu H
$s(f)$	-	Funkcija raspodele prilagođenosti
$\bar{s}(f)$	-	Funkcija raspodele prilagođenosti (neprekidna)
$S(f_k)$	-	Kumulativna raspodela prilagođenosti
$\bar{S}(f)$	-	Kumulativna raspodela prilagođenosti (neprekidna)
Ω	-	Metod selekcije
Ω^*	-	Očekivana raspodela prilagođenosti po primeni metoda selekcije
σ_s^2	-	Disperzija prilagođenosti
$\overline{M}$	-	Očekivana prosečna prilagođenost populacije pre selekcije
$\overline{M^*}$	-	Očekivana prosečna prilagođenost populacije posle selekcije
$\left(\overline{\sigma}\right)^2$	-	Disperzija raspodele prilagođenosti pre selekcije
$\left(\overline{\sigma^*}\right)^2$	-	Disperzija raspodele prilagođenosti posle selekcije
$\overline{R(f)}$	-	Stopa reprodukcije datog selekcionog metoda
p_d	-	Gubitak raznovrsnosti datog selekcionog metoda

1.UVOD

Glavna tema ovog rada su operatori selekcije i migracije kod paralelnih evolutivnih algoritama, kao i implementacije evolutivnih algoritama.
Centralni deo rada je posvećen proučavanju operatora selekcije, preciznije turnirske selekcije i fino gradirane turnirske selekcije. Fino gradirana turnirska selekcija, kao što ćemo videti, predstavlja poboljšanje turnirske selekcije, uz zadržavanje svih dobrih osobina koje karakterišu turnirsku selekciju. U radu se ispituju razne varijacije fino gradirane turnirske selekcije i za predložene varijacije operatora selekcije izvode odgovarajuće teorijske procene. Potom se, u seriji testova, nove varijacije operatora porede međusobno i sa drugim selekcijama na De Jongovom skupu test-funkcija, na nekim novijim test-funkcijama, na setu instanci NP-teških problema i instancama problema iz domena tipičnog za genetsko programiranje.
Operator migracije se javlja samo kod grubo usitnjenih paralelnih evolutivnih algoritama (odnosno ostrvskih evolutivnih algoritama, ili distribuiranih paralelnih evolutivnih algoritama - gde svaki proces izvršava lokalni evolutivni algoritam na svojoj sopstvenoj potpopulaciji, uz povremenu razmenu jedinki između susednih potpopulacija). Međutim, kako je ovaj tip paralelnog evolutivnog algoritma najviše korišćen (i za sada jedini koji se jednostavno i efikasno može implementirati u Internet okruženju), to je značaj migracije veliki. U radu se daje teorijska procena osobina migracije i njenog doprinosa paralelnom evolutivnom algoritmu, te opisuju i analiziraju različite varijante migracije.
Detaljnije su opisane implementacije koje je razvijao autor i na kojima su u proteklom periodu vršeni eksperimenti iz oblasti Evolutivnih algoritama. Posebno je podrobno opisan dizajn web servisa i aplikacija razvijenih radi izvršavanja paralelnih evolutivnih algoritama. Motivacija za razvoj web servisa je korišćenje računarskih resursa raspoloživih na Internetu. Pri izvršavanju, aplikacija šalje web servisu XML nisku sa informacijama o parametrima evolutivnog algoritma koji će biti izvršen i informacijama o instanci problema koja se rešava. Neki od parametara algoritma (kao što su zamena populacije, selekcija, kriterijum završetka itd.) ne zavise od reprezentacije jedinke, dok neki drugi (npr. tip populacije, inicijalizacija, ukrštanje, mutacija itd.) zavise. Web servis podržava tri tipa reprezentacije jedinki: binarna niska, broj u pokretnom zarezu i drvoidna reprezentacija. Web servis vraća (u XML formatu) nađeno rešenje i zahtevane dodatne rezultate potrebne za kreiranje raznovrsnih izveštaja. Web servis dopušta aplikaciji da prekine tekuće izvršavanje i vrati dotadašnji rezultat u ma kom momentu izvršavanja.
Ovaj servis i prateće aplikacije su razvijene pod .NET platformom, na programskom jeziku C#. Servis je kompleksan, ali i vrlo efikasan i može lako da se nadograđuje. Lakoća nadogradnje proizilazi iz činjenice da je tokom dizajna i razvoja dosledno korišćena objektno orjentisana paradigma i obrasci dizajna.
Izvršena testiranja ukazuju na širinu polja primene razvijenog softvera. Web servis i prateće aplikacije se uspešno mogu primenjivati pri rešavanju krajnje raznorodnih problema – sve poznate evolutivno inspirisane tehnike se krajnje jednostavno preslikavaju u razvijen programski model. Rezultati izvršavanja na instancama izabranih problema pokazuju superiornost operatora fino gradirane turnirske selekcije nad alternativama i daju jasnu preporuku za tip i veličinu parametara migracije.
Rad sadrži šest poglavlja. U prvom poglavlju se opisuju evolutivni algoritmi (istorijat, oblasti primene, struktura, osobine i modifikacije), paralelni evolutivni algoritmi i postojeći softverski sistemi za implementaciju evolutivnih algoritama. Drugo poglavlje sadrži nekoliko metodologija teorijske analize evolutivnih algoritama (sekvencijalnih i paralelnih) i pojedinih njegovih aspekata.. U trećem poglavlju su detaljno opisani najvažniji operatori selekcije i migracije, te predložene nova poboljšanja ovih operatora. Operatori su analizirani u svetlu prethodno definisanih metodologija. Četvrto poglavlje opisuje EA sisteme koje je razvijao autor, kako one dominantno posvećene rešavanju NP teških problema, tako i novorazvijeni EA web servis i aplikacije koje ga koriste Peto poglavlje sadrži opis metodologije empirijskog poređenja raznih evolutivnih algoritama. U njemu su izloženi rezultati poređenja korišćenjem konkretnih, realnih problema. Poslednje, šesto poglavlje sadrži zaključak. Literatura i indeks su izdvojeni u posebne celine.

1.1. Evolutivni algoritmi

1.1.1. Istorijat i značaj evolutivnih algoritama.

Evolutivni algoritmi (eng. evolutionary algorithms), u daljem tekstu označeni sa EA su zasnovane na Darvinovoj teoriji evolucije. Darvinova knjiga "Postanak vrsta putem prirodnog odabiranja" (videti [Darvin85]), napisana u drugoj polovini XIX veka (dakle mnogo pre nastanka računara) je teorijska osnova za ovu klasu algoritama. Zakoni nasleđivanja Georga Mendela predstavljaju, ne samo osnovu genetike, već i drugu čvornu tačku na koju se oslanjaju evolutivni algoritmi.

Postoji više tehnika koje pri rešavanju problema koriste gore opisan pristup. Pored genetskih algoritama, koji su najčešće zastupljeni, u Nemačkoj i Engleskoj su razvijeni i drugi, dosta slični, pristupi za rešavanje različitih problema - evolutivno programiranje i evolutivne strategije. Potom se, pojavilo i genetsko programiranje. U najšire prihvaćenoj klasifikaciji ovakvih tehnika Evolutivni algoritmi su pojam koji natkriljuje i obuhvata sve prethodno pobrojane pristupe (videti [Heitko93], [Goldbe94]). Mnogi autori kao sinonim za Evolutivne algoritme koriste termin Evolutivno izračunavanje (eng. Evolutionary Computation - EC).

Mada su pre 70.-tih godina prošlog veka, naročito u Nemačkoj i Engleskoj, predlagana rešenja raznovrsnih problema koja emuliraju ponašanje, odnose i veze kao u prirodi, široka primena evolutivne paradigme u raznim domenima se vezuje za 1975. godinu i Hollandovu knjigu "Adaptation in natural and artificial systems" ([Hollan75]). Holland je proučavao EA radi praćenja procesa prilagođavanja kod prirodnih sistema i radi razvoja sistema veštačke inteligencije koji oponašaju modele prilagođavanja.

Činjenica je da EA u poslednje vreme doživljavaju sve veću popularnost. Evolutivna paradigma koju eksploatišu EA postaje široko prihvaćena kao put za rešavanje određenih klasa problema. Danas postoji ogroman broj radova i mnogo knjiga koje se bave genetskim algoritmima. Postoje i časopisi specijalizovani za EA (IEEE on Evolutionary Computation, Evolutionary Computation, BioSystems, Adaptive Behavior, itd.), a članci vezani za ovu problematiku se javljaju i u časopisima posvećenim Veštačkoj inteligenciji (Complex Systems, Artificial Intelligence, Artificial Life, Machine Learning, IEEE Transactions on Systems Man and Cybernetics, IEEE Expert, itd.), Paralelnom procesiranju, Operacionim istraživanjima, itd. Redovno se organizuju međunarodne konferencije posvećene EA (CEC – Congress of Evolutionary Computation, GECCO – Genetic and Evolutionary Computation, PPSN - Parallel Problem Solving from Nature, Alife - International Conference on Artificial Life, EvoCOP – European Conference on Evolutionary Computation in Combinatorial Optimization, EvoGP - European Conference on Genetic Programming, EvoComNet - European Workshop on Evolutionary Computation in Communications, Networks, and Connected System itd.) sa kojih se objavljuju zbornici radova.

Među istraživačima u oblasti računarskih nauka je uobičajena klasifikacija po kojoj se EA, kao i druge slične tehnike, zajedno sa Fazi logikama i Neuralnim mrežama, smeštaju u tzv. Soft computing.

Veliki interes koji su EA pobudili u akademskim krugovima može se kvantifikovati i brojem javno dostupnih programskih paketa koje su razvili istraživači u ovoj oblasti. Tih paketa ima više desetina (opis u [Heitko93], [Filho94], [Whitle89], [Bäck92b], [Cantu94]), a od njih su najpoznatiji i najpopularniji: DGenesis (Erick Cantu-Paz); GAL (William Spears); GAlib (Matthew Wall); Gaucsd (Nici Schraudolph); GA Workbench (Mark Hughes); Genesis (John Grefenstette); GENEsYs (Thomas Bäck); GenET (Cezary Janikow), itd. Ovo poglavlje sadrži i klasifikaciju i opis najvažnijih među pobrojanim programima.

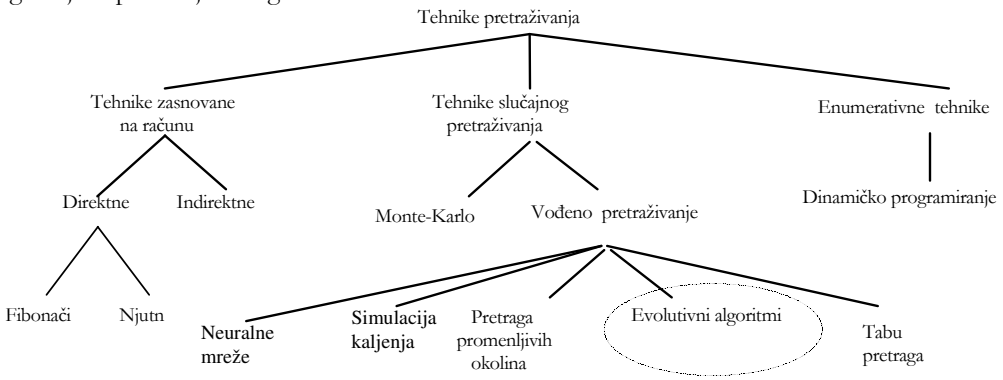
Do skora su EA bili predmet izučavanja isključivo u akademskim krugovima. Danas postoji veliki broj programskih paketa zasnovanih na EA, koji nalaze primenu u praksi. Tu spadaju (videti [Heitko93], [ManEAn93], [Filho94]): EnGENEer (Logica Cambridge Ltd.); MicroGA, EVOFrame i Galapagos (Optimum Software); Evolver (Ax celis Inc.), itd. Prethodno pobrojani paketi predstavljaju alate za lakše i brže kreiranje EA aplikacija. Broj praktičnih aplikacija (nastalih bilo uz pomoć ovih alata, bilo bez njih) koje eksploatišu evolutivnu paradigmu je još mnogo veći, i svakim danom sve više raste.

1.1.2. Oblasti primene evolutivnih algoritama

EA se primenjuju za rešavanje raznih optimizacionih problema (videti [DeJong89], [Goldbe93], [Schwef95], [DeJong95a]), kao što su: problem kreiranja rasporeda (videti [Burke95]), problem optimizacije parametara (videti [Wienho93]), problem trgovačkog putnika (videti [Goldbe89], [Kratic98]), problem pakovanja (videti npr. [Khuri95]), problem izbora lokacija za skladišta bez obzira na kapacitet (videti [Kratic96]), problem

dizajniranja mreže bez obzira na kapacitet (u radu [Kratic02]), problem uspostavljanja habova u transportnoj mreži (videti npr. [Kratic05], [Kratic06]), raspoređivanje poslova na višeprocesorskom sistemu (videti [Hou94]), problem određivanja minimalnog Štajnerovog drveta u grafu (videti [Ljubić98], [Ljubić04] i [Ljubić06]), problem dopune do dvostruko i do dvostrano povezanih grafova (videti [Ljubić00] , [Ljubić00a], [Ljubić00b], [Ljubić01], [Raidl02], [Ljubić03] i [Ljubić04]), problemi zadovoljivosti SAT i PSAT (videti [Spears95], [Ognjan01], [Ognjan04]) itd. Obuhvatan pregled primena EA u operacionim istraživanjima dat je u radu [Alande95].

Problem pretraživanja prostora u potrazi za optimalnim rešenjem se proučava od davnina. Razvijene su raznovrsne tehnike pretraživanja u potrazi za optimalnim rešenjem (npr. Njutnova metoda, Metoda gradijentnog spusta, Metoda Monte-Karlo itd.). Svaka od metoda ima određene prednosti i mane. Tako Njutnova metoda brzo konvergira, ali zahteva glatkost funkcije i može se primeniti samo za probleme globalne optimizacije (funkcije realnih argumenata). Enumerativne tehnike nemaju dodatnih uslova za konvergenciju, ali u slučaju kada je prostor pretrage suviše veliki, a najčešće je baš taj slučaj interesantan, pretraga traje neprihvatljivo dugo.



slika 1.1 Klasifikacija tehnika pretraživanja

Tehnike pretraživanja zasnovane na računu direktno eksploatišu neke od dodatnih osobina funkcija čiji se ekstremum traži. Enumerativne tehnike sukcesivno prolaze kroz diskretizovan prostor pretraživanja, kako bi se našao ekstremum. Tehnike slučajnog pretraživanja koriste slučajne brojeve, tj. slučajne izbore, u procesu traženja optimalnog rešenja. Dok Monte-Karlo tehnike nisu u mogućnosti da eksploatišu već dobijene rezultate, tehnike vođenog pretraživanja eksploatišu te rezultate time što nove aproksimacije zavise od rezultata koji su postignuti u prethodnim izborima.

Rezultati koje su EA pokazali pri rešavanju optimizacionih problema su doveli do korišćenja EA za rešavanje velikog broja konkretnih realnih problema: aproksimacija funkcija ([Rodger95]), uklapanje splajn funkcija u zadate podatke ([Manela93]), numerička optimizacija ([Michal95]), obezbeđivanje optimalnog utroška energije u nekom procesu proizvodnje ([Goldbe89]), optimizacija troškova transporta ([Goldbe89]), dizajniranje integrisanih kola ([SanMar93] i [Stanle95]), dizajniranje fazi kontrolera ([Pavlic96]), optimizacija upita u bazama podataka ([Yang93], [Kratic03]), generisanje pseudo slučajnih brojeva (u radu [Koza91]), projektovanje najoptimalnije trase za cevovod ([Goldbe89], [Kratic97a]), održavanje gasovoda ([Goldbe89]), kompresija podataka ([Ng95]), balansiranje ulaza na presama za šećer ([Fogart95]), optimalno isecanje staklenih površina (rad [Puch04]) itd. Potpunije informacije o realnim problemima koji su sa uspehom rešavani pomoću EA mogu da se nađu u knjigama [Chamb95], [Chamb99], [Chamb01] i [Karr99].

EA se koriste i u mnogim oblastima veštačke inteligencije (videti [Nillso98]): mašinskom učenju ([Goldbe89], [Kelly92], [Neri95a], [Vugdel96], [Grefen96]), teoriji igara ([Goldbe89]), kao dokazivači teorema ([DeJong89], [Spears95]), kod neuronskih mreža ([Janson93], [Spears95]), za finansijsko modeliranje ([Bauer84]), za dizajniranje kontrolera robota koji bi omogućio njegovo autonomno kretanje po morskom dnu, za prepoznavanje likova sa foto-robot crteža ([Goldbe89]), itd.

U literaturi su zabeleženi pokušaji korišćenja EA u crtanju i u komponovanju, kao i uspešna primena za rešavanje nekih problema iz oblasti virtualne realnosti (videti [Jacq95]).

Potpuniji pregled evolutivnih algoritama je dat u [Bäck00] i [Bäck00a].

1.1.3. Osobine evolutivnih algoritama

Ideja EA zasniva se na oponašanju prirodnih procesa evolucije. Proces evolucije se ovde razmatra kao razvoj složenih organizama (npr. biljke i životinje) iz prostijih formi. EA je uprošćeni model suštine procesa evolucije i nasleđivanja.

Rast biljaka i životinja je primarno kontrolisan genima koji su nasleđeni od roditelja. Geni su smešteni u jednoj ili više traka hromozoma. U aseksualnoj reprodukciji, hromozom organizma je kopija roditeljevog, uz moguće slučajne izmene, poznate kao mutacije. U seksualnoj reprodukciji, nova jedinka nasleđuje osobine oba roditelja. Najčešće se oko polovine genetskog materijala kopira od jednog roditelja i to se spoji sa genetskim materijalom koji se nalazi u hromozomima drugog roditelja. Na taj način, genetski kod potomka se razlikuje i od jednog i od drugog roditelja.

Evolucija deluje tako što najbolje prilagođene jedinke opstaju, reprodukuju se i prenose svoj genetski materijal u sledeće generacije. Kako se genetski materijal u populaciji menja, tako se vrsta kao celina menja, tj. evoluira kao rezultat selekcionog opstanka jedinki od kojih je sastavljena.

EA sadrži populaciju probnih rešenja problema, pri čemu je obično svako rešenje (tj. svaka jedinka) u populaciji modelirana niskom koja predstavlja njen genetski kod. Ova populacija „evoluira“ uzastopnim izborima „boljih“ rešenja i proizvođenjem novih rešenja na osnovu njih (borba za opstanak). Novoformirana rešenja zamenjuju postojeća rešenja u populaciji. Ta nova rešenja, odnosno nove jedinke se formiraju bilo aseksualno, bilo seksualno.

Možemo izdvojiti sledeće osobine po kojima se EA razlikuju od drugih metoda za rešavanje problema:

- ne koriste se sami parametri u izvornom obliku, već se vrši kodiranje parametara;
- ne pretražuje se jedan čvor prostora pretraživanja, već populacija čvorova, što ukazuje na robustnost algoritma (ovo je najčešći slučaj, mada ima i izuzetaka);
- ne koriste se deterministička, već probabilistička pravila prelaska;
- tokom procesa rešavanja ne eksploatišu se eventualne dodatne informacije o prirodi problema.

Prema Darvinovom učenju, jedinke neke populacije nadmeću se za resurse kao što su: hrana, voda, skloništa, jedinke - partneri za uparivanje, itd. U osnovi procesa odabiranja koji se odvija u prirodi, posmatranog sa darvinističkog stanovišta, leže sledeće činjenice:

- jedinke bolje prilagođene okolini preživljavaju i imaju veći uticaj na formiranje sledećih pokolenja;
- jedinke jedne generacije u populaciji formiraju novu generaciju na taj način što se nove jedinke dobijaju kombinacijom genetskog materijala roditelja;
- s vremena na vreme dolazi do mutacije, tj. do slučajne izmene genetskog materijala jedne jedinke;

Ako Darvinova teorija o postanku vrsta predstavlja opis prirodnog procesa, onda se može reći da EA imitiraju prirodne procese u cilju rešavanja problema (videti [Tošić97]). Iz ove karakteristike proizilaze tri značajne konsekvence (opisane u [Filipo98]):

- istraživači treba da imaju u vidu da osnovu EA predstavljaju biološke teorije, odnosno da se imitiranje događanja u prirodi svodi na imitiranje istraživačevog shvatanja o tome šta se događa u prirodi (a te dve stvari ne moraju nužno biti iste). Stoga svaki eventualni novi prodor iz oblasti bioloških nauka i genetike može promeniti poimanje o tome kako stvari u prirodi zbilja teku, što bi zasigurno dovelo do izmene strukture samog EA;
- široko primenjive modifikacije EA su obično inspirisane ne samo pokušajem da se postignu bolji rezultati, već i pokušajem da se bolje modelira ponašanje prirode ([Goldbe89]). Drugim rečima, u zajednici EA istraživača je dominantan stav da je priroda učitelj koga treba oponašati, tj. da obično široku primenljivost i popularnost imaju one varijante EA za koje postoji neki analogon među procesima u prirodi;
- EA treba da modeliraju suštinsku, a ne pojavnu stranu procesa i događaja u prirodi.

Stavovi koji korespondiraju prethodno istaknutim tezama se (čak i u nešto oštrijem tonu) sve češće čuju u EA zajednici u poslednje vreme. Tako u radu [Luke00] autori zapažaju da veliki broj ideja koje u računarstvo dolaze iz biologije nastavljaju da daju pozitivan doprinos – tako je npr. regulacija gena omogućila da se stekne uvid u bolje metode evolutivnog izračunavanja. Po rečima autora, sadašnji EA pristup kodiranju, selekciji, ukrštanju i mutaciji potiče iz Mendelovske genetike, čiji je jedan deo dobro funkcionisao, ali drugi deo nije.

Autori (Tate i Smith 1993; Hinterding, Gielewski i Peachey 1995, Angeline 1997) naročito oštro kritikuju EA pristup rekombinaciji gena. U biologiji rezultat rekombinacije se pojavljuje u obliku blagih raznovrsnih gradacija, a u najvećem broju EA domena rekombinacija nije ništa više od randomizacije. U ovom trenutku, najverodostojnija je pretpostavka da do ovolike razlike u tretmanu rekombinacije dolazi zbog ogromnog jaza između biološkog i računarskog kodiranja (reprezentacije). Pokazuje se da pristup preko mapa regulacije gena može da se jedan-jedan preslika na veštački EA domen.

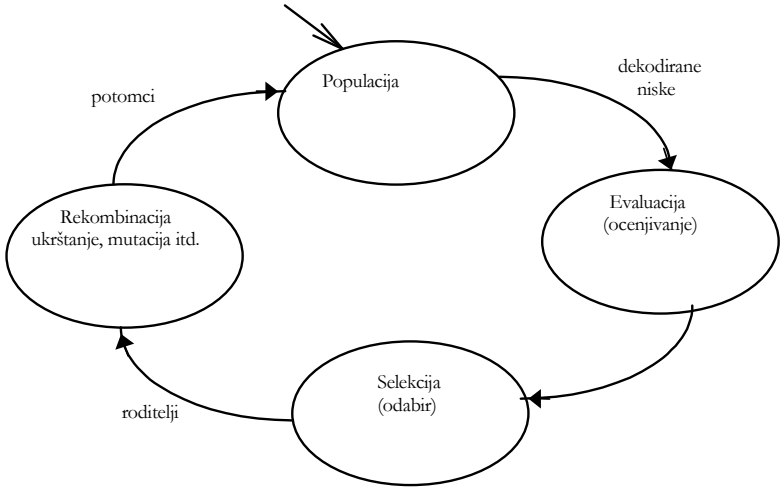
Neophodno je istaći činjenicu da EA dobar dio svog današnjeg značaja i popularnosti duguju činjenici da je postupak dobijanja nove generacije u svojoj suštini paralelan. Tu činjenicu (paralelne prirode reproduktivne paradigme i inherentne, tj. nerazdvojive efikasnosti paralelnog procesiranja) je Holland uočio desetak godina pre formalnog nastanka EA.

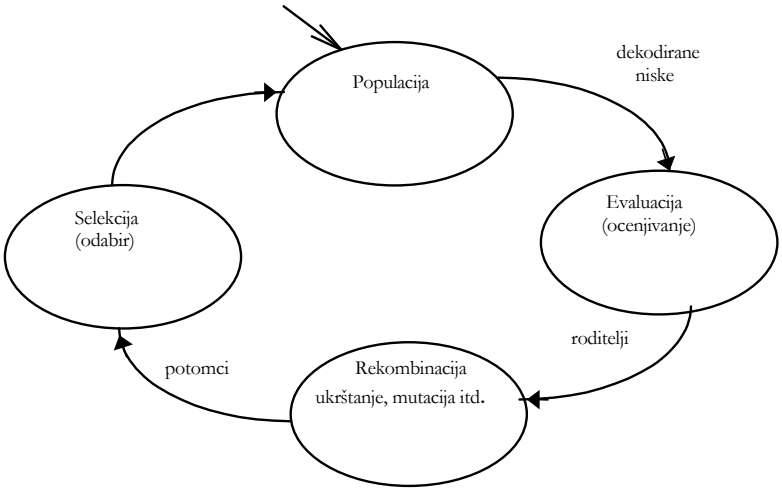
Važna osobina EA je i mogućnost njihove hibridizacije, tj. mogućnost prirodnog kombinovanja EA sa nekim drugim algoritmom za rešavanje zadatog problema. Taj novi, hibridni, postupak se kreira tako da sadrži dobre osobine i jednog i drugog metoda.

1.1.4. Struktura evolutivnih algoritama

Evolutivni algoritmi sadrže:

- pretraživački prostor tj. skup svih mogućih rešenja;
- populaciju tj. skup aktuelnih kandidata za rešenja; elementi populacije su jedinke (čvorovi pretraživanja, tj. tačke u pretraživačkom prostoru);
- prostor niski (stringova), najčešće fiksirane dužine, tj. prostor reprezentacija jedinki, kao i funkcije za preslikavanje pretraživačkog prostora u prostor niski i obratno;
- skup genetskih operatora za generisanje novih stringova, a samim tim i novih jedinki;
- funkciju prilagođenosti (još se naziva i funkcija ocene, eng. fitness function), koja utvrđuje prilagođenost određene jedinke;
- stohastičku kontrolu genetskih operatora;



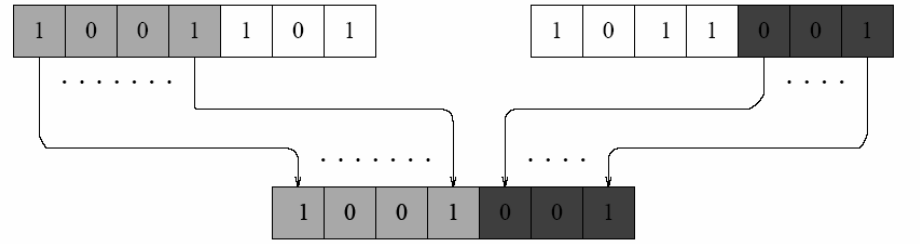


slika 1.2 Struktura EA (dve mogućnosti)

Sa prethodne šeme mogu se uočiti i osnovni koraci EA:
Inicijalizacija - pseudoslučajno se generiše inicijalna populacija jedinki, što se svodi na generisanje elemenata u prostoru niski (u nekim varijantama EA populacija može biti jednočlana). U pojedinim varijantama EA, neke jedinke početne populacije (ili čak cela početna populacija) se ne generišu pseudoslučajno, već se za generisanje koristi neka jednostavnija heuristika.
Evaluacija - izračunavanje funkcije prilagođenosti za svaku jedinku populacije.
Operacija izbora tj. selekcije - biraju se jedinke populacije za preživljavanje na osnovu vrednosti funkcije prilagođenosti konkretne jedinke.
Operacija rekombinacije (obuhvata ukrštanje i/ili mutaciju, ali se ne ograničava na njih) - izmena genetskog koda jedinki tj. elemenata u prostoru niski.
Ponavljanje koraka **2)**, **3)** i **4)** sve dok se ne ispuni kriterijum završetka EA. Neki od najjednostavnijih kriterijuma završetka EA su: maksimalan broj generacija i maksimalan broj ponavljanja najbolje jedinke tokom generacija..

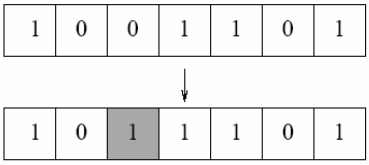
Genetski algoritmi

Genetski algoritmi (eng. Genetic Algorithms - GA) su najviše doprineli popularnosti korišćenja evolutivne paradigme za rešavanje raznovrsnih problema. Njihov nastanak se vezuje za Holland-ov rad "Adaptation in natural and artificial systems" ([Hollan75]). GA karakteriše reprezentacija genetskog koda jedinke pomoću binarnih niski fiksne dužine, operator ukrštanja koji se izvršava nad parovima jedinki i operator mutacije koji na slučajan način menja genetski kod jedinke.
Neophodno je prvo opisati GA koji predstavljaju polaznu tačku za sve te modifikacije. To su tzv. prosti genetski algoritmi (eng. Simple Genetic Algorithm - SGA), sa sledećim operatorima: jednopoziciono ukrštanje (eng. one-point crossover), prosta mutacija (eng. simple mutation), proporcionalna ili rulet selekcija (eng. proportional selection, roulette selection). . Jedinke kod SGA (kao i kod svih ostalih GA) bivaju kodirane binarnim niskama, tj. binarnim stringovima.



slika 1.3 Operator jednopozicionog ukrštanja

Kao što se sa ovih slika i može videti, jednopoziciono ukrštanje se svodi na razmenu genetskog materijala između dve jedinke od neke, na slučajan način odabrane, tačke (pozicije), dok se prosta mutacija svodi na izmenu jednog gena u jedinki.



slika 1.4 Operator proste mutacije

Pored ova dva operatora, izuzetnu važnost kod GA ima i operator selekcije. Taj genetski operator ne modifikuje jedinke u populaciji, već određuje koje će jedinke opstati i tako preneti svoje gene u sledeću populaciju i one koje će biti uklonjene iz populacije. SGA koristi tzv. proporcionalnu (ili rulet) selekciju – formira se rulet sa slotovima čija je veličina proporcionalna prilagođenosti svake od jedinki, a potom se simulira bacanje kuglice na takvom ruletu. Operatori selekcije, pa i prethodno pomenuta proporcionalna selekcija, će biti detaljno opisani u kasnijim poglavljima.

Zbog ovakve reprezentacije (kod modernih računara su, kao što znamo, manipulacije sa bitovima direktno podržane instrukcijama asemblerskog jezika i implementirane u hardveru) genetski operatori i sve aktivnosti tokom izvršenja GA su jako efikasne, pa je izvršavanje GA efikasnije od svake druge EA-bazirane metode.

Postoji veliki broj preglednih radova o genetskim algoritmima. Spomenimo samo neke od njih: [Goldbe89], [Davis91], [Michal96], [Mühlen97], [Mitch99], [Bäck00] i [Bäck00a]. Opšte informacije o GA se mogu naći i u domaćoj literaturi: [Filipo97a], [Kratic97a], [Tošić97], [Filipo98], [Trmčić00] i [Kratic00].

Što se odnosa između EA i GA tiče, očigledno je EA kao pojam natkriljuje GA, tj. da je svaki GA istovremeno i EA. Iako je u EA zajednici usvojen stav da obratna inkluzija ne važi, interesantno je pomenuti da postoje i argumenti koji idu u prilog važenju inkluzije u suprotnom smeru (bar dok se EA izvršavaju na elektronskim računarima). Naime, dok god se EA izvršavaju na računarima gde se informacije pamte u memoriji kao sekvence nula i jedinica, svaka jedinka EA je (bez obzira da li se radi o nizu, drvetu, programu, realnom broju), kad se posmatra na tom nivou apstrakcije, predstavljena sekvencom jedinica i nula ograničene dužine – dakle ima bitovnu reprezentaciju. Operatori ukrštanja, mutacije, dominacije itd., ma kako egzotični bili, sa te tačke gledanja vrše manipulaciju nad nizovima bitova fiksne dužine. Dakle, celokupno EA izračunavanje se može uklopiti u GA.

Evolutivno programiranje

Evolutivno programiranje (eng. Evolutionary Programming - EP) je zamislio Lawrence Fogel, 1960. godine. Evolutivno programiranje se odnosi na domen veštačke inteligencije vezan za sposobnost predviđanja promena u okruženju. Okruženje se opisuje kao sekvenca simbola (iz konačnog alfabeta) i algoritam koji se izvršava je trebao da proizvede novi simbol na izlazu. Izlazni simbol bi trebao da maksimalizuje funkciju isplativosti, a ta funkcija meri tačnost predviđanja.

Primer. Razmotrimo seriju događaja, označenih simbolima $a_1, a_2, \ldots$; algoritam treba da na osnovu prethodnih simbola $a_1, a_2, \ldots, a_n$ predvidi sledeći (nepoznati) simbol a_{n+1}. Ideja kod EP je da se evolucijom dobije takav algoritam. U ovom primeru će konačni automati biti izabrani kao hromozomska

reprezentacija jedinki. Dakle, EP održava populaciju konačnih automata – svaka takva jedinka predstavlja potencijalno rešenje problema, tj. određeno ponašanje. Tokom svog izvršavanja, EP algoritam evaluira prilagođenost svakog konačnog automata u populaciji. To se radi na sledeći način: svaki konačni automat se iz populacije se izlaže okruženju. Izlaganje konačnog automata okruženju se svodi na ispitivanje svih prethodno viđenih simbola (za svaki podniz simbola $a_1, a_2, \ldots, a_i$ se vidi proizvedeni izlaz a_{i+1}' i on se poredi sa stvarno uočenim simbolom a_{i+1} iz okruženja, pri čemu se njihovim razlikama pridružuju određeni težinski faktori). EP prvo formira potomke, a potom vrši selekciju jedinki za sledeću generaciju. Svaka jedinka proizvodi jedan potomak, pa se pre selekcije veličina populacije udvostruči. Potomci, tj. novi konačni automati, se formiraju slučajnim mutacijama roditelja. Obično se koriste sledećih pet operatora mutacije: promena izlaznog simbola, promena prelaska iz stanja u stanje, dodavanje stanja, brisanje stanja i promena početnog stanja (postoje i uslovi vezani za najveći i najmanji broj stanja automata). Verovatnoće primene mutacija mogu da se menjaju tokom izvršavanja algoritma EP.

Evolutivne strategije

Evolutivne strategije (nem. Evolution Strategie - ES) su slične evolutivnom programiranju, iako su nastale potpuno nezavisno. Kreatori evolutivnih strategija su Ingo Rechenberg, Hans-Paul Schwefel i Peter Bienert. Ovaj metod se prvenstveno koristio za rešavanje tehničkih optimizacionih problema. On je doskora bio poznat samo užem krugu inžinjera - i to kao alternativa nekim standardnim rešenjima. Kako za tehničke optimizacione probleme obično ne postoji analitički iskazana funkcija cilja, za takve probleme i ne postoji odgovarajući direktan metod optimizacije, pa evolutivne strategije (zato što ne zahtevaju da funkcija cilja bude precizno definisana, kao što je to slučaj sa drugim metodama optimizacije) kod tehničkih optimizacionih problema mogu dati dobre rezultate ([Bäck91]).

Najveća sličnost između ES i GA je u tome što oboje održavaju populaciju potencijalnih rešenja i pri selekciji koriste princip opstanka najbolje prilagođene jedinke (videti [Michal96]). Osnovna razlika između ES i GA je u domenima njihove primene. ES su, kao što je već istaknuto, razvijene kao metod za numeričke optimizacije i tek nedavno su počele da se primenjuju na diskretne probleme optimizacije. Nadalje, ES operišu nad vektorima brojeva u pokretnom zarezu, dok klasični GA operišu nad vektorima bitova. ES i GA se razlikuju i po operatoru selekcije – kod ES se radi o determinističkoj selekciji (gde bolje prilagođena uvek pobeđuje), dok je kod GA selekcija stohastička (pa se dešava da slabija jedinka koja ima „sreće“ ponekad pobedi bolje prilagođenu). Sledeća razlika između GA i ES je u redosledu primene operatora selekcije i rekombinacije: u ES selekcija se izvršava posle primene rekombinacije, dok je kod GA suprotno. Klasični GA i ES se razlikuju i po tome da li stohastički parametri evolutivnog procesa ostaju nepromenjeni tokom izvršavanja algoritma ili ne – nepromenljivost karakteriše klasične GA, dok dinamičnost parametara karakteriše ES.

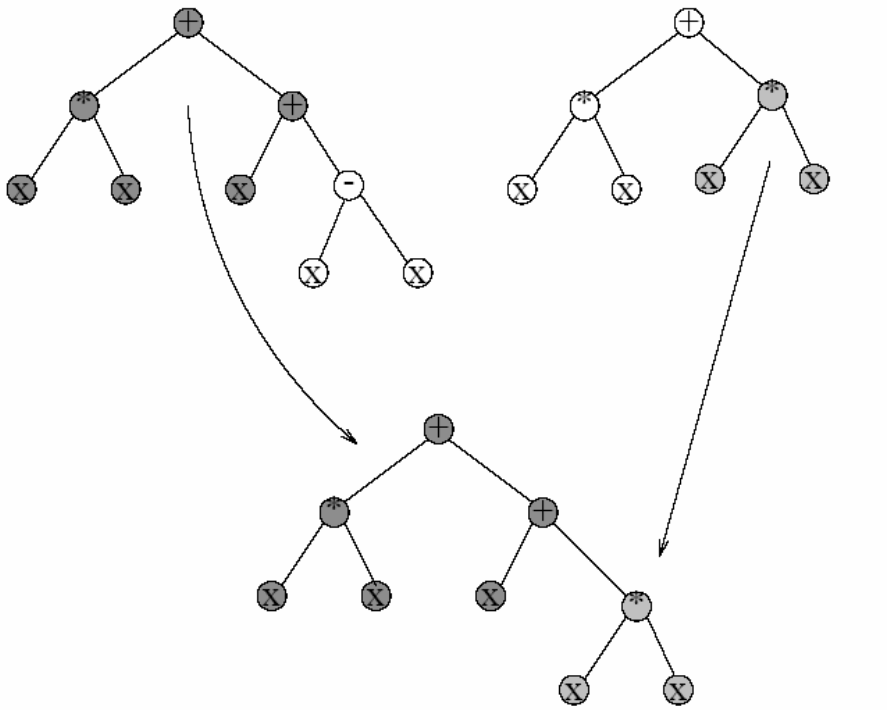
Lako se može uočiti da EP i ES imaju dosta sličnosti: u reprezentaciji pri rešavanju problema, jer za rešavanje problema funkcionalne optimizacije ne koriste binarno kodiranje; u realizaciji genetskih operatora; u korišćenju samoadaptivnih metoda za određivanje kakva će se mutacija koristiti; itd. Najveći broj teorijskih rezultata o konvergenciji izvedenih za EP može direktno da se primeni kod ES, i obratno. Međutim, postoje i razlike između EP i ES: EP koristi stohastičku (obično turnirsku) selekciju, dok ES koristi determinističku selekciju. Treba napomenuti da se metode EP i ES razvijaju tokom vremena, pa sada poseduju mnoge od osobina koje su inicijalno karakterisale samo GA.

Genetsko programiranje

Genetsko programiranje (eng. Genetic Programming - GP) predstavlja nešto noviji pristup rešavanju problema korišćenjem evolutivne paradigme. Njegov začetnik je Koza (videti [Koza91], [Koza93]). GP je raširenje genetskog modela učenja u prostor programa. Ono omogućuje da računar rešava probleme za koje nije bio eksplicitno programiran.

Jedinke koji obrazuju populaciju nisu niske karaktera fiksne dužine, već programi koji, kad budu izvršeni, predstavljaju kandidate za rešenja problema.

Programi se kod genetskog programiranja obično izražavaju drvetom izvođenja, a ne linijama koda. Programi-jedinke su sastavljeni od elemenata tj. simbola iz skupa funkcija i iz skupa terminala. Novi programi se proizvode uklanjanjem grana iz jednog drveta i ubacivanjem u drugo drvo. Ovaj jednostavan proces obezbeđuje da novi program takođe bude sintaksno ispravno drvo.



slika 1.5 Operator ukrštanja kod GP

Kod GP se operator ukrštanja implementira tako što se na slučajan način izaberu čvorovi u drvetima izvođenja kod dve jedinke, pa se poddrveta sa korenom u izabranim čvorovima zamene. Genetsko programiranje obično ne koristi nikakvu mutaciju kao genetski operator.

Modifikacije EA

U dizajnu EA, a radi povećavanja njihove efikasnosti, koriste se raznovrsne modifikacije (naime, autori često vrše prožimanje različitih evolutivno inspirisanih tehnika):

- pri formiranju niske koja reprezentuje jedinku - npr. diploidna reprezentacija (češća u prirodi, gde postoje po dva gena za svaku osobinu, dominantni i recesivni) i haploidna reprezentacija (gde jedan gen nosi informaciju) (videti [Goldbe89]);
- pri kodiranju, odnosno preslikavanju pretraživačkog prostora u prostor niski - npr. klasično kodiranje, Grejovi kodovi (karakteriše ih činjenica da se kodovi bliskih jedinki razlikuju na malom mestu bitovnih mesta - videti [Heitko93]), višedimenzionalna reprezentacija (videti [Bui95]), drvoidna reprezentacija, reprezentacija pomoću brojeva u pokretnom zarezu itd.;
- pri računanju funkcije prilagođenosti i pri realizaciji jednog koraka EA ([Kratic97], [Kratic99]); tu su i modifikacije nazvane zbrkani EA (eng. messy EA - mEA) (videti [Goldbe91]), kompaktni EA (u [Harik97]), te virtualni EA (eng. virtual EA) (videti [Grefen95]);
- pri izboru i dizajnu genetskih operatora - tako se za diploidnu reprezentaciju uvodi dominacija (eng. dominancy) (videti [Goldbe89]); nadalje, pored jednopozicionog ukrštanja koriste se i uniformno ukrštanje (eng. uniform crossover), zamućeno ukrštanje (eng. shuffle crossover) itd. (videti [Syswer89], [Caruana91], [Radcli92], [Eshelm93], [Kahng95]); tu su i raznovrsni operatori selekcije ([Whitle89], [Bäck00a]), koji će biti detaljnije opisani u sledećim poglavljima; u nekim eksperimentima se dopušta da nad istom populacijom deluje npr. više operatora ukrštanja, svaki sa svojom verovatnoćom realizacije; sa mutacijom je slično kao sa ukrštanjem – postoji veliki broj vrsta mutacija - specifične za problem, mutacije koje primenjuju hibridizaciju itd. (videti [Spears00]).
- pri utvrđivanju politike zamene jedinki u populaciji - dok su originalni EA zasnovani na potpunoj smeni generacija (zato se još zovu i generacijski EA), u nekim modifikacijama EA, nazvanim

stacionarni EA (ili EA stabilnog stanja, eng. steady state EA), dopušta se da roditelj, naravno pod uslovom da je nivo njegove prilagođenosti zadovoljavajući, nastavlja da živi naporedo sa svojim potomcima; U konkretnim realizacijama EA često se eksploatiše međurešenje - najbolje prilagođena jedinka iz prethodne generacije nastavlja svoj život, i to je poznato pod imenom elitna strategija (eng. elitist strategy).

- pri utvrđivanju stohastičkih veličina, tj. verovatnoća koje određuju primenu genetskih operatora. Vršeni su mnogobrojni, dosta uspešni, pokušaji poboljšavanja EA u smislu da se vrednosti kontrolnih parametara algoritma menjaju tokom vremena (tj. tokom izvršavanja samog algoritma) i to pomoću nekog adaptivnog mehanizma ([Spears91], [Bäck92], [Aizawa93], [Spears95a] itd.). U takvim slučajevima se govori o meta EA, pri čemu su argumenti meta EA baš vrednosti kontrolnih parametara samog EA.
- pri izvršavanju evolutivnog algoritma – pored sekvencijalnih, postoje i razne vrste paralelnih EA i o njima će biti više reči u sledećem poglavlju.
- pri utvrđivanju kriterijuma završetka - npr. da li se najbolja jedinka promenila tokom fiksiranog broja generacija, da se li više od pola jedinki u populaciji poklapa sa najboljom, da li se prosečna vrednost jedinki malo razlikuje od najbolje, itd.

Već je istaknuta mogućnost stvaranja hibrida EA i nekog drugog algoritma (videti [Kelly92], [Burke95]). Mnogobrojni autori su dizajnirali raznovrsne hibridne EA: hibrid ES i EA (videti [Bull94]); hibrid stohastičkog penjanja uz brdo (eng. stochastic hillclimbing) i EA (rad [Juels94]); hibrid pretrage po susedima (eng. neighborhood search) i EA ([Reevs94]), hibrid EA i neuralnih mreža ([Spears95]).
Potrebno je istaći da je paralelno sa razvojem tehnika i primena EA razvijana (i još se razvija) odgovarajuća teorija, kojom se pokušava dati odgovor na pitanja o značaju i kvalitetu kako "čistog" EA, tako i pojedinih njegovih modifikacija. Zbog složene i slučajne prirode samog EA, određivanje nekih osobina primenjenog algoritma zahteva određeno matematičko znanje, a rezultati koji se takvim izvođenjem dobijaju nisu ni precizni u onoj meri u kojoj smo na to navikli kod deterministčkih metoda. Ipak, razvoj matematičkog aparata koji predstavlja teorijsku osnovu EA relativno kratko traje, a već su postignuti određeni rezultati.

1.2. Paralelni evolutivni algoritmi

1.2.1. Višeprocesorske arhitekture i paralelni računari

Programiranje višeprocesorskih računara je mnogostruko složenije od programiranja jednoprocesorskih računara, jer mnogo više faktora bitno utiče na ukupno vreme izvršavanja (detaljnije videti u [Bertse89], [Мирећк89] [Lewis92], [Filipo97] i [Dongar03]). Takođe se pri i analizi efikasnosti algoritama i vremena izvršavanja programa na višeprocesorskim računarima susrećemo sa brojnim problemima koji ne postoje pri proceni složenosti sekvencijalnih algoritama i programa.
Ubrzavanje izvršavanja programa na višeprocesorskim računarima se, osim standardnih tehnika primenjivanih na jednoprocesorskim sistemima, uglavnom postiže maksimalnom iskorišćenošću procesora, odnosno podešavanjem algoritma tako da pojedini procesori čekaju što je moguće manje druge procesore. Koliko je to moguće, zavisi direktno od prirode problema, odnosno to od toga da li se i kako problem može razložiti na potprobleme, koji što manje zavise jedni od drugih. Tada se svi potproblemi mogu istovremeno izvršavati na različitim procesorima, uz najmanju moguću sinhronizaciju i međuprocesorsku komunikaciju. Ukoliko nije moguće da potproblemi budu potpuno nezavisni, teži se ka tome da međusobna zavisnost bude što manja. Na taj način se problemi sinhronizacije procesa i međuprocesorske komunikacije svode na najmanju moguću meru.

Modeli računara

S obzirom na popularnost i širinu korišćenja paralelnih računara, a da bi se omogućila adekvatna i precizna analiza paralelnih algoritama koji se izvršavaju na ovakvim računarima, neophodno je izvršiti klasifikaciju tih računara.,
Postoji više podela računarskih sistema, od kojih je jedna od najstarijih i najpoznatijih Flynn-ova podela (videti [Flynn66], [Flynn72] i [Ribar86]), gde se računari po arhitekturi dele na:

- Jednostruki tok instrukcija i jednostruki tok podataka (eng. Single Instruction Stream Single Data Stream - SISD) gde jedan procesor sekvencijalno izvršava jedan tok instrukcija i operiše nad jednim tokom podataka.
- Mnogostruki tok instrukcija i jednostruki tok podataka (eng. Multiple Instruction Stream Single Data Stream - MISD) gde *N* procesora, svaki sa po jednom kontrolnom jedinicom, deli isti tok podataka. Svi procesori, u svakom koraku, obrađuju istovremeno jedan podatak prihvaćen iz zajedničke memorije, prema instrukciji koju svaki procesor prima sa sopstvene upravljačke jedinice.
- Jednostruki tok instrukcija i mnogostruki tok podataka (eng. Single Instruction Stream Multiple Data Stream - SIMD) gde svaki od *N* identičnih procesora sa zajedničkom kontrolnom jedinicom izvršava isti program nad svojim lokalnim podacima. Izvršavanje se odvija sinhrono, gde svaki procesor, nad svojim tokom podataka, istovremeno izvršava istu instrukciju.
- Mnogostruki tok instrukcija i mnogostruki tok podataka (eng. Multiple Instruction Stream Multiple Data Stream - MIMD) je najsloženiji model, u kome svaki procesor poseduje sopstvenu lokalnu memoriju. Pošto svaki procesor ima svoj tok instrukcija i podataka, oni obično rade asinhrono, što znači da, u istom vremenskom trenutku, mogu izvršavati različite instrukcije nad različitim podacima. Asinhrono izvršavanje uzrokuje brojne probleme, koji se ne javljaju kod prethodnih 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 sistemi sa komunikacijom preko deljene memorije (eng. shared memory multiprocessors).
- Višeprocesorski sistemi sa komunikacijom prosleđivanjem poruka (eng. message passing multiprocessors).

Komunikacija preko deljene memorije

U ovom modelu svi procesori imaju jedinstven memorijski prostor zajednički za sve procesore. Pošto operativni sistem sam mora da vodi računa o paralelnom čitanju i upisu, posledica je olakšano korišćenje i programiranje ovakvih računara. Jedna od najpoznatijih programskih biblioteka ujedno i IEEE standard je Pthread (videti [Nichol96]). Operativnom sistemu je prilično teško da obezbedi efikasno paralelno čitanje i upis, pogotovo kod višeprocesorskih sistemima sa velikim brojem procesora (videti [Akl89]). Međutim, u poslednje vreme OpenMP standard (detaljno opisan u [Chandra01]) prilično dobro rešava datu anomaliju korišćenjem paralelizma u više nivoa (eng. multilevel parallelism, nested parallelism). To se može videti i na osnovu rezultata primena na neke realne probleme, čiji se detaljniji opis može naći u radu [Bücker04]. Budući da OpenMP relativno jednostavno podržava dodavanje paralelnih konstrukcija u postojeći sekvencijalni kod, on je u poslednje vreme postao najčešće korišćen model višeprocesorskih sistema sa komunikacijom preko deljene memorije.

Komunikacija prosleđivanjem poruka

Pojavom višeprocesorskih računara sa više stotina ili hiljada procesora, odnosno korišćenjem mreže radnih stanica i Interneta kao višeprocesorskih sistema, naročito su postali popularni paralelni računari gde se međuprocesorska komunikacija ostvaruje prosleđivanjem poruka (eng. message passing). Projektovanje i implementacija takvih sistema je relativno laka, ali je na njima teže programirati, pošto program (a ne operativni sistem) vodi računa o sinhronizaciji procesa i razmeni podataka između njih.

Međuprocesorska komunikacija je, kod ovih sistema, jedan od presudnih faktora koji utiče na ukupno vreme izvršavanja. Vreme potrebno za prenos određenog podatka ili grupe podataka, između različitih procesora, je ponekad, nekoliko puta veće od vremena potrebnog za aritmetičku operaciju nad njim (videti [Kratic94]). Obično veća iskorišćenost procesora zahteva i veću međuprocesorsku komunikaciju, i obrnuto, pa se mora naći kompromis koji obezbeđuje minimalno vreme izvršavanja.

U početku je svaki proizvođač paralelnih računara konstruisao i sopstveni programski sistem za razvoj paralelnih aplikacija (Cray Research, IBM, Intel, Meiko, Ncube i transpjuteri). Ovakvi sistemi su bili vrlo različiti, a aplikacije praktično neprenosive između njih. Kasnije se pojavio veći broj pokušaja standardizacije ovakvih sistema: CHIMP (videti [CHI91] i [CHI92]), p4 (videti [Butler94]), PVM (videti [Geist94]), Zipcode (rad [Skjell92]). Međutim, svaki od ovih sistema je posedovao određene nedostatke, i umesto nadogradnje nekog od njih pristupilo se konstruisanju novog sistema, uz standardizaciju višegodišnjeg iskustva eksperata.

MPI

MPI (eng. Message Passing Interface) je specifikacija standardne biblioteke funkcija za paralelni model prosleđivanjem poruka. Definisanje i razvoj ovog standarda je trajalo preko dve godine od strane profesionalaca za paralelne računare, njihovo programiranje i razvoj. Posledica toga je standard koji je podržan na velikom broju različitih paralelnih platformi, efikasan i sa programskim konstrukcijama relativno visokog nivoa. MPI standard je preuzeo konstrukcije raznih postojećih sistema koji su se dobro pokazale u praksi, i ugradio ih u jedan pouzdan i efikasan sistem visokog nivoa. Bez zahteva za kompatibilnošću sa starim verzijama, bilo je moguće projektovati standard koji zadovoljava i najzahtevnije primene, uz očuvanje efikasnosti izvršavanja. Početni dogovor oko MPI standarda je nastao aprila 1992 na međunarodnoj konferenciji Supercomputing '92 (videti [Dongar93]) a konačna verzija MPI standarda 1.0 nastala je u maju 1994 (detaljno u [MPI94] i [MPI95]).

MPI standard sadrži sledeće elemente:

- pojedinačne međuprocesorske komunikacije (eng. 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.

Kako je MPI standard nezavisan od platforme i operativnog sistema na kojem 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 C, Fortran ili Java prevodioci i ostali alati posebno konfigurisani i podešeni za MPI aplikacije.

Iskustvo tima koji je učestvovao u stvaranju MPI standarda, imalo je za posledicu pojavu prvih konkretnih MPI implementacija vrlo brzo nakon usvajanja standarda. Kasnije je MPI standard postao prihvaćen od mnogih i pojavio se veliki broj MPI implementacija za razne vrste paralelnih računara od mreže radnih stanica do superračunara. 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.

Web Servisi

Web servisi, su bitni u razvoju web-baziranih tehnologija i predstavljaju distribuisane serverske softverske komponente. Preciznije, web servis predstavlja bilo koji servis dostupan u distribuiranim okruženjima, kao što je Internet ili intranet mreže, a koji koristi standardizovani XML sistem za razmenu poruka, te koji nije isključivo vezan za konkretan operativni sistem ili programski jezik. Dakle, web servisi predstavljaju moćan metod za pružanje servisa kojima se (preko Internet-a ili intranet-a) može pristupiti programskim putem.

Ono što, verovatno, stvara veliku konfuziju kod neupućenih u vezi web servisa je sam termin, koji i nije baš najbolji izbor, jer reč "servisi" upućuje na pružanje neke usluge. Ali, u svojoj osnovi, web servisi su samo revizija standardnih softverskih tehnologija. Oni programerima pružaju mogućnost povezivanja raznorodnih sistema na novi način, preko Interneta, i to unutar jednog ili više poslovnih procesa.

Već je istaknuto da web servisi mogu biti korišćeni interno (od strane jedne aplikacije) ili izloženi eksterno preko Interneta za korišćenje od strane većeg broja aplikacija. Pored toga, više različitih aplikacija može koristiti jedan web servis. S obzirom na to da su dostupni preko standardnih interfejsa, web servisi dopuštaju da heterogeni sistemi rade zajedno kao jedna mreža izračunavanja.

Web servisi su izgrađeni na standardima kao što su protokol za poziv udaljenih metoda SOAP (eng. Simple Object Access Protocol), proširiv jezik za označavanje XML (eng. eXtensible Markup Language), i jezik za opis web servisa WSDL (eng. Web Service Description Language) o kojima će biti više reči u sledećim poglavljima.

Jedna od osnovnih karakteristika web servisa je visok nivo apstrakcije između implementacije servisa i korišćenja servisa. Korišćenjem XML-zasnovanih poruka kao mehanizma za kreiranje i pristup servisu,

klijent web servisa i sam web servis su oslobođeni potrebe da znaju bilo šta o onom drugom, osim ulaza, izlaza i lokacije (adrese).

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.

Simulatori paralelnih računara

Upoznavanje sa paralelizacijom i paralelnim programiranjem je najjednostavnije (i uz minimalne troškove) realizovati, korišćenjem simulatora (emulatora) višeprocesorskih računara. Njihova namena može biti:

- Modeliranje nekog realnog višeprocesorskog sistema u cilju bližeg upoznavanja sa datim sistemom i eventualno razvijanje konkretnih aplikacija. Pri tome bi sve pripremne radnje u razvoju date aplikacije bile urađene na simulatoru, jedino će konačno izvršavanje aplikacije biti realizovano na konkretnom paralelnom računaru.
- Simulatori nekog imaginarnog višeprocesorskog sistema, čija je najvažnija namena što lakše učenje i upoznavanje paralelnih konstrukcija. U ovim slučajevima kompatibilnost sa ostalim sistemima i performanse datog simulatora nisu od bitnog značaja.

Ovakvi sistemi su uglavnom namenjeni za učenje i obično nisu pogodni za širu primenu. Iako je teorijski moguće simulirati sve elemente paralelnog računara, u praksi rezultati u velikoj meri odstupaju od očekivanih. Simulacija na jednoprocesorskom računaru obično ne može predstaviti na pravi način neke, vrlo važne, fizičke karakteristike višeprocesorskog računara: relativno spora međuprocesorska komunikacija, problemi sa paralelnim čitanjem/pisanjem u memoriji, različite osobine pojedinačnih procesora unutar paralelnog računara i neke od ostalih osobina realnog paralelnog računara. Rezultati dobijeni na simulatoru mogu biti potpuno nereprezentativni, pa čak i netačni, i obično u velikom neskladu sa rezultatima iste aplikacije na realnom paralelnom računaru. Pošto su u početku višeprocesorski računari bili uglavnom vrlo skupi i nepristupačni, ovakav pristup je često korišćen, i pored pomenutih nedostataka.

Multiprocesorski sistemi nižih performansi

Prvi šire dostupni višeprocesorski sistemi relativno prihvatljive cene i performansi su se pojavili 80-tih godina (transpjuteri, ICUBE, itd). Transpjuterski sistemi su bili najpoznatiji predstavnici ove klase paralelnih računara, zasnovanih na osnovnim procesorima - transpjuterima, uz međuprocesorsku komunikaciju prosleđivanjem poruka. Ovi sistemi su realizovani kao ploče koje su se ugrađivale u PC računare ili Sun radne stanice, koji su obezbeđivali ulazno/izlazne operacije, korišćenjem postojećih operativnih sistema domaćinskih računara (eng. host computers). Paralelne konstrukcije su nadograđivane u postojeće programske jezike (paralelni C) ili su razvijani potpuno novi programski jezici prilagođeni paralelnom izvršavanju (Occam) (detaljnije u [Mock89]).

Iako tehnički ograničeni, često podložni kvarovima i sa malim brojem programskih alata, transpjuterski sistemi su u vreme pojavljivanja doprineli širokoj popularizaciji paralelnih računara i upoznavanju šireg kruga ljudi sa paralelnim konstrukcijama na realnom višeprocesorskom računaru.

Mreža radnih stanica

Zbog svoje niske cene, najčešće platforme za razvoj, testiranje i izvršavanje programa i rešavanje raznih problema, su bile (a i danas su) radna stanica (eng. workstation) i PC računar. Veliko i stalno poboljšanje performansi PC računara je praktično ukinulo razliku između prethodnih pojmova . Uporedo sa korišćenjem pojedinačnih PC računara/radnih stanica pristupilo se i njihovom povezivanju i umrežavanju u lokalne (eng. Local Area Networks - LAN) i globalne računarske mreže (eng. Wide Area Networks - WAN). Pravi pomak u pouzdanosti ovih mreža i korišćenju svih prednosti umrežavanja je, na žalost, učinjen tek korišćenjem prvih sigurnijih i fleksibilnijih operativnih sistema za PC (UNIX, Windows NT). Tek tada je postalo praktično moguće, kao paralelni računar, koristiti novu ili postojeću mrežu radnih stanica. U slučaju PC računara se prednosti više procesora i paralelnog izvršavanja mogu dobiti na dva načina:

- Korišćenje matičnih ploča sa više procesora (2-8), koji komuniciraju preko zajedničke (deljene) memorije;
- Korišćenje LAN kao paralelnog računara, gde procesori komuniciraju prosleđivanjem poruka preko date lokalne mreže.

Razni aspekti mreža radnih stanica kao paralelnog računara mogu se videti u [Wilki98] i [Koncha98]. Brz razvoj Interneta i odgovarajuće infrastrukture u poslednjih dvadesetak godina doveo je do mogućnosti da se radne stanice koja izvršavaju određeni algoritam ne nalaze u lokalnoj mreži, već da su na Internet –u. U tom kontekstu, Internet procesiranje možemo posmatrati kao specijalan slučaj mreže radnih stanica. Prednost ovakvog pristupa je potencijalno veliki broj računara koji mogu biti korišćeni za izvršavanje, a mana prilično sporija međuprocesorska komunikacija. Zbog toga ovakav pristup daje izuzetne rezultate kod rešavanja problema gde se međuprocesorska komunikacija može smanjiti na minimum.

Superračunari i paralelni računari visokih performansi

Neke zajedničke osobine superračunara i višeprocesorskih sistema visokih performansi su:

- Strogo kontrolisan i vrlo komplikovan pristup takvim sistemima, zbog njihove vrlo visoke cene;
- Vrlo skupo računarsko vreme i visoki troškovi razvoja, testiranja i izvršavanja aplikacija;
- Izuzetno veliki napor u optimizaciji i efikasnom paralelnom izvršavanju, koji obavljaju specijalizovani prevodioci datog programskog jezika ili sam programer. U suprotnom može doći do osetne degradacije performansi celog paralelnog računara;
- Prethodne tehnike optimizacije i poboljšanja performansi se obično mogu primeniti samo na tu klasu višeprocesorskih računara i često su neprimenjive na ostale klase paralelnih računara.

Izuzetno visoka cena nabavke i troškovi održavanja superračunara je vrlo ograničavajući faktor, pa ih uglavnom poseduju samo velike kompanije i bogate institucije i praktično su nedostupni za širu klasu istraživača i programera. Cene paralelnih računara visokih performansi su znatno manje u odnosu na superračunare, ali su takođe uglavnom prevelike, pa ih najčešće poseduju samo veći univerziteti i istraživački centri u zemljama zapadne Evrope, SAD i Japana. Pristup im je takođe vrlo ograničen na uzak krug istraživača i programera u tim institucijama.

1.2.2. Klasifikacija paralelnih evolutivnih algoritama

Osnovna ideja kod mnogih paralelnih programa je da podele zadatak u podzadatke i da korišćenjem više procesora simultano rešavaju podzadatke. Ovaj pristup može biti iskorišćen na različite načine, što dovodi do različitih metoda za paralelizaciju EA (videti [Mühlen89], [Mühlen92], [Shonkw93], [Stanle95], [Cantu97], [Cantu00], [Vinc02], [Li02] itd.). Neki metodi paralelizacije menjaju ponašanje algoritma, a neki ne. Neki od metoda mogu da eksploatišu paralelne računarske arhitekture sa ogromnim brojem procesora, dok su druge bolje prilagođene paralelnim računarima sa manjim brojem moćnijih procesora.

U radu [Kapsal94] je izvršen pokušaj postavljanja jedinstvene paradigme koja se odnosi na sve paralelne EA.

Istorijski gledano, od pojave paralelnih EA počinje i njihova klasifikacija - Grefenstette sa četiri prototipa (1981): sinhronizovani gospodar-rob (eng. master-slave), polusinhronizovani gospodar-rob, distribuisani konkurentni i mrežni, potom Schwehm u [Schweh96], pa Cantu u [Cantu00] – da nabrojim samo nekoliko njih.

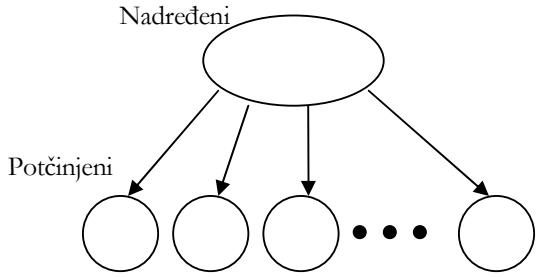
Danas se najčešće srećemo sa sledećom klasifikacijom paralelnih evolutivnih algoritama:

- Globalni paralelni EA
- Grubo usitnjen paralelni EA
- Fino usitnjen paralelni EA
- Hibridni paralelni EA

Globalni paralelni EA

Prvi metod paralelizacije EA je globalna paralelizacija. Globalni paralelni EA sadrže samo jednu populaciju (kao i kod klasičnih serijskih EA), ali se određivanje prilagođenosti jedinki i izvršavanje genetskih operatora eksplicitno paralelizuju. S obzirom da je populacija jedinstvena, selekcija uzima u obzir sve jedinke i svaka jedinka ima šansu da se rekombinuje sa ma kojom drugom, pa ponašanje algoritma ostaje nepromenjeno. Ovaj metod se relativno lako implementira i daje značajno ubrzanje ukoliko vreme komunikacije nije

dominantni deo vremena izvršavanja EA (npr. tamo gde je funkcija prilagođenosti veoma složena, pa izračunavanje prilagođenosti traje jako dugo).



slika 2.1 Globalni paralelni EA

Kao što se vidi sa gornje slike, kod globalnih paralelnih EA uočavaju se „nadređeni" i „potčinjeni". Nadređeni čuva populaciju, izvršava EA operatore i distribuiše jedinke robovima, koji izračunavaju prilagođenost prosleđene jedinke. Zbog ovakve svoje strukture, globalni paralelni EA se još nazivaju i Gospodar-rob paralelni EA. U radu [ChenH98] je prikazan Globalni paralelni EA .

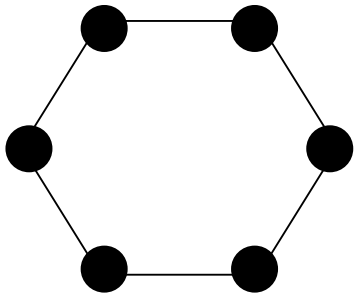
Grubo usitnjen paralelni EA

Grubo usitnjen paralelni EA (eng. Coarse Grained Parallel EA) koristi drugačiju ideju. U ovom slučaju, populacija je podeljena na potpopulacije koje najveći deo vremena evoluiraju nezavisno jedna od druge, ali koje s vremena na vreme razmene jedinke. Ova razmena jedinki se zove migracija i nju kontroliše nekoliko parametara.

Grubo usitnjen paralelni EA je veoma popularan i zbog toga će se u poglavljima koja slede njemu posvetiti veća pažnja nego ostalim vrstama paralelnih EA.

Grubo usitnjen paralelni EA se jako teško kontroliše, jer efekat migracije nije tako iscrpno proučen, kako je to slučaj sa drugim genetskim operatorima. Tokom implementacije ovog paralelnog algoritma, autor treba da odredi broj i veličinu potpopulacija, frekvenciju migracija, broj i odredište migranata, kao i metod kojim se bira koje jedinke će migrirati.

Napomena. Iako migracija u svakodnevnom govoru znači preseljenje, kod paralelnih EA je uobičajeno da migranti ne napuštaju potpopulaciju u kojoj su potekli i da umesto originala, njihova kopija putuje u odredišnu potpopulaciju.



slika 2.2 Grubo usitnjen paralelni EA (topologija veza može biti i drugačija)

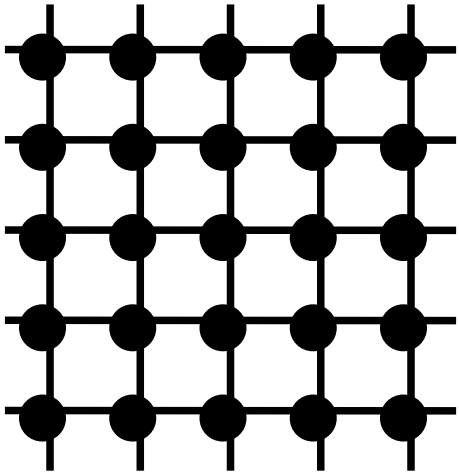
Grubo usitnjen paralelni EA uvodi fundamentalne promene u način izvršavanja EA i njegovo ponašanje se razlikuje od ponašanja sekvencijalnog EA. Ovaj paralelni EA je poznat i pod nazivom Distribuisani EA, zato što su ovi algoritmi obično implementirani na MIMD računarima sa distribuisanom memorijom. Ova klasa paralelnih EA se često naziva i Ostrvski Paralelni EA„ jer u genetici populacija postoji model strukturisanja populacije koji razmatra relativno izolovane potpopulacije i taj model se zove ostrvski model.

U poglavlju 1.3. je već istaknuto da modifikacije EA treba da budu inspirisane ne samo pokušajem da se postignu bolji rezultati, već i pokušajem da se bolje modelira ponašanje prirode ([Tošić97], [Filipo03]). Veći

broj autora je (videti [Cohoon91a], [Cantu97], [Cantu00]) uočio da grubo usitnjen paralelni EA predstavlja bližu analogiju situaciji u prirodi, nego što je to slučaj sa sekvencijalnim EA (u kojem postoji jedna monolitna populacija). Dalje, već su uočene sličnosti (u [Cantu00]) između evolutivnog procesa kod paralelnih EA i tzv. Teorije tačkaste ravnoteže koja se pojavila u 1977.-78. Autori ove teorije su Niels Eldridge iz Američkog prirodnjačkog muzeja i Jay Gould, profesor paleontologije na Harvardu. Ta teorija je nastala u pokušaju da se objasne pojave u prirodi koje se nisu uklapale u dotad najšire prihvaćenu Darvinovu teoriju. Teorija tačkaste ravnoteže sugeriše da tokom najvećeg dela vremena nema značajnijih promena u populaciji (tj. populacija je u ravnoteži), ali da onda neki događaji mogu biti okidač za brze evolutivne promene. Tako, na primer, nepostojanje fosilnih ostataka prelaznih životnih oblika između reptila i sisara, kao i dokazi o koegzistenciji nekih vrsta koje bi trebalo da se pojavljuju jedne posle drugih predstavljaju veliki problem za darvinizam, ali su potpuno konzistentne sa teorijom tačkaste ravnoteže. Jer, ukoliko se dešavaju evolutivni skokovi, tada neće biti prelaznih oblika. Nadalje, ako se (prema ovoj teoriji) skokovi najpre dešavaju na manjem delu populacije, onda se može očekivati i koegzistencija jedinki različitog nivoa evolucije. Teorija iznosi pretpostavku da su se ovi evolutivni skokovi dešavali prilikom izlaganja manjeg dela populacije nekom abnormalnom stresu. Stres tada dovodi do prekida ravnoteže u izloženoj zajednici i uspostavlja unutrašnje preturbacije koje je moguće rešiti jedino skokom koji će uspostaviti novu ravnotežu na višem nivou.
Cohoon je u radu [Cohoon91a] uočio da migracija može biti takav događaj. Drugi autori su (videti [Sumida90], [Cantu97]), takođe uočavajući veću sličnost paralelnih EA i procesa u prirodi, analizirali grubo usitnjen paralelni EA inspirisani Wright-ovim modelom pomerenog balansa, koji je nastao 1932. godine. U suštini, Wright-ov model pomerenog balansa opisuje male populacije koje pronalaze različita lokalno-optimalna rešenja. Populacije šalju emigrante, koji imaju efekat privlačenja drugih potpopulacija ka njihovim rešenjima. Pri tom, privlačenju se potencijalno može proći kroz udubljenja (ili doline) sa niskom prilagođenošću, koji bi inače ostali neistraženi – a često neka takva dolina „krije“ optimalno ili bolje suboptimalno rešenje.
Pored ovih bioloških implikacija, koje su vrlo značajne, potrebno je korišćenje paralelnih EA analizirati i sa stanovišta performansi - grubo usitnjeni paralelni EA su značajni zato što je veličina potpopulacija manja od veličine populacije kod sekvencijalnog EA i stoga bi algoritam na potpopulacijama (gde svaku potpopulaciju obrađuje jedan procesor) morao da konvergira brže nego u sekvencijalnom slučaju. Naravno, pri poređenju performansi sekvencijalnog i paralelnog EA, mora se voditi računa i o kvalitetu rešenja koja su pronađena – jer iako male potpopulacije obično brže konvergiraju, obično je i rešenje prema kojem one konvergiraju slabijeg kvaliteta. Određivanje balansa između brzine izvršavanja algoritma i kvaliteta dobijenih rešenja je veoma složen problem.
Više detalja o Distribuiranim EA može se videti i u [Beldin95], [Alba99].

Fino usitnjen paralelni EA

Treći pristup u paralelizaciji EA koristi fino usitnjen paralelizam. Fino usitnjen paralelni EA (eng. Fine Grained Parallel EA) deli populaciju u veliki broj veoma malih potpopulacija koje su prostorno strukturisane (bilo bi idealno da postoji tačno jedna jedinka za svaki procesni element koji je na raspolaganju). Ovaj model je prilagođen masivnim paralelnim računarima i prostorna struktura potpopulacija odslikava hardverske veze među procesnim elementima na masivnom paralelnom računaru na kojem se algoritam izvršava.

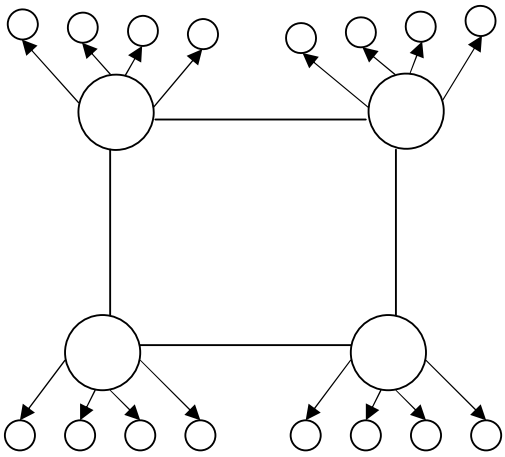


slika 2.3 Fino usitnjen paralelni EA

Kod fino usitnjenog paralelnog EA, selekcija i ukrštanje su ograničeni samo na neposredne susede svake jedinke. S obzirom da se susedstva preklapaju, i dalje se može dogoditi da se tokom vremena superiorna jedinka raširi preko cele populacije.
Ova klasa paralelnih EA se u literaturi još naziva i Difuziono modelirani EA , zato što se dobre jedinke šire kroz populaciju analogno procesu difuzije kod fluida. Za fino usitnjene paralelne EA se još koristi i termin Ćelijski EA, jer oni predstavljaju klasu ćelijskih automata sa stohastičkim pravilima prelaska (videti [Mander89], [Spiess91], [Whitley93], [Li02], [Vinc02]).

Hibridni paralelni EA

Poslednji metod paralelizacije EA koristi neke kombinacije prethodna tri metoda. Ova klasa evolutivnih algoritama se naziva Hibridni paralelni EA. Kombinovanje tehnika paralelizacije dovodi do algoritama koji sadrže pozitivne karakteristike uključenih komponenti.
Najčešće se kod hibridnih paralelnih EA vrši kombinacija više potpopulacija sa globalnim paralelnim EA u dva hijerarhijska nivoa – na višem nivou radi se o algoritmu sa više potpopulacija, dok se na nižem nivou, u okviru svake potpopulacije, izvršava globalni paralelni EA ili fino usitnjen paralelni EA. Hijerarhija paralelnih EA, kao i kod svake druge hibridizacije, kombinuje prednosti obeju njegovih komponenti, čime se često postižu bolje performanse.



slika 2.4 Primer hijerarhijskog hibridnog paralelnog EA

Hijerarhijski hibridni paralelni EA su veoma dobro prilagođeni za izvršavanje na simetričnim multiprocesorskim računarima.

1.2.3. Rezultati primene paralelnih evolutivnih algoritama

Istraživači su paralelnim evolutivnim algoritmima poklonili veliku pažnju iz dva razloga. Prvi razlog je brz razvoj višeprocesorskih računarskih sistema i računarskih mreža. Drugi, ne manje značajan, razlog je paralelna priroda reproduktivne paradigme koja se koristi u EA. Naime, EA se, za razliku od najvećeg broja drugih klasa algoritama, mogu lako i prirodno paralelizovati.

U radu [Tanese87] se grubo paralelni usitnjeni EA koristi za optimizaciju Walsh-olikih funkcija korišćenjem paralelnog računara sa 64 procesora povezanih u hiperkocku (eng. hypercube). Iz prikazanih rezultata se vidi skoro linearno ubrzanje, u odnosu na odgovarajući sekvencijalni EA, a kao najbolji kompromis se preporučuje korišćenje 8 potpopulacija. Kasnija unapređenja ove implementacije su opisana u [Tanese89].

Sličan, ali nešto drugačiji pristup je korišćen u [Starkw91]. Svaka potpopulacija se takođe izvršava na svakom pojedinačnom procesoru i one razmenjuju jedinke tokom izvršavanja algoritma. Primljene jedinke u svakoj potpopulaciji zamenjuju bivše najlošije jedinke. Istraživan je uticaj veličine potpopulacija i primenjenih operatora na konačne rezultate. Uočeno je da su bolji rezultati dobijeni ukoliko se nivo mutacije adaptivno menja prateći rast sličnosti jedinki u populaciji. Ukoliko se koristi veći broj manjih potpopulacija moraju se mnogo pažljivije odrediti granice i tok promene nivoa mutacije, jer je tada mnogo veća mogućnost preuranjene konvergencije ili gubitka genetskog materijala. Takođe je uočeno da učestalost razmene jedinki između potpopulacija vrlo bitno utiče na performanse date implementacije.

PGA za rešavanje problema K-podele (eng K-partition problem) je opisan u [Cohoon91] i [Cohoon91a]. Paralelni računar sadrži 16 procesora povezanih u hiperkocku i svaki procesor izvršava GA na potpopulaciji od 80 jedinki. Razmena jedinki se vrši na svakih 50 generacija, uz korišćenje kaznenih funkcija za nekorektne jedinke. Eksperimenti sa korišćenjem kaznene funkcije su pokazali da su najbolji rezultati dobijeni ako je za svaku potpopulaciju kaznena funkcija pomnožena slučajnim brojem $\lambda \in [0, 1]$, umesto fiksnog izbora $\lambda=1$ za sve potpopulacije.

Gordon i Whitley su u radu [Gordon93] izvršili uporednu analizu osam različitih PGA i verzije Goldbergovog SGA ([Goldbe89]) 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 opštijih paralelnih EA implementacija je data u [Mühlen92], i ona je korišćena za rešavanje problema trgovačkog putnika (eng. Traveling Salesman Problem - TSP) i problema podele grafa (eng. graph partitioning problem). Korišćen je globalni model PEA, gde je populacija globalna, a svaka jedinka dodeljena tačno jednom procesoru. Genetski operatori selekcije, ukrštanja i mutacije se vrše isključivo korišćenjem susednih jedinki, bez ikakve globalne kontrole. Na svaku jedinku se zatim primenjuje lokalno pretraživanje čime se kvalitet njenog rešenja dodatno poboljšava. Preuranjena konvergencija se sprečava uz pomoć sporadičnog prosleđivanja dobrih jedinki kroz celu populaciju. Pri rešavanju TSP-a su dobri rezultati postignuti i korišćenjem PGA na 64-procesorskom transpjuterskom sistemu ([Gorges89], [Gorges91]). Globalna populacija sadrži 64 jedinke (svaka jedinka dodeljena jednom procesoru), pri čemu svaka jedinka ima 8 suseda. Prilagođenost pojedinačne jedinke se računa relativno u odnosu na susede, a ne globalno za celu populaciju. Neki potomak je izabran u narednu generaciju, samo u slučaju da je bolji od svog roditelja. Pošto je ostvarena mala međuprocesorska komunikacija, zbog relativno malog broja suseda, ubrzanje izvršavanja EA u odnosu na sekvencijalnu implementaciju je bilo blisko linearnom.

U radu [Fogart91] je primenjen nešto drugačiji globalni PEA, na transpjuterskom sistemu, za rešavanje problema kontrole u realnom vremenu. EA se izvršava na prvom procesoru na populaciji od 250 jedinki, a ostali procesori služe samo za izračunavanje funkcije cilja pojedinačnih jedinki. Pri tome, prvi procesor prosleđuje genetske kodove jedinki ostalim procesorima, a oni izračunavaju njihove vrednosti i prosleđuju ih nazad. Pri paralelnom radu do 20 procesora je postignuto skoro linearno ubrzanje, pošto je računanje vremenski vrlo zahtevne funkcije cilja prebačeno na ostale procesore. Međutim, pri radu većeg broja procesora, rezultati nisu bili zadovoljavajući.

Levine je za rešavanje problema podele skupa (eng. Set Partitioning Problem - SPP) primenio dva modela paralelizacije EA: distribuirani model je korišćen u implementaciji opisanoj u [Levine93] i [Levine93a], a centralizovani model, razvojem PGAPack paralelne biblioteke ([Levine95] i [Levine95a]). U distribuiranom modelu se odabir jedinke, koja će biti izbačena iz populacije da bi se na njeno mesto kopirala najbolja jedinka

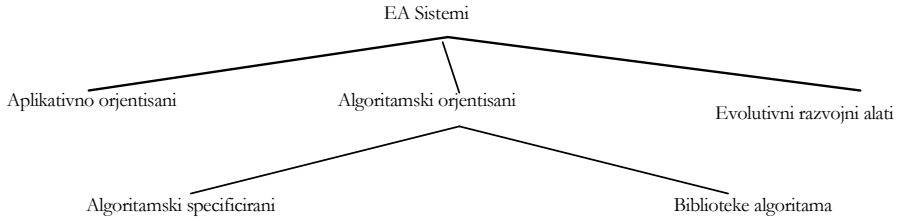
susedne populacije, vrši turnirskom selekcijom (izbacuje se najlošija jedinka na turniru). To se opravdava činjenicom da, u datom slučaju, i najlošija jedinka u populaciji ima neke šanse za izbor u narednu generaciju, pa se time doprinosi raznovrsnosti genetskog materijala. Paralelni EA na računaru IBM SP1 sa 128 procesora je pokazao dobre rezultate, pri čemu su rešavane i neke, vrlo teške, praktične SPP instance sa nekoliko desetina hiljada celobrojnih promenljivih. Centralizovani model je takođe pokazao dobre performanse, a osim izvršavanja na specijalizovanom paralelnom računaru, ostavljena i mogućnost korišćenja mreže radnih stanica.
U [Chen98] se kombinuju paralelni EA i metoda simuliranog kaljenja za rešavanje NP-teškog problema koda za korekciju maksimalnog broja greški u prenosu telekomunikacionih signala (eng. error correcting code that corrects a maximum number of errors). Prikazani su rezultati izvršavanja instanci velike dimenzije na SIMD paralelnom računaru sa od 256 do čak 16384 procesora. Za rešavanje istog problema u [Alba02] se primenjuje Distribuirani EA, a u [Alba04] se isti pristup kombinuje sa jednom metodom lokalnog pretraživanja nazvanom repulzioni algoritam, čija ideja je preuzeta iz elektrostatike. Iako se testiranja vrše samo na 1 i 5 procesora, svaki procesor je Pentium IV na 2.4 GHz sa 512 MB unutrašnje memorije, pa su rezultati uporedivi sa prethodnim metodama.
Analiza i poređenje nekoliko strategija migracije fino gradiranih paralelnih EA je data u [Vinc02], a jedna studija o uticaju frekvencije migracije i broja razmenjenih jedinki na kvalitet rešenja fino usitnjenog EA se može naći u [Alba04a].

1.3. Postojeći EA Sistemi

Tokom višedecenijskog razvoja Evolutivnih algoritama razvijen je veliki broj EA programskih sistema. U ovom poglavlju biće opisani najznačajniji među njima. Po najčešće prihvaćenoj klasifikaciji, svi EA sistemi se dele na sekvencijalne i paralelne EA sisteme.

1.3.1. Sekvencijalni EA sistemi

Jedna od često korišćenih klasifikacija (iz [Ribeir94]), prikazana na slici 7.1, deli EA sisteme na: aplikativno orjentisane, algoritamski orjentisane i evolutivne razvojne alate.



slika 2.5.

U nastavku će biti razmotren svaki od ovih tipova EA sistema.

Aplikativno orjentisani sistemi

Neki korisnici, a takvih je veliki broj, ne žele da programiraju ili uče EA, već samo da korišćenjem gotovih programskih paketa reše problem koji imaju. U takvim sistemima se EA tretira kao crna kutija, a teži se zadovoljavanju specifičnih zahteva korisnika. Dati sistemi imaju grafički dobro osmišljen sistem menija i prezentacije rezultata, obimnu i profesionalno urađenu dokumentaciju i interaktivnu pomoć u svakom delu programa. Uglavnom su komercijalno dostupni kao ekspertni sistemi za podršku u poslovnom okruženju. Neki od najpoznatijih programskih paketa ove klase su: PC/Beagle, XpertRule, GenAsys, Evolver, Omega... Interesantno je napomenuti da Microsoft Analytical Services (raniji OLAP services) koji se isporučuju u okviru Microsoft SQL Servera 2005 imaju podršku za analizu i optimizaciju pomoću evolutivnih (preciznije genetskih) algoritama.
U radu [Riberi94a] je dat njihov kratak pregled, ali i neke informacije o ostalim vrstama programskih paketa povezanim sa EA.

Algoritamski orjentisani sistemi

Naprednim korisnicima i programerima uglavnom više odgovaraju algoritamski orjentisani sistemi, koji mogu biti pojedinačne EA implementacije ili skup programskih EA biblioteka za razvoj, koje uz modifikovane GA mogu obuhvatiti i ES, odnosno EP ili GP. Oni često obuhvataju izvorni kod i mogu se lako uključivati u druge aplikacije.

Oni su najčešće:

- razvijani na univerzitetima ili istraživačkim razvojnim centrima, u javnom su vlasništvu i dostupni preko Interneta;
- dati u izvornom kodu, što olakšava prilagođavanje specifičnim zahtevima nekog od problema optimizacije;
- modularne strukture, što olakšava zamenu ili nadogradnju određenih delova;
- bez grafičkog korisničkog interfejsa ili uz rudimentaran sistem menija;
- korišćeni za rešavanje konkretnih optimizacionih problema ili za testiranje i analizu određenih genetskih operatora ili algoritama.

Dalje će ukratko biti opisane karakteristike nekoliko najpoznatijih algoritamski orjentisanih sistema.

Genesis

Jedna od najstarijih i najpoznatijih EA implementacija (preciznije GA implementacija) opšte namene je bio Genesis, detaljno opisan u [Grefen84]. Ova implementacija je vrlo dugo korišćena za ispitivanje i testiranje genetskih operatora pri rešavanju praktičnih problema (videti [Grefen86] i [Grefen96]). Implementiran je u programskom jeziku C, prenosiv uz manje izmene i na druge platforme, koji rade pod različitim operativnim sistemima. Data je gotova biblioteka genetskih operatora selekcije, ukrštanja i mutacije, a od korisnika se zahteva samo da definiše funkciju prilagođenosti, pošto ona u potpunosti zavisi od prirode problema koji se rešava. Jedinke se mogu efikasno kodirati binarno, alfanumerički i pomoću realnih brojeva. Korisnik u konfiguracionoj datoteci može zadati sve parametre važne za primenu GA kao što su: način kodiranja, broj gena i veličina svakog od njih, broj ponavljanja eksperimenta, broj jedinki u populaciji, nivo ukrštanja, nivo mutacije i drugo.

GAGA

GAGA (eng. **G**enetic **A**lgorithms for **G**eneral **A**pplication) je bila jedna od prvih GA implementacija razvijenih u Evropi (Pascal verzija: Hillary Adams, University of York; C verzija: John Crowcroft, University College London). Korisnik mora da definiše ciljnu funkciju koja je predmet optimizacije i ostale parametre potrebne za izvršavanje GA. Iako u početku relativno skromnih mogućnosti, data implementacija je postigla dobre rezultate, čak i pri izvršavanju na relativno teškim problemima.

Genitor

Genitor (eng. **Gen**etic **I**mplementa**tor**) je modularno implementiran programski paket u programskom jeziku C, koji dopušta binarnu, celobrojnu i reprezentaciju pomoću realnih brojeva. Forsira se korišćenje selekcije zasnovane na rangu ([Whitle89]) i stacionarnog GA, mada se može izvršavati i drugačije. U stacionarnom GA se u svakoj generaciji generiše samo jedna nova jedinka, i ona zamenjuje najlošiju jedinku u prethodnoj generaciji. Detaljan opis implementacije se može videti u [Whitle88].

GENEsYs

GENEsYs ([Bäck92]) predstavlja proširenje Genesis 4.5 implementacije sledećim elementima:

- m-poziciono [DeJong75], uniformno ukrštanje [Syswer89], diskretna rekombinacija i rekombinacija u srednjem [Bäck91a];
- Adaptivna promena nivoa mutacije [Bäck92a];
- (μ, λ) - selekcija [Bäck91a] i Boltzmann-ova selekcija [Goldbe90];
- Veliki broj višemodalnih test funkcija obmanjivačkog tipa, relativno teških za optimizaciju.

Programski kod implementiran u programskom jeziku C pod UNIX operativnim sistemom.

GAGS

Ona je jedna od najstarijih objektno orjentisanih GA implementacija u javnom vlasništvu i dostupnih preko Interneta ([Merelo94]). Osim standardnih varijanti genetskih operatora, realizovani su i genetski operatori promenljive dužine, gde se može vršiti slučajno dodavanje, brisanje ili kopiranje nekog gena. Dati programski

paket sadrži i sistem menija odnosno grafički prikaz rezultata izvršavanja, uz detaljna uputstva za korišćenje ([Merelo94]) i programiranje ([Merelo94a]).

EM

EM (eng. 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 (eng. 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 [Voigt91] se može videti detaljniji opis date implementacije.

OOGA

OOGA (eng. Object-Oriented Genetic Algorithm) je objektno orjentisana GA implementacija, čiji je autor Lawrence Davis, a prvobitna namena je bila podrška primerima iz njegove knjige [Davis91]. 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.

Evolutivni razvojni alati

Ovi sistemi se mogu po svojim karakteristikama uglavnom podeliti u dve grupe:

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

Školski EA služe za upoznavanje novih korisnika sa raznim delovima EA i učenje. Jednostavni su za korišćenje, a konfiguracija sistema se vrši preko sistema menija. Jedan od prvih primera ovakvog programskog paketa je GA Workbench ([Hughes89]).

Sistemi EA opšte namene se koriste od strane zahtevnih korisnika, koji na lak način žele da samostalno razvijaju sopstvene EA aplikacije. Ovakvi sistemi moraju imati: grafičko okruženje i pregledan sistem menija, biblioteku gotovih algoritama, sopstveni jezik visokog nivoa za kombinovanje evolutivnih komponenti i otvorenu arhitekturu za dalja proširenja. Neki od prvih EA sistema opšte namene su bili Splicer [Bayer91] koji uvodi pojam razmenljivih biblioteka i MicroGA [MGA92]. Zbog velikog interesovanja za razne praktične primene EA, kasnije je razvijeno dosta specijalizovanih EA sistema.

1.3.2. Paralelni EA sistemi

Kao što je već rečeno, paralelizacija EA se najčešće odvija po globalnom (centralizovanom), grubo usitnjenom (distribuiranom, odnosno "modelu ostrva"), fino usitnjenom ili hibridnom modelu.

U radu [Hamil02] se pokazuje da su paralelni EA prilično otporni na greške, da uz male izmene, mogu uspešno biti korišćeni i u mrežama sa velikom količinom šuma. Sa druge strane, EA istraživači su uočili da za neke od paralelnih EA projektovanih za homogene paralelne računare, gde su svi procesori istih performansi, može doći do drastičnog pogoršanja performansi ukoliko se primenjuju na nehomogene paralelne računare (mreže radnih stanica). U [Branke04] je utvrđeno da dosta često, u većoj ili manjoj meri, dolazi do date anomalije, pa se predlažu razne migracione šeme koje uspešno rešavaju takve slučajeve. Performanse nekih od predloženih operatora na nehomogenim paralelnim računarima skoro da dostižu performanse uobičajenih operatora migracije na homogenim paralelnim sistemima.

Genitor II

Algoritamski orjentisan sistem Genitor je tokom vremena nadograđen - implementirana je i distribuirana verzija date implementacije pod nazivom Genitor II, čiji se opis može naći u [Whitle90].

NBPGA

U radu [Kojima02] je predložen mrežno zasnovani paralelni GA (network based parallel GA - NBPGA), implementiran u programskom jeziku Java, koji se izvršava na mreži radnih stanica (jednoprocesorskih PC računara) i koristi klijent-server model komunikacije. Pošto u ovom distribuiranom (ostrvskom) modelu paralelnog GA i klijent-server modelu komunikacije može doći do zagušenja usled prevelikog broja podataka koji se prenose, u datom radu su predložene neke metode za smanjenje ostvarene komunikacije preko mreže, bez negativnih uticaja na konačne rezultate GA. Takođe su prikazani i rezultati ovog pristupa pri rešavanju problema trgovačkog putnika i nekih jednomodalnih i višemodalnih funkcija.

EA Java apleti i distribuisano izvršavanje algoritma

U radovima [Chong98] i [Chong99] opisano je EA okruženje realizovano na programskom jeziku Java, gde su Java apleti iskorišćeni da se na njima izvršavaju potpopulacije a centralni server se koristi radi agregacije rezulata i migracije jedinki iz raznih potpopulacija. Izvršeno je nekoliko eksperimenata u kojima su rešavani problemi iz GP domena, pri čemu su čvorovi izračuvanja bili računari dobrovoljaca na Internetu, koji su učitali prethodno pomenuti aplet. U toj implementaciji paralelna arhitektura je zvezda, pa u slučajevima velikog broja potpopulacija server (tj. agregacija i migracija) postaje potencijalno usko grlo.

2. ANALIZA KONVERGENCIJE EVOLUTIVNIH ALGORITAMA

Teorijsko razmatranje EA najčešće polazi od Holland-ovog doprinosa u ovoj oblasti, tj. od njegovih rezultata. I pre Holland-ovog rada su postojali evolutivno inspirisani programski sistemi, ali tek njegov rad daje određeni uvid u to kako ova klasa algoritama dovodi do rešenja, i zašto su EA toliko efikasni pri rešavanju određenih klasa problema.
U poglavljima koja slede uvode se veličine koje daju odgovore na sledeća pitanja:

- Na koji se deo okruženja sistem adaptira, tj. kakvo je okruženje u kojem se pomoću EA traži najbolje rešenje?
- Kako okruženje deluje na organizme, tj. kako se određuje koliko se uspešno jedinka prilagodila okruženju?
- Koje su strukture ispod adaptacije, tj. kako se reprezentuje genetski kod jedinki populacije koje se prilagođavaju okruženju?
- Kakvi su mehanizmi adaptacije, tj. kako deluju operatori EA?
- Koji se deo istorije interakcije organizma i okruženja čuva radi dodavanja na testirane strukture, tj. koliko se pamte (i u proceni prilagođenosti jedinki koriste) prilagođenosti jedinki prošlih populacija?
- Koja ograničenja postoje za adaptivni proces, tj. koje familije EA su nam od interesa?
- Kako se različiti adaptivni procesi mogu porediti, tj. koji su kriterijumi za poređenje EA?

2.1. Teorema o shemama

U razmatranju koje sledi, pretpostavlja se da su niske reči formirane nad dvočlanim alfabetom $\{0,1\}$ i da je njihova dužina l. Dakle,

Definicija 2.1.1. Niska, u oznaci a, je reč dužine l nad dvočlanim alfabetom $\{0,1\}$.

Pojam, tj. koncept sheme predstavlja osnovu ovog razmatranja. Naime, umesto koncentrisanja na pojedine niske, koncentriše se na sličnosti između njih, tj. na pitanje koliko jedna niska može biti bliska nekoj drugoj niski. šablon sličnosti, koji opisuje podskup niski istih na određenim pozicijama, naziva se shema.

Definicija 2.1.2. Shema, u oznaci H, je reč dužine l nad tročlanim alfabetom $\{0,1,*\}$. Simbol * je džoker simbol (može da zameni *0* ili *1*) - on je metasimbol za EA, tj. on se eksplicitno ne obrađuje genetskim algoritmom. Pozicije koje zauzimaju simboli *0* i *1* nazivaju se fiksirane pozicije sheme.

Definicija 2.1.3. Shema odgovara određenoj niski ukoliko na svakoj lokaciji u shemi simbolu *1* odgovara *1* iz niske, simbolu *0* u shemi odgovara *0* iz niske, a simbol * se zamenjuje jednim od simbola *0* ili *1*.

Dakle, shema daje moćan i kompaktan put za razmatranje svih dobro definisanih sličnosti među niskama konačne dužine nad konačnim alfabetom. Treba napomenuti da pored ovakvog, postoje i neki drugi načini definisanja shema. U njima se shema definiše bilo kao drvo izvođenja kod konteksno slobodnog izvođenja ([Whigha96]), bilo kao predikat ([Vose91]), bilo preko ravni i hiperravni u višedimenzionalnom bitovnom prostoru ([Goldbe89]). Ti drugačiji pogledi u nekim slučajevima daju bolju i precizniju sliku, odnosno jasniji pogled, na problematiku koja se proučava.

Lako se može utvrditi da postoji 2^l različitih niski dužine l i 3^l različitih shema iste dužine.

Uvođenjem simbola * proučava se veći broj reči nego kada nema ovog simbola. Ta novouvedena redundansa je od koristi jer kada bi se posmatrale samo pojedinačne niske, posmatrali bi se samo delovi informacije. Potpuna informacija, informacija koja omogućava praćenje načina izvršavanja, tj. načina traženja rešenja kod EA, zahteva poznavanje sličnosti među niskama.

Pitanje broja informacija koje se posmatraju razmatranjem sličnosti se svodi na pitanje broja različitih shema koje sadrži populacija.

Odgovor na ovo pitanje zahteva određivanje broja jedinki koje se sadrže u jednoj niski. Korišćenjem nekih od osnovnih tvrđenja kombinatorike, jednostavno se pokazuje da niska dužine l sadrži 2^l različitih shema, tj. postoji 2^l različitih shema koje odgovaraju fiksiranoj niski dužine l.

Donja i gornja granica broja shema koje se sadrže u populaciji od n jedinki se sada trivijalno izvodi, pa populacija od n jedinki (članova) sadrži ne manje od 2^l i ne više od $n2^l$ shema.

Po utvrđivanju obima informacija koje koristi EA, postavlja se pitanje koliko efikasno EA eksploatiše ove informacije. Drugim rečima, utvrđena je gornja i donja granica za broj shema koje se sadrže u populaciji, pa se postavlja pitanje koliko shema među postojećim EA obrađuje na koristan način. Dakle, treba razmotriti efekte genetskih operatora selekcije, ukrštanja i mutacije na širenje i opadanje pojedinih shema iz generacije u generaciju.

Što se efekta selekcije tiče, niske sa najvećom prilagođenošću preživljavaju sa većom verovatnoćom, odnosno sa većom verovatnoćom prelaze u sledeću generaciju. Zato će prosečan broj shema koje se odgovaraju najboljim niskama tokom vremena biti sve veći. Sama selekcija ne uključuje nove uzorke u prostor niski, tj. primenom ovog genetskog operatora ne kreira se nov genetski materijal.

Dejstvo operatora ukrštanja na shemu je nešto složenije. Naime, ukrštanje će ostaviti shemu nepromenjenom ukoliko je ne „raseče", a ukoliko dođe do „rasecanja" tokom ukrštanja shema će biti razorena. Pri tome ne bivaju sve sheme "rasečene" sa istom verovatnoćom.

Primer. Uočimo sheme dužine pet: *1***1* i ***01**. Prva od njih dve bi lakše mogla biti razorena ukrštanjem, dok bi se to za drugu relativno teže desilo (naravno, uz pretpostavku da su sve pozicije pri izboru tačke ukrštanja jednako verovatne). □

Dakle, neke će sheme sa manjom, a neke sa većom verovatnoćom biti razorene prilikom ukrštanja, tj. ukrštanje kod prostog EA olakšava širenje nekih shema, a razara neke druge.

Operator mutacija se kog EA obično primenjuje sa malom verovatnoćom. Ipak, i ovaj operator utiče na broj shema u populaciji, jer se njegovom primenom može (ali ne mora) razoriti neka fiksirana shema.

Već se vidi da, iako EA manipulišu direktno i eksplicitno sa niskama u populaciji, izvršavanje EA prouzrokuje implicitno procesiranje mnogih shema tokom svake od generacija ([Hollan75], [Goldbe89]).

Da bi se strože odredila verovatnoća preživljavanja i verovatnoća uništavanja sheme pod dejstvom genetskih operatora, potrebno je definisati veličine koje karakterišu otpornost sheme na uništavanje:

Definicija 2.1.4. Red (eng. order) sheme H, u oznaci $o(H)$, je broj fiksnih pozicija, tj. broj nula i jedinica koje su prisutne u šablonu.

Definicija 2.1.5. Definišuća dužina (eng. defining length) sheme H, u oznaci $\delta(H)$, je rastojanje između prve i poslednje specificirane pozicije u niski, tj. rastojanje između prve i poslednje pozicije u niski na kojoj se ne nalazi simbol *.

Primer. Za fiksirane sheme *011*1**** i *0******* važe sledeće jednakosti:

$$o(011*1***)=4,\ o(0*******)=1,\ \delta(011*1***)=5-1=4,\ \delta(0*******)=1-1=0\ \square$$

Sada je moguće preciznije odrediti efekat jednog koraka tj. jedne iteracije prostog genetskog algoritma na očekivani broj shema u populaciji.

Dejstvo operatora proporcionalne selekcije na broj pojava neke fiksirane sheme u populaciji opisano je sledećim stavom:

Stav 2.1.1. Neka je $m(H,t)$ broj koji označava koliko se puta neka shema H sadrži u $a(t)$ tj. u zadatoj populaciji u vremenskom trenutku t. Neka je sa $f(H)$ označena srednja vrednost prilagođenosti svih niski populacije kojima odgovara shema H, a sa $\overline{f}$ srednja vrednost prilagođenosti svih niski u populaciji. Tada je (ukoliko se radi o operatoru proporcionalne, tj. rulet selekcije):

$$m(H,t+1)=m(H,t)\frac{f(H)}{\overline{f}}.$$

Dokaz: Neka je a_k oznaka fiksirane niske u populaciji i neka je prilagođenost niske a_k označena sa f_k. Tada je, budući da se radi o proporcionalnoj selekciji (videti i poglavlje 2.1.1), verovatnoća izbora tj. prelaska u sledeću generaciju za jedinku a_k data sa

$$p_s(a_k) = \frac{f_k}{\sum_{j=1}^{n} f_j} = \frac{f_k}{n\overline{f}}.$$

Iz ove jednakosti direktno (sumiranjem po svim jedinkama koje odgovaraju datoj shemi *H*) sledi tvrđenje stava. ∎

Dakle, brzina širenja određene sheme je jednaka količniku između prosečne prilagođenosti sheme i prosečne prilagođenosti populacije. Prema tome, sheme sa natprosečno dobrom prilagođenošću se šire, dok sheme sa ispodprosečnom prilagođenošću odumiru. Treba napomenuti da bi se sličan, mada malo manje pregledan, a samim tim i manje upotrebljiv, rezultat dobio i kad bi umesto operatora rulet selekcije bio neki drugi operator selekcije.

Stav 2.1.2. Neka određena shema *H* ostaje natprosečna sa faktorom *c*, tj. prosečna prilagođenost sheme *H* iznosi $\overline{f} + c\overline{f}$, pri čemu se pretpostavlja da je $\overline{f}$ konstanta, tj. da se tokom vremena prosečna ocena populacije ne menja. Tada je $m(H,t) = m(H,0)(1+c)^t$

Dokaz: Na osnovu prethodnog stava je:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\overline{f}} = m(H,t)\frac{\overline{f} + c\overline{f}}{\overline{f}} = m(H,t)(1+c)$$

Iz ove jednakosti, primenom principa matematičke indukcije, direktno sledi tvrđenje stava. ∎

Što se ukrštanja tiče, ono se posmatra kao struktuirana, ali i probabilistička razmena informacija među niskama. Ukrštanje kreira nove strukture, uz minimalno razaranje strategije alokacije shema, koje diktira sama selekcija. Sledeći stav opisuje dejstvo jednopozicionog ukrštanja na verovatnoću preživljavanja fiksirane sheme u populaciji.

Stav 2.1.3. Neka je pri jednopozicionom ukrštanju tačka ukrštanja slučajna promenljiva sa uniformnom raspodelom, i neka je p_c verovatnoća da je uopšte došlo do ukrštanja. Tada je verovatnoća da shema ne bude uništena tokom ukrštanja, u oznaci p_{surv} data sledećom formulom:

$$p_{surv} \geq 1 - p_c \frac{\delta(H)}{l-1}.$$

Dokaz: Verovatnoća da shema bude uništena je verovatnoća da dođe do ukrštanja i da slučajno izabrana tačka ukrštanja bude između prve i poslednje specificirane pozicije u shemi. Ova dva događaja su nezavisna. Verovatnoća prvog je p_c, dok je verovatnoća drugog $\delta(H)/l-1$ (količnik broja povoljnih mogućnosti i ukupnog broja mogućnosti). Tvrđenje stava proizilazi iz prethodne dve jednakosti primenom odgovarajućih tvrđenja teorije verovatnoće. ∎

Sledeći stav daje verovatnoću da shema bude prekinuta, odnosno da ne bude prekinuta, usled proste mutacije.

Stav 2.1.4. Neka je verovatnoća mutacije označena sa p_m. Tada je verovatnoća da shema preživi tokom mutacije data izrazom $p_{surv} = (1-p_m)^{o(H)}$.

Dokaz: Tvrđenje stava proizilazi iz činjenice da je verovatnoća da shema preživi mutaciju jednaka verovatnoći da ne bude zamenjen nijedan od simbola na pozicijama koje su fiksirane u shemi. Verovatnoća za jednu poziciju iznosi $1-p_m$, a pozicija ima $o(H)$. ∎

Napomena. Ako je $p_m << 1$ (što je realna situacija u primenama), tada je $(1-p_m)^{o(H)} \approx 1 - p_m o(H)$, pa se u daljim aproksimacijama koristi desni izraz. □

Sledeći stav, nazvan Teorema o shemama, ima veliki značaj u proučavanju EA (videti [Hollan75], [Goldbe89]):

Stav 2.1.5. $m(H,t+1) \geq m(H,t)\frac{f(H)}{\overline{f}}\left(1 - p_c\frac{\delta(H)}{l-1} - p_m o(H)\right)$

Dokaz: Na populaciju se sukcesivno deluje ukrštanjem, pa mutacijom i na kraju selekcijom. Verovatnoća da shema ne bude uništena tokom ukrštanja i mutacije je, stoga, verovatnoća komplementa događaja da shema bude prekinuta. Verovatnoća prekida je manja od verovatnoće da se dogodi ukrštanje pri čemu se tačka ukrštanja našla između dve fiksirane pozicije sheme ili da se dogodi mutacija pri čemu je pozicija koja se

komplementira neka od fiksiranih pozicija sheme. Dakle, verovatnoća prekida sheme je, na osnovu stava 2.1.4. i stava 2.1.3. odozdo ograničena izrazom

$$p_c \frac{\delta(H)}{l-1} + p_m o(H).$$

Prema tome, verovatnoća da shema preživi ukrštanje i mutaciju se može minorirati izrazom:

$$1 - p_c \frac{\delta(H)}{l-1} - p_m o(H).$$

Tvrđenje stava se dobija uklapanjem ovog rezultata i stava 2.1.1. ∎

Na osnovu teoreme o shemama može se zaključiti da broj pojava fiksirane sheme natprosečne prilagođenosti, kratke dužine i niskog reda, iz generacije u generaciju eksponencijalno raste.

2.2. Teorema o implicitnom paralelizmu

Postavlja se pitanje procene donje granice broja uspešno obrađenih shema. Populacija se sastoji od *n* binarnih niski dužine *l*. Razmatraju se samo sheme koje preživljavaju sa verovatnoćom većom od p_s. Dakle, prihvataju se samo one sheme čiji je nivo greške manji od ε, pri čemu je $\varepsilon < 1 - p_s$. Ovo dovodi do razmatranja samo onih shema čija je definišuća dužina manja od l_s, pri čemu je $l_s < \varepsilon(l-1)+1$.

Neka bude fiksirana definišuća dužina sheme - označimo tu dužinu sa ls. Da bi se ograničio broj shema koje obrađuje inicijalna slučajna populacija i čija je definišuća dužina manja od ls, potrebno je odrediti koliko se najmanje shema mora sadržati u jednoj niski.

Primer. Pretpostavimo da treba prebrojati sheme definišuće dužine manje od $l_s = 5$ u fiksiranoj niski dužine $l = 10$, i da je niska oblika *1011100010*. Postupak prebrojavanja je sledeći: Prvo se izračuna broj shema u podvučenom delu dužine *5*, tj. u niski *1011100010*, uz pretpostavku da je poslednji bit fiksiran. To je u ovom slučaju broj shema u formi *%%%%1*****, pri čemu simboli *%* označava vrednost koja može stajati na poziciji koja nije fiksirana - broj takvih shema iznosi $2^{5-1} = 16$. Potom se pet mesta od interesa klizno pomere za jednu poziciju udesno, kako bi se računao broj shema u niski *1011100010* - opet se radi o broju $2^{5-1} = 16$. Kada se izračuna broj ovih shema, vrši se sledeće klizno pomeranje udesno, itd. dok se ne dođe do kraja niske. Pomeranja za po jedno mesto udesno u ovom primeru ima *10-5+1 = 6*. □

Dakle, lako se može pokazati da važi sledeći stav:

Stav 2.2.1. Neka je data niska dužine *l*. Broj shema definišuće dužine manje od l_s u takvoj niski se može odozdo ograničiti izrazom $2^{l_s-1}(l-l_s+1)$.

Dokaz: Uopštavanjem izvođenja iz prethodnog primera (u prethodnom primeru se svugde stavi umesto broja *5* veličina l_s, a umesto broja *10* veličina *l*). ∎

Stav 2.2.2. Neka je slučajno odabrana inicijalna populacija od *n* niski dužine *l*. Broj shema definišuće dužine l_s u takvoj niski je reda $n2^{l_s-1}(l-l_s+1)$.

Dokaz: Izvodi se direktnom primenom prethodnog stava. U najboljem slučaju, ako se sve sheme sadržane u različitim jedinkama populacije međusobno razlikuju (lako je uvideti da taj slučaj i nije moguć), broj shema je odozgo ograničen sa $n2^{l_s-1}(l-l_s+1)$. ∎

Napomena. Može se primetiti da broj shema ima binarnu distribuciju, pa je broj shema čiji je red manji od $\frac{l_s}{2}$ jednak broju shema reda većeg od $\frac{l_s}{2}$. Ukoliko se broje samo sheme reda većeg od $\frac{l_s}{2}$, donje ograničenje njihovog broja u populaciji dato je jednakošću $n_s = n2^{l_s-2}(l-l_s+1)$. □

Nadalje, uvodi se pretpostavka da je broj jedinki u populaciji $n = 2^{\frac{l_s}{2}}$. Ova pretpostavka unekoliko umanjuje opštost daljeg razmatranja, ali omogućuje da rezultati budu iskazani u preglednijoj formi:

Stav 2.2.3. Neka je broj jedinki u populaciji $n = 2^{\frac{l_s}{2}}$, pri čemu je sa l_s označena maksimalna definišuća dužina sheme. Broj shema koje obrađuje (tj. sadrži) populacija je:

$$n_s = \frac{n^3 (l - l_s + 1)}{4}.$$

Drugim rečima, broj shema koje u datom trenutku bivaju obrađene od strane EA je proporcionalan kubu veličine populacije.

Dokaz: Tvrđenje stava direktno sledi zamenom vrednosti za *n* sa $2^{\frac{l_s}{2}}$ u tvrđenje iskazano u napomeni iza stava 2.3.2.■

Tvrđenje poslednjeg stava, zbog svog značaja, u literaturi ima svoje ime: Teorema o implicitnom paralelizmu ([Goldbe89]).

Paradigma EA, dakle, nudi veliki unutrašnji paralelizam pri pretrazi kroz veliki pretraživački prostor. Jednostavnije rečeno, bez obzira što se obrađuje samo n jedinki, EA će obraditi bar n^3 shema.

Treba napomenuti da je ova procena u kasnijim radovima donekle profinjena, a istovremeno i uopštena, tj da je određena viša donja granica za broj shema koje se sadrže u populaciji ([Berton93]). Kako je analitički izraz za funkciju koja predstavlja donju granicu kod nove procene duži i nepregledniji od procene iz stava 2.3.3, a već iskazani zaključak je isti, u literaturi se za procenu broja shema koje se sadrže u populaciji i nadalje češće koristi procena utvrđena stavom 2.3.3, tj. teoremom o implicitnom paralelizmu. Uticaj ukrštanja na sheme i na implicitni paralelizam je razmatran i u radu [Mason93].

2.3. Hipoteza o gradivnim blokovima

Pregled performansi EA biva mnogo jasniji ukoliko se EA posmatraju iz perspektive shema. Hipoteza o gradivnim blokovima pretpostavlja da EA optimalna ili suboptimalna rešenja dobijaju rekombinovanjem gradivnih blokova. Gradivni blokovi su sheme sa dobrim performansama, malim redom i malom definišućom dužinom. Prema hipotezi gradivnih blokova, takve sheme (tj. gradivni blokovi) se sempliraju, rekombinuju, ponovo sempliraju, ponovo rekombinuju, itd. kako bi oformili niske sa potencijalno većom prilagođenošću. Dakle, hipoteza o gradivnim blokovima tvrdi da se pri izvršavanju EA, niske visokih performansi ne formiraju pokušavajući sa svakom shvatljivom kombinacijom, već se sve bolje i bolje niske grade od najboljih parcijalnih rešenja u prošlim sempliranjima.

EA-obmanjivački problemi i epistaza

Hipoteza o gradivnim blokovima doživljava novo prevrednovanje. Ova hipoteza se od strane nekih autora kritikuje zbog statičnosti ([Grefnen89]), dok drugi revalorizuju njene postavke (videti [Goldbe89], [Davis91]).

U bliskoj vezi sa hipotezom o gradivnim blokovima su problemi kod kojih prosti EA ne nalazi optimalno (ni suboptimalno) rešenje. Takvi problemi su nazvani EA-obmanjivački (eng. EA-deceptive) i ova vrsta problema predstavlja važan objekt proučavanja među autorima koji se bave evolutivnim programiranjem, odnosno genetskim algoritmima ([Kalyan92], [Grefnen92]). Među takvim problemima se izdvaja minimalni problem takav da može da obmane konkretan EA, tj. minimalni EA-obmanjivački problem. Dizajniranje i proučavanje takvih problema daje bolju sliku o karakteristikama EA (videti [Khuri93], [Goldbe89], [Grefen92]).

Primer. Sledeći uslovi određuju dvobitnu minimalnu obmanjivačku funkciju. Takva funkcija predstavlja najprostiji minimalni obmanjivački problem. Uslovi su:

- Vrednost *f(11)* je maksimum funkcije *f*, tj. *f(11)>f(00)* i *f(11)>f(10)* i *f(11)>f(01)*;
- Shema koja na fiksiranoj poziciji sadrži nulu ima bolju prosečnu prilagođenost od sheme koja na toj istoj poziciji sadrži jedinicu, tj. *f(0*)>f(1*)* ili *f(*0)>f(*1)*.

U ovom slučaju, sheme niskog reda *0** i ***0* ne sadrže optimalnu nisku *11*, pa mehanizam pretraživanja vodi EA pored *11*, tj. EA biva obmanut. □

Kako obmanjivanje proizilazi iz statičke analize hiperravni u prostoru bitovnih niski, koja ne vodi računa o mogućem raznovrsnom i dinamičnom ponašanju EA, to analiza EA preko obmanjivačkih problema nužno pravi statičke pretpostavke.

Pojam epistaze (eng. epistasis) je usko povezan sa gradivnim blokovima i obmanjivačkim problemima. Pored temina epistaza, koristi se još i termin nelinearnost (eng. nonlinearity). Epistaza je ma koja vrsta jake interakcije među genima u hromozomu, tj. jake interakcije između različitih bitova niske. Do epistaze dolazi

zato što učešće u vrednosti prilagođenosti za neki gen zavisi od vrednosti drugih gena (videti [Reevs95a]). Problemi sa malom ili nikakvom epistazom su trivijalni za rešavanje - sasvim je dovoljno koristiti tehniku nazvanu penjanje uz brdo (eng. hillclimbing). Ali, problemi sa velikom epistazom su teško rešivi, čak i za EA. Velika epistaza obično znači da se ne mogu oformiti gradivni blokovi, i da je u pitanju obmanjivanje.
O svemu ovom treba voditi računa prilikom izbora standardnih test-problema na kojima će se porediti raznovrsni EA. Takvi problemi biće podrobnije opisani u poglavlju 5.
Matematički okvir za statičku analizu EA-obmanjivačkih problema i epistaze je dat particionim koeficijentima, Walshovim i Haarovim funkcijama ([Goldbe89], [Khuri93], [Hecker96]).

2.4. Analiza evolutivnih algoritama uz pomoć Markovljevih lanaca

Mnogi autori su, u vezi sa prethodno izvedenim rezultatima, naglasili izvesnu statičnost koju nameću polazne pretpostavke, odnosno istakli su da se pri ovakvoj analizi samog EA i njegovih performansi ne uvažava u dovoljnoj meri dinamičnost samog algoritma. Pristup problemu analize preko verovatnosnih matrica, tj. Markovljevih lanaca u mnogo manjoj meri „boluje“ od statičnosti (videti [Horn93], [Reynol96], [Suzuki95]). Ovakav pristup nije karakterističan samo za EA, već i za neke druge srodne meta-algoritme, kao što je simulacija kaljenja, kao i za pojedine discipline mašinskog učenja.

2.4.1. Markovljevi lanci

Markovljevi lanci opisuju stohastičke sisteme „bez memorije“ tj. takve sisteme kod kojih verovatnoće budućih stanja zavise samo od sadašnjeg, a ne od prošlih stanja (videti [Ivkov89]). Izvršavanje Evolutivnih algoritama u svom najvećem delu predstavlja takav stohastički sistem koji se veoma tačno može opisati Markovljevim lancima ([Herber96], [Suzuki97]).

Definicija 2.4.1. Neka je dat fizički sistem ξ u vremenskim trenucima $0,1,2,...$ Stanje sistema u trenutku $n, n \in N_0$ opisano je slučajnom promenljivom X_n diskretnog tipa sa skupom mogućih vrednosti $\{x_1, x_2, ...\}$. Skup mogućih vrednosti slučajne promenljive se još naziva i skup stanja sistema ξ. Evolucija sistema ξ tokom vremena opisana je nizom slučajnih promenljivih $X_0, X_1, X_2, ...$ Ovaj niz slučajnih promenljivih čini lanac Markova ako za svaku kolekciju vremenskih trenutaka $0 \le n_r < n_{r-1} < n_2 < n_1 \le m$ i za svaku kolekciju stanja $\{x_{i_r}, x_{i_{r-1}}, ..., x_{i_2}, x_{i_1}, x_j\}$ važi

$$P\left(\{X_m = x_j\} \middle| \{X_{n_1} = x_{i_1}\} \cap \{X_{n_2} = x_{i_2}\} \cap \cdots \cap \{X_{n_r} = x_{i_r}\}\right) = P\left(\{X_m = x_j\} \middle| \{X_{n_1} = x_{i_1}\}\right)$$

Dakle, ako trenutak n_1 nazovemo sadašnjost, onda lanac Markova označava da su budućnost i prošlost, kada se posmatraju uslovno u odnosu na sadašnjost, nezavisne.

Definicija 2.4.2. Verovatnoća $p_{ij}(n,m) = P\left(\{X_m = x_j\} \middle| \{X_n = x_i\}\right)$ za $0 \le n < m$ je verovatnoća prelaska iz stanja x_i u trenutku n u stanje x_j u trenutku m.

Definicija 2.4.3. Ako verovatnoća prelaska $p_{ij}(n,m)$ kao funkcija od n i m, zavisi samo od razlike $n-m$, tada je Markovljev lanac homogen.

Primer. Neka X_n predstavlja broj automobila koji prođu kroz sekciju puta tokom n minuta, počev od nekog trenutka. Tada niz slučajnih promenljivih $X_0, X_1, X_2, \ldots, X_n, \ldots$ čini Markovljev lanac. Međutim, taj Markovljev lanac nije homogen, zato što $p_{ij}(n,m)$ ne zavisi samo od razlike $n-m$, već i od vrednosti n (odnosno m), tj. od doba dana kada se vrši merenje. Tako se, na primer, verovatnoća prelaska iz stanja 100 u stanje 102 može razlikovati od verovatnoće prelaska iz stanja 120 u stanje 122, iako je u oba slučaja $n-m=2$. □
Prema tome, homogenost Markovljevog lanca ukazuje na nepromenljivost verovatnoća prelaska tokom vremena.

Primer. Posmatrajmo slučajno pomeranje čestice, koja se nalazi u jednom od položaja $\ldots, -k, \ldots, -2, -1, 0, 1, 2, \ldots, k, \ldots$ i koja se sa verovatnoćom p pomera za jedno mesto udesno, a sa

verovatnoćom $q = 1 - p$ se pomera jedno mesto ulevo. Neka je na početku posmatranja sistema čestica na poziciji 0 i neka X_n predstavlja poziciju čestice u trenutku n (posle n pomeranja). Niz slučajnih promenljivih $X_0, X_1, X_2, \ldots, X_n, \ldots$ čini Markovljev lanac. Lako je utvrditi da je taj Markovljev lanac homogen□

U daljem izvođenju će se isključivo posmatrati homogeni Markovljevi lanci (videti [Ivkov89]).

Neka je $p_{ij}(n) = P(\{X_{k+n} = x_j\} | \{X_k = x_i\}), n \in N$. S obzirom da se radi o homogenim lancima, to prethodno definisana verovatnosna funkcija prelaska ne zavisi od k, već se radi o verovatnoći prelaska iz stanja x_i u stanje x_j u n vremenskih jedinica (ili u n koraka). Neka je nadalje $p_{ij} = p_{ij}(1)$.

Stav 2.4.1. Verovatnoća prelaska u n koraka homogenog Markovljevog lanca $p_{ij}(n)$ se može izraziti preko verovatnoća prelaska u jednom koraku p_{ij}:

$$p_{ij}(n) = \sum_k p_{ik}(m) p_{kj}(n-m)$$

gde je $1 \le m < n$ i gde se sumiranje vrši po svim mogućim indeksima stanja $x_1, x_2, \ldots, x_n, \ldots$.

Dokaz: Izvodi se pomoću formule totalne verovatnoće. Ako se postavi da je A događaj da sistem iz stanja x_i pređe u stanje x_j u n koraka, a da su $B_k, k \in N$ događaji da sistem iz stanja x_i pređe u stanje x_k u m koraka ($m < n$) u homogenom Markovljevom lancu, tada važi: $\sum_k B_k = \Omega$, $P(A) = p_{ij}(n)$, $P(B_k) = p_{ik}(m)$ i $P(A \mid B_k) = p_{kj}(n-m)$. Tvrđenje stava se direktno dobija uvrštavanjem prethodnih jednakosti u formulu totalne verovatnoće■

Jednakost iz prethodnog stava naziva se jednakost Čepmen-Kolmogorova.

Definicija 2.4.4. Matrica prelaska homogenog Markovljevog lanca u n koraka je

$$P_n = \begin{bmatrix} p_{11}(n) & p_{12}(n) & \cdots \\ p_{21}(n) & p_{22}(n) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

Matrica prelaska homogenog Markovljevog lanca u jednom koraku

$$P_1 = P = \begin{bmatrix} p_{11} & p_{12} & \cdots \\ p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

Lako se uočava da je svaki elemenat ovih matrica prelaska broj između 0 i 1, a da je suma vrednosti u svakoj vrsti matrice tačno 1 (jer izraz $\sum_j p_{ij}(n)$ predstavlja verovatnoću da sistem izvrši prelazak iz stanja x_i u neko (bilo koje) stanje tj. verovatnoću sigurnog događaja).

Definicija 2.4.5. Matrica $P = \{p_{ij}\}_{ij}$ za čije elemente važi da $(\forall i, j) p_{ij} \in [0,1]$ i da je $(\forall i) \sum_j p_{ij}(n) = 1$ se naziva stohastička matrica.

Napomena. Jednostavno se pokazuje da je proizvod stohastičkih matrica takođe stohastička matrica□

Primer. Slučajno pomeranje čestice iz prethodnog primera, uz pretpostavku da ima konačan broj pozicija s i da su pozicije 1 i s apsorbirajuće, dovodi do sledeće matrice prelaska u jednom koraku:

$$P=\begin{bmatrix} q & p & 0 & 0 & & 0 & 0 & 0 & 0 \\ q & 0 & p & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & q & 0 & p & & 0 & 0 & 0 & 0 \\ & \cdots & & & \cdots & & \cdots & & \\ 0 & 0 & 0 & 0 & & q & 0 & p & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & q & 0 & p \\ 0 & 0 & 0 & 0 & & 0 & 0 & q & p \end{bmatrix}_{s\times s}. \quad \square$$

Stav 2.4.2. Za matricu prelaska $P_n, n=2,3,4,\ldots$ važi: $P_n = P_m P_{n-m}$, gde je $1 \le m < n$.

Dokaz: Direktno iz jednačine Čepmen-Kolmogorova. ∎

Stav 2.4.3. Za matricu prelaska $P_n, n=2,3,4,\ldots$ važi: $P_n = P_1^n = P^n$.

Dokaz: Direktno iz prethodnog stava. ∎

Primer. Niz diskretnih nezavisnih slučajnih promenljivih $\{X_k\}_k$ sa istom raspodelom $P(\{X_k = x_i\}) = p_i$ može se posmatrati kao specijalan slučaj Markovljevog lanca, gde je:

$$p_{ij}(n) = P(\{X_{k+n} = x_j\} \mid \{X_k = x_i\}) = P(\{X_{k+n} = x_j\}) = p_j, \text{ za svako } i \in N.$$

Stoga su sve vrste matrice prelaska u n koraka potpuno iste, tj.

$$P_n = \begin{bmatrix} p_1 & p_2 & p_3 & \cdots \\ p_1 & p_2 & p_3 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ p_1 & p_2 & p_3 & \cdots \end{bmatrix}. \quad \square$$

Definicija 2.4.6. Raspodela verovatnoća stanja sistema ξ u nekom trenutku n, $n \in N_0$ je niz $\{p_i(n)\}_i$, gde je $p_i(n) = P(\{X_n = x_i\})$.

Stav 2.4.4. Neka je raspodela verovatnoća stanja sistema u početnom trenutku $\{p_i(0)\}_i$. Neka je verovatnoća prelaska data matricom $\{p_{ij}\}_{ij}$. Tada je raspodela verovatnoća stanja sistema $\{p_i(n)\}_i$ data sa $p_i(n) = \sum_k p_k(n-1)p_{ki}, n \in N$.

Dokaz: Neposredno korišćenjem formule totalne verovatnoće, gde je $A_i = \{X_n = x_i\}$, $B_k = \{X_{n-1} = x_k\}$, $P(A_i \mid B_k) = p_{ki}$. ∎

Napomena. Lako se utvrđuje da je homogeni Markovljev lanac potpuno određen uređenim parom $(\{p_i(0)\}_i, \{p_{ij}\}_{ij})$ koji obrazuju početno stanje sistema i matrica prelaska.

Definicija 2.4.7. Ako je raspodela verovatnoća ista na svakom koraku, tj. $p_i(n) = p_i$, tada je homogeni Markovljev lanac stacionaran. U tom slučaju se verovatnoće p_i nazivaju stacionarnim verovatnoćama.

Stav 2.4.5. Ako verovatnoće početnih stanja $\{p_i\}_i$ zadovoljavaju homogene jednačine $p_i = \sum_k p_k p_{ki}, i \in N$, onda je Markovljev lanac stacionaran.

Dokaz: Sledi iz rekurentne formule (strogo se može pokazati matematičkom indukcijom, a ovde je urađena baza indukcije, koja jasno ukazuje na pravac izvođenja).

Kada je $n = 1$, tada je

$$p_i(1) = \sum_k p_k p_{ki} = p_1 p_{1i} + p_2 p_{2i} + \cdots = p_i, \text{ za } i \in N$$

Kada je $n = 2$, tada

$$p_i(2) = \sum_k p_k(1) p_{ki} = \sum_k p_k p_{ki} = p_1 p_{1i} + p_2 p_{2i} + \cdots = p_i, \text{ za } i \in N \blacksquare$$

Definicija 2.4.8. Neka je $q_{ii}(n)$ verovatnoća da Markovljev lanac iz stanja x_i prelazi u isto stanje x_i za tačno n koraka.

Stav 2.4.6. Neka je Q_i verovatnoća da se Markovljev lanac u toku evolucije sistema bar jednom vrati u stanje x_i, ako je u nekom trenutku već bio u stanju x_i. Tada je $Q_i = \sum_{n=1}^{\infty} q_{ii}(n)$.

Dokaz: Sledi direktno iz prethodne definicije$\blacksquare$

Definicija 2.4.9. Stanje x_i je povratno ako je $Q_i = 1$, a nepovratno ako je $Q_i < 1$.

Stav 2.4.7. Stanje x_i je povratno ili nepovratno u zavisnosti od toga da li red $\sum_{n=1}^{\infty} p_{ii}(n)$ divergira ili konvergira.

Dokaz: Na osnovu teoreme o totalnoj verovatnoći je $p_{ij}(n) = \sum_{m=1}^{n} q_{ij}(m) p_{jj}(n-m)$, $n = 1, 2, \ldots$ ($p_{jj}(0) = 1$), pa se tvrđenje stava dobija oformljenjem parcijalne sume reda i njegovim ograniča vanjem odozdo i odozgo, pa puštanjem da broj sabiraka parcijalne sume teži ka beskonačnosti$\blacksquare$

Stav 2.4.8. Ako se Markovljev lanac u nekom koraku nađe u povratnom stanju, onda se za beskonačan broj koraka sa verovatnoćom 1 beskonačno puta vraća u to stanje. Ako se Markovljev lanac nađe u nepovratnom stanju, onda se za beskonačno mnogo koraka najviše konačno mnogo puta može sa verovatnoćom 1 da se ponovo nađe u tom stanju (alternativna formulacija je da se posle konačno mnogo koraka više nijednom sa verovatnoćom 1 ne vraća u to stanje).

Dokaz: Tačnost iskaza koji se odnosi na povratna stanja direktno proističe iz homogenosti Markovljevog lanca i .definicije povratnog stanja.

Što se dela tvrđenja koji se odnosi na nepovratna stanja tiče, neka v_1 predstavlja broj koraka do prvog povratka u nepovratno stanje A_i, v_2 broj koraka do drugog povratka u to isto stanje, itd.

Ako je stanje A_i nepovratno, onda je $P(\{v_1 < \infty\}) = q < 1$. Nadalje, zbog homogenosti Markovljevog lanca je $P(\{v_2 < \infty \mid v_1 < \infty\}) = q$.

Dakle, $P(\{v_1 < \infty\})P(\{v_2 < \infty \mid v_1 < \infty\}) = q^2$, tj. $P(\{v_2 < \infty\}) = q^2$. Na isti način je (što se strogo može izvesti matematičkom indukcijom) $P(\{v_k < \infty\}) = q^k$, $k \in N$. Budući da je red $\sum_{k=1}^{\infty} P(\{v_k < \infty\}) = \sum_{k=1}^{\infty} q^k$ konvergentan (jer je $q < 1$) to na osnovu Borel-Kantelijeve leme I može da se realizuje samo konačno mnogo događaja $\{v_k < \infty\}$, $k \in N$ $\blacksquare$

Napomena. Borel-Kantelijeva lema I glasi: Ako red $\sum_{i=1}^{\infty} P(A_i)$ konvergira, onda se sa verovatnoćom 1 može realizovati samo konačno mnogo događaja iz niza $\{A_i\}_i$ $\square$

Nadalje se proučava asimptotsko ponašanje Markovljevih lanaca sa konačno mnogo stanja tj. vrednosti verovatnoća prelaska $p_{ij}(n)$ kada broj koraka n neograničeno raste.

Stav 2.4.9. Ako je za neko $n = n_0 \in N$ svaki elemenat matrice prelaska P_{n_0} strogo pozitivan, tj. $p_{ij}(n_0) > 0$ za sve $i, j \in \{1,2,3,\ldots,s\}$, tada postoji $\lim_{n\to\infty} p_{ij}(n) = p_j^*$ ($j \in \{1,2,3,\ldots,s\}$) koji ne zavisi od i. Brojevi p_j^* ($j \in \{1,2,3,\ldots,s\}$) se nazivaju finalne verovatnoće. Ovaj stav se u literaturi često naziva i Ergodička teorema za Markovljeve lance sa konačno mnogo stanja.

Dokaz: Pokazuje se da je funkcija $\min_i p_{ij}(n)$ monotono neopadajuća funkcija po n, da je funkcija $\max_i p_{ij}(n)$ monotono nerastuća funkcija po argumentu n i da razlika $(\max_i p_{ij}(n) - \min_i p_{ij}(n)) \to 0$, kada $n \to \infty$. Iz tih tvrđenja direktno sledi tvrđenje stava ∎

Ergodička teorema pokazuje da (ako važe uslovi teoreme) verovatnoća prelaska u velikom broju koraka ne zavisi od početnog stanja, tj. da vrednosti finalnih verovatnoća p_j^* ($j \in \{1,2,3,\ldots,s\}$) ne zavise od početnog stanja sistema.

Napomena. Finalne verovatnoće moraju zadovoljavati uslov $p_1^* + p_2^* + p_3^* + \cdots + p_s^* = 1$ (do ovog se dolazi direktno: $\sum_{j=1}^{s} p_j^* = \sum_{j=1}^{s} \lim_{n\to\infty} p_{ij}(n) = \lim_{n\to\infty} \sum_{j=1}^{s} p_{ij}(n) = \lim_{n\to\infty} 1 = 1$). Nadalje, finalne verovatnoće moraju zadovoljavati i sistem homogenih jednačina $p_j^* = \sum_{k=1}^{s} p_k^* p_{kj}$, za $j \in \{1,2,3,\ldots,s\}$ (ove homogene jednačine proizilaze iz jednačina Čepmen-Kolmogorova $p_{ij}(n) = \sum_k p_{ik}(m) p_{kj}(n-m)$ kada pustimo da $n \to \infty$) □

Primećuje se da za finalne verovatnoće p_j^* ($j \in \{1,2,3,\ldots,s\}$) važe iste jednakosti kao i za stacionarne verovatnoće p_j. Dakle, bez obzira na raspodelu verovatnoća stanja u početnom trenutku, ergodički Markovljev lanac se posle velikog broja koraka asimptotski ponaša kao stacionarni lanac: $p_j \approx p_j^*$ ($j \in \{1,2,3,\ldots,s\}$)

Primer. Ponovo uočimo slučajno pomeranje čestice sa apsorbujućim barijerama, ovog puta sa tri stanja ($s = 3$). Tada je:

$$P = \begin{bmatrix} q & p & 0 \\ q & 0 & p \\ 0 & q & p \end{bmatrix} \quad P_2 = P^2 = \begin{bmatrix} q & pq & q^2 \\ q^2 & 2pq & p^2 \\ p^2 & pq & p \end{bmatrix}$$

U ovom slučaju važe uslovi Ergodičke teoreme za Markovljeve lance, gde je $n_0 = 2$, pa se finalne verovatnoće mogu izračunati kao rešenje sistema koji čine tri homogene jednačine $p_j^* = \sum_{k=1}^{3} p_k^* p_{kj}$, tj.

$$\begin{bmatrix} p_1^* \\ p_2^* \\ p_3^* \end{bmatrix} = \begin{bmatrix} p_1^* & p_2^* & p_3^* \end{bmatrix} \begin{bmatrix} q & p & 0 \\ q & 0 & p \\ 0 & q & p \end{bmatrix}, \text{ a odavde je } \begin{cases} p_1^* = qp_1^* + qp_2^* \\ p_2^* = pp_1^* + qp_3^* \\ p_3^* = pp_2^* + pp_3^* \end{cases}$$

i jednačina $p_1^* + p_2^* + p_3^* = 1$. Rešavanjem i diskusijom ovakvog sistema dobijamo da je za $p \neq q$ jedinstveno rešenje sistema dato formulom

$$p_j^* = \frac{1-\frac{p}{q}}{1-\left(\frac{p}{q}\right)^3}\left(\frac{p}{q}\right)^{j-1},\ j=1,2,3$$

a da je za $p=q=\frac{1}{2}$ rešenje $p_1^* = p_2^* = p_3^* = \frac{1}{3}$ □

Definicija 2.4.10. Matrica $P=\{p_{ij}\}_{nxn}$.je:

a) nenegativna, ako je $(\forall i, j \in \{1,\ldots,n\})p_{ij} \geq 0$

b) pozitivna, ako je $(\forall i, j \in \{1,\ldots,n\})p_{ij} > 0$.

Definicija 2.4.11. Nenegativna matrica $P=\{p_{ij}\}_{nxn}$.je:

a) primitivna, ako $(\exists k \in N)P^k$ je pozitivno

b) reducibilna, ako postoje kvadratne matrice C i T takve da se primenom istih permutacija na vrste i kolone matrice P dobije matrica sa kvadratnom nula podmatricom u gornjem desnom uglu, tj.

$$P \approx \begin{bmatrix} C & 0 \\ R & T \end{bmatrix}$$

c) ireducibilna, ako nije reducibilna.

Napomena. Svaka pozitivna matrica je i primitivna□

Definicija 2.4.12. Stohastička matrica $P=\{p_{ij}\}_{nxn}$.je:

a) stabilna, ako ima identične vrste

b) dostupna kolonama, ako ima bar jedan pozitivan elemenat u svakoj koloni.

Stav 2.4.10. Neka su C, M i S stohastičke matrice, pri čemu je . M pozitivna, a S je dostupna kolonama. Tada je CMS pozitivna.

Dokaz: Neka je $A=CM$ i $B=AS$. Kako je C stohastička, to postoji bar jedan pozitivan elemenat u svakoj njenoj vrsti. Kako je $A=CM$, to je $a_{ij}=\sum_{k=1}^{n} c_{ik}m_{kj}$ i taj izraz je pozitivan za svako $i, j \in \{1,\ldots,n\}$. Slično je $b_{ij}=\sum_{k=1}^{n} a_{ik}s_{kj} > 0$ jer je S dostupno kolonama■

2.4.2. Optimalno upravljanje Markovljevim lancima

Optimalno upravljanje Markovljevim lancima može poslužiti kao teorijska osnova pri analizi i izboru vrednosti parametara koji određuju način primene stohastičkih operatora EA.

Definicija 2.4.13. Neka za Markovljev lanac $X_0, X_1, \cdots, X_n, \ldots$ verovatnoće prelaska p_{ij} zavise od jednog parametra d, koji pripada skupu D, tj. $p_{ij}=p_{ij}(d)$, $d \in D$. Na svakom koraku k, $k \in \{1,2,3,\ldots,n\}$ i za svako stanje sistema x_i na tom koraku k, operator može da izabere vrednost parametra d iz skupa D. Na taj način, operator upravlja Markovljevim lancem. Opšti problem se sastoji u sledećem: naći vrednost upravljajućeg parametra $d=d(k,x_i)$ kao funkcije rednog broja koraka k i stanja

sistema x_i na tom koraku k, tako da sistem ξ sa najvećom verovatnoćom dođe u određeni poskup stanja S, $S \subset \{x_1,\ldots,x_t,\ldots\}$.

Definicija 2.4.14. Funkcija $d = d(k, x_i)$ zove se funkcija odlučivanja, a funkcija $d^0 = d^0(k, x_i)$, za koju je $P(d^0) = \max_d P\{X_n \in S\}$ zove se funkcija optimalnog odlučivanja. Primenjujući funkciju optimalnog odlučivanja d^0, operator optimalno upravlja Markovljevim lancem. Problem optimalnog odlučivanja je problem određivanja $d^0 = d^0(k, x_i)$, $k \in \{0,1,\ldots,n-1\}$, $i \in N$, tako da primenom d^0, operator optimalno upravlja Markovljevim lancem.

Stav 2.4.11. Za Markovljev lanac sa verovatnoćama prelaska koje zavise od parametra, tj., $p_{ij} = p_{ij}(d)$ $d \in D$ je moguće kompletno rešiti problem optimalnog odlučivanja.

Dokaz: Biće opisan ceo postupak za kompletno rešavanje problema optimalnog upravljanja Markovljevim lancem. Po definiciji je , $P(k, x_i; d) = P(\{X_n \in S\} | \{X_k = x_i\})$

za $k \in \{0,1,\ldots,n-1\}$, $i \in N$ $d \in D$, .

Na osnovu teoreme o totalnoj verovatnoći je

$$P(k, x_i; d) = \sum_j p_{ij}(d) P(k+1, x_i; d)$$

pa je za $k = n-1$ (na pretposlednjem vremenskom trenutku, tj. u pretposlednjem koraku)

$$P(n-1, x_i; d) = \sum_j p_{ij}(d) P(n, x_i; d)$$

S obzirom da je u krajnjem koraku n verovatnoća da se stiglo u podskup optimalnih stanja data sa $P(n, x_j; d) = \begin{cases} 1, x_j \in S \\ 0, x_j \notin S \end{cases}$, to se zamenom u prethodnoj jednačini dobija:

$$P(n-1, x_i; d) = \sum_{j: x_j \in S} p_{ij}(d)$$

Odredimo sada $d = d^0(n-1, x_i)$ kao izbor vrednost parametra d u pretposlednjem koraku na osnovu stanja x_i tako da suma na desnoj strani prethodne jednačine bude najveća moguća, a sa $P^0(n-1, x_i)$ označimo tu maksimalnu vrednost desne strane prethodne jednačine, tj. $P^0(n-1, x_i) = P(n-1, x_i; d^0) = \max_d P(n-1, x_i; d)$. Prema tome, kada se sistem na $n-1$ koraku nađe u nekom stanju x_i, izborom vrednosti parametra $d^0 = d^0(n-1, x_i)$ se maksimizira verovatnoća da se u n-tom koraku nađe u nekom stanju iz skupa stanja S.

Za $k = n-2$, jednačina $P(k, x_i; d) = \sum_j p_{ij}(d) P(k+1, x_i; d)$ postaje

$$P(n-2, x_i; d) = \sum_j p_{ij}(d) P(n-1, x_i; d)$$

Ovde verovatnoće $p_{ij}(d)$ zavise samo od $d = d(n-2, x_i)$. U desnoj strani jednačine, izraz $P(n-1, x_i; d)$ se može zameniti sa $P^0(n-1, x_i) = P(n-1, x_i; d^0) = \max_d P(n-1, x_i; d)$, vrednošću koja se dobija optimalnim izborom d-a u pretposlednjem koraku $n-1$. Dakle,

$$P(n-2, x_i; d) = \sum_j p_{ij}(d) P^0(n-1, x_i; d)$$

Sada se $d^0 = d^0(n-2, x_i)$ određuje tako da suma na desnoj strani prethodne jednakosti bude maksimalna, a ta vrednost je: $P^0(n-2, x_i) = P(n-2, x_i; d^0) = \max_d P(n-2, x_i; d)$. Sada je, dakle, određena optimalna funkcija odlučivanja i za sistem u koraku $n-2$.

Ovaj postupak određivanja optimalne funkcije d^0 može da se nastavi unazad za $n-3, \ldots, 2, 1, 0$ i da se na taj način problem kompletno reši■

Napomena. U dokazu prethodnog stava osnovnu ulogu igra tzv. Belmanova jednačina

$$P^0(k, x_i) = \max_d \sum_j p_{ij}(d) P^0(k+1, x_i)$$

Belmanova jednačina pokazuje kako, po određivanju $d^0(k+1, x_i)$, treba da se izračuna $d^0(k, x_i)$ □

Primer. Neka postoji M predmeta koji se mogu upoređivati po nekom svom svojstvu. Zadatak se sastoji u tome da se izabere najbolji predmet. Mehanizam izbora je sledeći: slučajno se bira jedan predmet, registruje njegovo svojstvo i onda se odlučuje da li da se on odbaci i bira dalje u želji da se nađe još bolji, ili se kod tog izbora zaustavi. Pri tome se na odbačeni predmet ne može više vratiti.

Neka je $X_0 = 1$, X_1 redni broj predmeta koji se pokazao boljim od svih prethodnih, X_2 redni broj predmeta koji se pri daljem biranju pokazao boljim od svih prethodnih itd. Niz $X_0, X_1, X_2, \ldots$ se završava na nekom koraku v (v je slučajna promenljiva), kada je predmet sa rednim brojem X_v apsolutno najbolji među svim predmetima.

Uočimo stanja $x_1, x_2, \ldots, x_M, x_{M+1}$, gde x_i $i \in \{1, 2, \ldots, M\}$ označava da je predmet sa rednim brojem i bolji od svih ranije osmotrenih, dok x_{M+1} označava da je osmotren apsolutno najbolji predmet. Niz slučajnih promenljivih $X_0, X_1, X_2, \ldots, X_v, X_{M+1}, X_{M+1}, X_{M+1}, \ldots$ predstavlja Markovljev lanac. Verovatnoće prelaska kod ovakvog Markovljevog lanca su:

$p_{ij} = 0$, za $i \geq j$ i $j \leq M$;

$p_{M+1M+1} = 1$

$$p_{ij} = P(\{X_{n+1} = x_j\} \mid \{X_n = x_i\}) = \frac{P(\{X_{n+1} = x_j\} \cap \{X_n = x_i\})}{P(\{X_n = x_i\})} =$$

$$= \frac{\frac{(j-2)!}{j!}}{\frac{(i-1)!}{i!}} = \frac{i}{j(j-1)}, \text{ za } i < j \text{ i } j \leq M;$$

$$p_{iM+1} = \frac{\frac{(M-1)!}{M!}}{\frac{(i-1)!}{i!}} = \frac{i}{M}, \text{ za } i \leq M$$

Na primer, sledeća matrica prelaska je za $M = 3$: $P = \begin{bmatrix} 0 & 1/2 & 1/6 & 1/3 \\ 0 & 0 & 1/3 & 2/3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Razmotrimo optimalno upravljanje ovim Markovljevim lancem. U svakom stanju x_i $i \in \{1, \ldots, M\}$ operator može da odluči da li proces biranja treba da se nastavi ili da se zaustavi na tom i-tom predmetu. Ako se proces biranja nastavi, verovatnoće prelaska su

$$p_{ij} = \begin{cases} 0 & ,i \geq j \\ \dfrac{i}{j(j-1)} & ,i < j \leq M \\ \dfrac{i}{M} & ,j = M+1 \end{cases}.$$

Ako se proces biranja zaustavi na i-tom predmetu, verovatnoća prelaska je:

$$p_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

Što se parametra koji upravlja Markovljevim lancem tiče, možemo usvojiti da je $d = 0$ ako se zaustavljamo kod *i*-tog izbora i tada je $p_{ij}(d)$ dato drugom jednačinom, dok je $d = 1$ ako se postupak biranja nastavlja i u tom slučaju je $p_{ij}(d)$ dato prvom jednačinom-alternativom.

U ovom konkretnom slučaju, upravljajući parametar zavisi samo od stanja, a broj koraka *k* nema uticaja na proceduru izbora predmeta – pa je $d = d(x),\ x \in \{x_1, x_2, \ldots, x_M, x_{M+1}\}$. Kako broj koraka ne utiče na proceduru izbora predmeta, to on ne može uticati ni na optimalno upravljanje procedurom izbora predmeta.

Prema tome, u ovom slučaju optimalno upravljanje se sastoji u određivanju takve funkcije odlučivanja $d^0 = d^0(x),\ x \in \{x_1, x_2, \ldots, x_M, x_{M+1}\}$ koja čini verovatnoću izbora najboljeg predmeta maksimalnom.

Za neki konkretni (ne obavezno optimalni) izbor funkcije $d = d(x)$ verovatnoća izbora najboljeg predmeta (veličina koju treba optimalizovati) je

$$P(d) = \sum_{i:d(x_i)=0} \frac{i}{M} p_i$$

Pri tome, p_i predstavlja verovatnoću da je izbor završen kod *i*-tog predmeta, a količnik i/M je verovatnoća da je *i*-ti predmet apsolutno najbolji.

Svaka moguća funkcija $d(x)$ mora biti oblika $d(x) = 1$ za $x \in \{x_1, x_2, \ldots, x_{k-1}\}$ i $d(x) = 0$ za $x \in \{x_k, x_{k+1}, \ldots, x_M\}$, za neko *k*.

Neka je $P(k; d)$ verovatnoća da se izabere najbolji predmet ako je osmotreno ne manje od *k* predmeta. Na osnovu teoreme o totalnoj verovatnoći je:

$$P(k; d) = \sum_{j=k}^{M} p_{kj}(d) P(j; d)$$

Pri određivanju optimalnog upravljanja kreće se od kraja, tj. od koraka *M*:

$$d^0(x_M) = 0 \Rightarrow P^0(M) = \max_d P(M; 0) = 1$$

za $k = M-1$ je $P(M-1; d) = \begin{cases} \dfrac{M-1}{M} & ,d(x_{M-1}) = 0 \\ \dfrac{M-1}{M(M-1)} & ,d(x_{M-1}) = 0 \end{cases}$

S obzirom da je $P(M-1;0) > P(M-1;1)$, to je $d^0(x_{M-1}) = 0$, pa je $P^0(M-1) > P(M-1;0)$.

Postupak se nastavlja ovim putem, sve dok se ne stigne do koraka *k-1*. Tu je $d^0(x) = 0$ za $x \in \{x_k, x_{k+1}, \ldots, x_M\}$ i traži se $d^0(x_{k-1})$.

$$P(k-1;d)=\begin{cases}\dfrac{k-1}{M} & ,d(x_{k-1})=0\\ \dfrac{k-1}{k(k-1)}\dfrac{k}{M}+\dfrac{k-1}{k(k+1)}\dfrac{k+1}{M}+\cdots+\dfrac{k-1}{M(M-1)}1 & ,d(x_{k-1})=1\end{cases}$$

Ako se uporede vrednosti $P(k-1;0)$ i $P(k-1;1)$ dobija se da je $d^0(x_{k-1})=0$ kada je $\frac{1}{k-1}+\frac{1}{k}+\cdots+\frac{1}{M-1}\le 1$ i da je $d^0(x_{k-1})=1$ za $\frac{1}{k-1}+\frac{1}{k}+\cdots+\frac{1}{M-1}>1$.

Zaključak: Postupak traženja treba prekinuti kada se dođe do rednog broja M_0, za koji je $\frac{1}{M_0}+\frac{1}{M_0+1}+\cdots+\frac{1}{M-1}>1$ i tada taj redni broj označava najbolji među svim do tada osmotrenim elementima. Lako je pokazati da, za veliko M, je $M_0\approx\frac{M}{e}$ □

2.4.3. Analiza konvergencije Kanonskog GA pomoću Markovljevih lanaca

Kanonski GA (eng. Canonic GA - CGA) obuhvata jedinke sa genetskim kodom predstavljenim pomoću binarnih niski dužine l, operatore proporcionalne selekcije, jednopozicionu mutaciju sa verovatnoćom p_m i nespecificirano ukrštanje sa verovatnoćom primene p_c. Funkcija prilagođenosti je označena sa f, a broj jedinki u populaciji sa n.Za CGA je:

$$P\{a_i \text{ je izabran}\}=\frac{f(a_i)}{\sum_{j=1}^{n} f(a_j)}>0$$

$$P\{a_i \xrightarrow[mutacijom]{} a_i'\}=p_m^{H(a_i,a_i')}(1-p_m)^{l-H(a_i,a_i')}>0,$$

pri čemu je $H(a,b)$ oznaka za Hemingovo rastojanje (opisano u [Goldbe89]) između bitovnih niski a i b iste dužine.

Tada, CGA može biti opisan kao Markovljev lanac (videti [Rudolp95], [Suzuki95], [Suzuki97]), jer stanje CGA zavisi samo od gena jedinki u populaciji, pa je prostor stanja Markovljevog lanca $\xi=\{0,1\}^{nl}$. Svako stanje iz ξ može da se posmatra kao ceo broj u binarnoj reprezentaciji. Projekcija $\pi_k(i)$ bira k-ti segment bitova dužine l iz binarne reprezentacije stanja i. Ova projekcija se koristi za identifikovanje genetskog koda jedinke u populaciji.

Probabilističke promene gena unutar populacije, proizašle od genetskih operatora, obuhvaćene su matricom prelaska P, koja može biti dekomponovana na prirodan način kao $P=CMS$, gde C, M, S opisuju neposredne prelaske koji su prouzrokovani ukrštanjem, mutacijom i selekcijom.

Stav 2.4.11. Neka Markovljev lanac ima primitivnu stohastičku matricu prelaska P. Tada je taj Markovljev lanac ergodičan, tj. $P^k \xrightarrow[k\to\infty]{} P^\infty$, gde je P^∞ pozitivna stabilna stohastička matrica $P^\infty=1'p^\infty$ (gde vektor $p^\infty=p^0\lim_{k\to\infty}P^k=p^0P^\infty$ nema nula elemenata i ne zavisi od početne raspodele).

Dokaz: Lako se pokazuje da za ovakav Markovljev lanac važe uslovi Ergodičke teoreme tj. stav 2.4.9. ■

Stav 2.4.12. Neka je P reducibilna stohastička matrica, gde je $C_{m\times m}$ primitivna stohastička matrica i $R,T\neq 0$. Tada je:

$$P^{\infty} = \lim_{k\to\infty} P^k = \lim_{k\to\infty} \begin{bmatrix} C & 0 \\ R & T \end{bmatrix}^k = \lim_{k\to\infty} \begin{bmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-i} & T^k \end{bmatrix} = \begin{bmatrix} C^{\infty} & 0 \\ R_{\infty} & 0 \end{bmatrix}$$

i P^{∞} je stabilna stohastička matrica, $P^{\infty} = 1' p^{\infty}$, a $p^{\infty} = p^0 P^{\infty}$ je jedinstveno bez obzira na inicijalnu raspodelu i važi: $p_i^{\infty} > 0$ za $1 \le i \le m$ i $p_i^{\infty} = 0$ za $m < i \le n$.

Dokaz: Proizilazi iz osobina množenja matrica, primitivnosti matrice $C_{m\times m}$ i Ergodičke teoreme za Markovljeve lance■

Stav 2.4.13. Matrica prelaska za CGA sa verovatnoćom mutacije $p_m \in (0,1)$, verovatnoćom ukrštanja $p_c \in [0,1]$ i proporcionalnom selekcijom je primitivna matrica.

Dokaz: Operator ukrštanja može biti posmatran kao slučajna funkcija $\xi \to \xi$, tj. svako stanje iz ξ je probabilistički preslikano u neko drugo stanje iz ξ. Dakle, C je stohastička matrica. Na isti način se pokazuje stohastičnost matrica M i S.

S obzirom da se operator mutacije primenjuje nezavisno na svaki gen u populaciji, verovatnoća da stanje i posle mutacije pređe u stanje j dato je sa $m_{ij} = {p_m}^{H_{ij}} (1-p_m)^{nl-H_{ij}}$, za sve $i, j \in \xi$. Dakle matrica M je pozitivna.

Verovatnoća da selekcija ne menja stanje do kojeg se stiglo primenom mutacije može biti odozdo omeđena: $s_{ii} \ge \dfrac{\prod_{k=1}^{n} f(\pi_k(i))}{\left(\sum_{k=1}^{n} f(\pi_k(i))\right)^n} > 0$, za sve $i \in \xi$, pa je matrica S dostupna kolonama. Sada je, na osnovu stava 2.4.10, matrica $P = CMS$ pozitivna. Kako je svaka pozitivna matrica i primitivna, to je dokaz stava kompletiran■

Stav 2.4.14. CGA predstavlja ergodičan Markovljev lanac, što znači da postoji jedinstvena granična raspodela za stanja u lancu, pri čemu je verovatnoća da se bude u nekom od stanja pozitivna i nezavisna od inicijalne raspodele.

Dokaz: Direktna posledica stavova 2.4.14 i 2.4.12■

Prethodni stavovi ukazuju da inicijalna raspodela nema efekta na granično ponašanje Markovljevog lanca, pa pri analizi CGA inicijalizacija algoritma može biti isključena iz razmatranja.

Osobina ergodičnosti ima konsekvence na konvergenciju CGA. S obzirom da se u ovom poglavlju analizira konvergencija CGA, potrebno je precizno definisati taj pojam.

Definicija 2.4.15. Neka je $Z_t = \max\{f({\pi_j}^{(k)}(i)) \mid j \in \{1,\ldots,n\}\}$ sekvenca slučajnih promenljivih sa najboljom prilagođenošću u populaciji u stanju i pri koraku k. CGA konvergira ka globalnom optimumu ako i samo ako je $\lim_{k\to\infty} P\{Z_k = f^*\} = 1$, pri čemu je $f^* = \max\{f(b) \mid b \in \{0,1\}^l\}$ globalni maksimum.

Stav 2.4.15. CGA sa parametrima koji su opisani u stavu 2.4.13. ne mora da konvergira ka globalnom optimumu.

Dokaz: Neka je $i \in S$ stanje u kom je $\max\{f({\pi_j}^{(k)}(i)) \mid j \in \{1,\ldots,n\}\} < f^*$ i neka je p_i^k verovatnoća da će CGA biti u tom stanju i pri koraku izvršenja k. Jasno je da je $P\{Z_k \ne f^*\} \ge p_i^k \Leftrightarrow P\{Z_k = f^*\} \le 1 - p_i^k$. Na osnovu stava 2.4.11., verovatnoća da će CGA konvergirati ka stanju i iznosi $p_i^{\infty} > 0$. Stoga je (puštanjem da $k \to \infty$ u prethodnoj nejednačini)

$\lim_{k\to\infty} P\{Z_k = f^*\} \le 1 - p_i^\infty < 1$. Dakle, uslov iz definicije 2.4.15. nije ispunjen, pa CGA ne konvergira ka globalnom optimumu■

Može se uočiti da algoritam za prethodno definisan Markovljev lanac (tj. CGA) ne predstavlja praktičan Markovljev lanac. Naime, u realnim primenama GA (šire gledano i u realnim primenama EA) skoro uvek čuva dotad najbolje pronađeno rešenje. Jasno je da prethodna analiza ne odslikava tu činjenicu.

Analizom graničnog ponašanja Markovljevog lanca koji opisuje GA sa implementiranom strategijom zadržavanja dotad najbolje jedinke, dobija se da takav algoritam konvergira ka globalnom rešenju. Ovo tvrđenje je direktna posledica toga što je u ergodičkom Markovljevom lancu vreme prelaska od stanja i u stanje j konačno, bez obzira na i i j.

Da bi se pokazalo da je ovo tvrđenje u saglasnosti sa definicijom 2.4.15., potrebno je adaptirati opis Markovljevog lanca, uvećanjem populacije za dodatnu superjedinku, koja ne uzima učešće u evolucionim procesu. Zato se kardinalnost prostora stanja sistema povećava sa 2^{nl} na $2^{(n+1)l}$. Zbog povoljnosti pri notaciji, postaviće se superjedinka na krajnje levu poziciju u $(n+1)$–torki i neka pristup toj jedinki bude označen sa $\pi_0(i)$.

Verovatnoće prelaska iz onih stanja koja sadrže istu nisku superjedinke su takve da su izlistane ispod ostalih u matrici prelaska. Budući da se superjedinka ne menja primenom selekcije, ukrštanja i mutacije, to se proširene matrice prelaska za ukrštanje C^+, mutacije M^+ i selekcije S^+ mogu odrediti kao blok dijagonalnih matrica:

$$C^+ = \begin{bmatrix} C & & & \\ & C & & \\ & & \ddots & \\ & & & C \end{bmatrix} \quad M^+ = \begin{bmatrix} M & & & \\ & M & & \\ & & \ddots & \\ & & & M \end{bmatrix} \quad S^+ = \begin{bmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{bmatrix}$$

Dakle, matrice C^+, M^+ i S^+ su napravljene od 2^l blokova kvadratnih matrica (C, M i S respektivno) dimenzije $2^{nl} \times 2^{nl}$ koje su „složene“ po glavnoj dijagonali i ti blokovi imaju istu strukturu kao u ergodičkom slučaju, pa važi

$$C^+M^+S^+ = \begin{bmatrix} CMS & & & \\ & CMS & & \\ & & \ddots & \\ & & & CMS \end{bmatrix}, \quad CMS \neq 0$$

Operacija kopiranja novonastale bolje jedinke na poziciju superjedinke je predstavljena matricom unapređenja U, koja unapređuje prelazno stanje, koje sadrži jedinku bolju od superjedinke, u stanje gde je superjedinka identična toj boljoj jedinki.

Konkretno, neka je $b = \arg(\max\{f(\pi_k(i)) \mid k = \overline{1,n}\}) \in \{0,1\}^l$ najbolja jedinka iz populacije u ma kom stanju i, isključujući superjedinku. Tada je $u_{ij} = 1$, ako je $f(\pi_0(i)) < f(b)$ i $j \overset{def}{=} (b, \pi_1(i), \pi_2(i), \ldots, \pi_n(i))$, odnosno $u_{ii} = 1$ inače. Stoga, u svakoj vrsti postoji tačno jedan elemenat jednak jedinici – što ne važi za kolone (jer se za ma koje stanje $j \in S$ za koje je $f(\pi_0(j)) < \max\{f(\pi_k(j)) \mid k = 1,\ldots,n\}$ dobija da je j-ta kolona sastavljena od nula, tj. $u_{ij} = 0$ za sve $i \in S$). Drugim rečima, stanje će ili ostati nepromenjeno ili biti unapređeno.

Stoga se struktura matrice unapređenja U može opisati na sledeći način:

$$U = \begin{bmatrix} U_{11} & & & \\ U_{21} & U_{22} & & \\ \vdots & \vdots & \ddots & \\ U_{2^l 1} & U_{2^l 2} & \cdots & U_{2^l 2^l} \end{bmatrix}$$

pri čemu su podmatrice (tj. blokovi) U_{ab} dimenzije $2^{nl} \times 2^{nl}$.

Radi pojednostavljenja daljeg izvođenja, pretpostavimo da problem nalaženja maksimuma skupa $\{f(b) \mid b \in \{0,1\}^l\}$ ima samo jedno globalno rešenje. Tada je samo podmatrica U_{11} jedinična matrica, a svi ostali blokovi U_{aa} za $a \geq 2$ matrice sastavljene od nula i jedinica, pri čemu se jedinice mogu naći samo na pojedinim (ali ne svim) mestima na glavnoj dijagonali.

U tom slučaju (s obzirom da je $P = CMS$) dobija se:

$$P^+ = \begin{bmatrix} P & & & \\ & P & & \\ & & \ddots & \\ & & & P \end{bmatrix} \begin{bmatrix} U_{11} & & & \\ U_{21} & U_{22} & & \\ \vdots & \vdots & \ddots & \\ U_{2^l 1} & U_{2^l 2} & \cdots & U_{2^l 2^l} \end{bmatrix} = \begin{bmatrix} PU_{11} & & & \\ PU_{21} & PU_{22} & & \\ \vdots & \vdots & \ddots & \\ PU_{2^l 1} & PU_{2^l 2} & \cdots & PU_{2^l 2^l} \end{bmatrix}$$

gde je $PU_{11} = P > 0$.

Podmatrice PU_{a1} za $a \geq 2$ mogu biti objedinjene kao pravougaona matrica R, pa se za dokazivanje da odgovarajući GA konvergira ka globalnom optimumu može iskoristiti stav 2.4.12 (ispunjeni su svi uslovi za njegovu primenu).

Stav 2.4.16. CGA sa parametrima koji su opisani u stavu 2.4.13. koji čuva superjedinku (tj. dotad najbolje nađeno rešenje) posle izvršenja selekcije konvergira ka globalnom optimumu.

Dokaz: Podmatrica $PU_{11} = P > 0$ predstavlja verovatnoće prelaska za stanja koja sadrže globalno optimalnu superjedinku, tj. ova podmatrica sadrži verovatnoće prelaska za globalno optimalna stanja. Kako je P primitivna stohastička matrica, i kako je $R \neq 0$, to stav 2.4.12 garantuje da će verovatnoća ostanka u ma kom stanju koje nije globalno optimalno konvergirati ka nuli. Odatle, verovatnoća da sistem bude u nekom globalno optimalnom stanju konvergira ka jedinici, pa stoga i $\lim_{k\to\infty} P\{Z_k = f^*\}$ mora biti jedinica■

Napomena. U realnosti je mnogo češće primenjena strategija da se bolje rešenje prekopira preko dotad najboljeg odmah nakon evaluacije, jer u procesu selekcije to rešenje može biti izgubljeno. Tvrđenje za takav slučaj (u kom se bolja jedinka kopira preko dotad najbolje pre selekcije, a ne posle nje) se može izvesti analogno prethodnom izvođenju□

Stav 2.4.17. CGA sa parametrima koji su opisani u stavu 2.4.13. koji čuva superjedinku (tj. dotad najbolje nađeno rešenje) pre izvršenja selekcije konvergira ka globalnom optimumu.

Dokaz: Matrica prelaska u ovom slučaju je $P^+ = T^+ US$, gde je $T^+ = C^+ M^+$. Ako se postavi da je $T = CM$, dobija se:

$$P^+ = \begin{bmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \end{bmatrix} \begin{bmatrix} U_{11} & & & \\ U_{21} & U_{22} & & \\ \vdots & \vdots & \ddots & \\ U_{2^l 1} & U_{2^l 2} & \cdots & U_{2^l 2^l} \end{bmatrix} \begin{bmatrix} S & & & \\ & S & & \\ & & \ddots & \\ & & & S \end{bmatrix} = \begin{bmatrix} TU_{11}S & & & \\ TU_{21}S & TU_{22}S & & \\ \vdots & \vdots & \ddots & \\ TU_{2^l 1}S & TU_{2^l 2}S & \cdots & TU_{2^l 2^l}S \end{bmatrix}$$

pri čemu je $TU_{11}S = P > 0$. Isto kao u prethodnom izvođenju, podmatrice $TU_{a1}S$ za $a \geq 2$ mogu biti objedinjene u pravougaonu matricu $R \neq 0$, pa se opet (na isti način kao u prethodnom dokazu) dobija da je $\lim_{k\to\infty} P\{Z_k = f^*\} = 1$■

Napomena. Prethodne dve teoreme ne pokrivaju slučaj elitne strategije (u literaturi se ta strategija još naziva i elitističkom). Kada se koristi elitistička strategija, najbolja jedinka ne samo da ostaje, već aktivno učestvuje u generisanju novih jedinki. Stoga je kod elitističke strategije drugačija matrica prelaska i drugačija dinamika pretrage, što može biti bolje u nekim slučajevima, a gore u nekim drugim slučajevima (uporediti sa NFL teoremom [Wolper95]). □

2.4.4. Odnos između analize CGA pomoću Markovljevih lanaca i Teoreme o shemama

Stav 2.4.15 implicira da iz Teoreme o shemama ne proizilazi konvergencija CGA prema globalnom optimumu (jer u CGA važi Teorema o shemama, a on na osnovu 2.4.15 ne konvergira ka globalnom optimumu). Pored ovog fundamentalnog tvrđenja, potrebno je i dodatno osvetliti veze između ova dva pristupa u teorijskoj analizi EA, odnosno GA.

U ovom pogledu na sheme, shema H opisuje specifičan tip podskupova u obećavajućim regionima prostora $\{0,1\}^l$ pri rešavanju problema nalaženja $\max\{f(b) \mid b \in \{0,1\}^l\}$ (za ovaj problem se pretpostavlja da postoji tačno jedno globalno rešenje $b^* \in \{0,1\}^l$). Obično su ovi regioni (tj. podskupovi) predstavljeni niskom dužine l nad alfabetom $\{0,1,*\}$, pri čemu simbol zvezdica može biti instanciran bilo na 0, bilo na 1.

Prostor stanja $S = \left(\{0,1\}^l\right)^n$ predstavlja sva moguća stanja populacije $X = (b_1, b_2, \ldots, b_n) \in S$. Dakle, populacija X sadrži n jedinki dužine l.

U razmatranju koje sledi, presek označava da međusobno isti elementi populacije koji se javljaju više puta ne bivaju svedeni na jednu pojavu – već broj pojava ostaje isti.

Kvalitet sheme H svedene na skup X se definiše kao prosek prilagođenosti svih elemenata koji odgovaraju shemi, tj. koji se nalaze u $H \cap X$:

$$m(H, X) = \frac{1}{|H \cap X|} \sum_{b \in H \cap X} f(b)$$

Teorema o shemama suštinski tvrdi da je:

$$E\left(|H \cap X_{t+1}|\right) \geq |H \cap X_t| \frac{m(H, X_t)}{m(S, X_t)} \left(1 - c(H, X_t)\right)\left(1 - m(H, X_t)\right), \text{ skoro uvek.}$$

U prethodnoj formuli X_t je sekvenca populacija koje generiše CGA, funkcije $c(x)$ i $m(x)$ su procene gornje granice za verovatnoću da je elemenat podskupa (sheme) H modifikovan ukrštanjem i mutacijom na takav način da se rezultujuća jedinka više ne nalazi u shemi H.

Grubo govoreći, prethodni rezultat kazuje da je očekivani broj jedinki u populaciji X_{t+1} koje imaju natprosečnu prilagođenost, veći ili jednak od broja takvih jedinki u populaciji X_t, pod pretpostavkom da su verovatnoće $c(x)$ i $m(x)$ male.

Iako ta formula može da ukaže na dinamiku pretrage, ona ne implicira da pretraga konvergira ka globalnom optimumu.

Ako je $|\{b^*\} \cap X_t| = n$, tada Teorema o shemama tvrdi da je:

$$E\left(|\{b^*\} \cap X_{t+1}|\right) \geq n\left(1 - c(\{b^*\}, X_t)\right)\left(1 - m(\{b^*\}, X_t)\right)$$

a ovo ne povlači da očekivanje konvergira ka n, jer je donja međa izraza na desnoj strani nejednačine manja od n.

Nadalje, gornja nejednačina ne garantuje ni konvergenciju. Za globalnu konvergenciju potrebno je i dovoljno da $\lim_{t \to \infty} E(I_t) = 1$, što implicira da je $\lim_{t \to \infty} E\left(|\{b^* \cap X_t\}|\right) \geq 1$, pri čemu je:

$$I_t = h(b^*, X_t) = \begin{cases} 1, & ako \quad b^* \in \{\pi_1(X_t), \pi_2(X_t), \ldots, \pi_n(X_t)\} \\ 0, & inace \end{cases}$$

Stav 2.4.18. a) $\lim_{t\to\infty} E(I_t) = 1 \Leftrightarrow \lim_{t\to\infty} P(\{Z_t = f^*\}) = 1$

b) $\lim_{t\to\infty} E(I_t) = 1 \Rightarrow \lim_{t\to\infty} E\left(\left|\{b^*\} \cap X_t\right|\right) \geq 1$

Dokaz: Za deo a) dokaz proizilazi iz činjenice da je $\{I_t = 1\} \Leftrightarrow \{Z_t = f^*\}$. Neka $1_A(x)$ predstavlja indikatorsku funkciju. S obzirom da za bilo koji događaj A važi identitet $E(1_A) = P\{A\}$, tvrđenje stava dobijamo kad se u prethodnom identitetu stavi događaj $A = \{Z_t = f^*\}$ i pusti da $t \to \infty$.

Za dokazivanje dela stava b) uočava se funkcija $g(b^*, X_t)$ koja prebrojava optimalna rešenja b^* u populaciji X_t i neka je:

$$h(b^*, X_t) = \begin{cases} 1, & ako \quad b^* \in \{\pi_1(X_t), \pi_2(X_t), \ldots, \pi_n(X_t)\} \\ 0, & inace \end{cases}$$

Kako je $g(x) \geq h(x)$, dobija se da je:

$$\lim_{t\to\infty} E\left(\left|\{b^*\} \cap X_t\right|\right) = \sum_{i=1}^{|S|} g(b^*, i)\, p_i^{\infty} \geq \sum_{i=1}^{|S|} h(b^*, i)\, p_i^{\infty} = \lim_{t\to\infty} E(I_t) = 1$$

pri čemu p_i^{∞} u gornjem izvođenju označava graničnu raspodelu Markovljevog lanca. Tvrđenje stava direktno sledi iz prethodne nejednakosti.

Napomena. Uočava se da, u opštem slučaju, ne važi inverzija prethodnog stava. Neka je, na primer, $S = \{00, 01, 10, 11\}$, gde je $g(1, s) = \{0, 1, 1, 2\}$ i $p^{\infty} = \{0.01, 0.25, 0{,}25, 0.99\}$. Tada je $\lim_{t\to\infty} E\left(\left|\{b^*\} \cap X_t\right|\right) = 1.48 > 1$, pri čemu je $\lim_{t\to\infty} E(I_t) = 0.99 < 1$. Ovo znači da, čak iako očekivani broj optimalnih rešenja u populaciji konvergira ka vrednosti većoj od 1, globalna konvergencija nije zagarantovana.□

2.5. Još neki pristupi u analizi evolutivnih algoritama

Iako se pomoću EA lepo rešavaju određeni problemi optimizacije funkcija, ne treba smatrati da se EA samo zato koriste. Kritikovan je, dosta često viđen, uprošćen prilaz problematici EA, u kojem se EA posmatra samo kao sredstvo za optimizaciju funkcija (videti [DeJong92]). U prethodnim izvođenjima tj. prethodnoj teorijskoj analizi (i preko shema i preko Markovljevih lanaca) se pretpostavljao da su EA optimizatori funkcija. Autori naglašavaju da je EA i robustan adaptivan sistem sa velikim brojem najraznovrsnijih primena, pa se ukazuje na razlike koje postoje između EA za optimizaciju funkcija i EA za rešavanje drugih problema.

2.5.1 Kolateralna konvergencija

Uočena je tzv. kolateralna konvergencija (još se koristi i termin spora konvergencija), koja usložnjava proučavanje obmanjivačkih problema i samog obmanjivanja (eng. deception). Pokazalo se da obmanjivački problemi nisu jedini način da EA pri radu ne dođe do korektnog rezultata, tj. da se klasa EA-teških problema ne poklapa sa klasom EA-obmanjivačkih problema ([Greffe89]). Neki od uzroka koji dovode da neke neobmanjivačke funkcije ne budu lako optimizovane od strane EA su: neodgovarajuće kodiranje problema, prekidna, tj. uništavajuća priroda ukrštanja i mutacije, konačna veličina populacije (koja uzrokuje stohastičku grešku pri sempliranju) i višemodalnost funkcije.

2.5.2 Pejzaži prilagođenosti

Neki od novijih rezultata pokazuju da određena poboljšanja EA, koja se uspešno koriste pri rešavanju mnogih problema nemaju opšti značaj. Takav je, na primer, slučaj sa kodiranjem. Naime, pored već pomenutih operatora selekcije, ukrštanja i mutacije, EA su ,u ne manjoj meri, određeni i kodiranjem: - preslikavanjem jedinki realnog problema u odgovarajuće niske. Poželjno je da se to preslikavanje izvrši tako

da se niske koje odgovaraju jedinkama sa približno istom prilagođenošću nalaze bliže nego niske koje odgovaraju jedinkama čija se prilagođenost u značajnoj meri razlikuje. Međutim, dokazano je da ma koja dva fiksna kodiranja imaju isti prosek uspešnosti pri primeni na sve moguće funkcije tzv. NFL teorema [Wolper95] . Ako ovo primenimo na klasično i Grejovo kodiranje, to znači da postoji jednaki broj funkcija za koje će klasično kodiranje biti bolje od Grejovog, i funkcija za koje će Grejovo kodiranje biti bolje od klasičnog ([Antoni89], [Battle93], [Vose91], [Vose92]).

U tradicionalnim GA jedinke populacije su predstavljene kao niske bitova fiksne dužine. Ukupan broj svih mogućih jedinki je 2^n, gde je n broj bitova u niski. Pretpostavimo da su ove jedinke nacrtane kao tačke u dvodimenzionalnoj ravni, tako da je rastojanje među njima jednako broju bitovnih pozicija na kojima se ove dve jedinke razlikuju (tzv. Hemingovo rastojanje). Neka se potom svakoj od tačaka u ravni doda vertikalna komponenta koja je proporcionalna vrednosti prilagođenosti odgovarajuće jedinke. Na taj način je dobijen pejzaž prilagođenosti, gde brda tj. ispupčenja predstavljaju oblasti visoke prilagođenosti, a doline predstavljaju oblasti slabije prilagođenosti. Kada se generiše inicijalna populacija na pseudoslučajan način, time je efektivno postavljena mreža preko pejzaža prilagođenosti, kojom se simultano samplira funkcija prilagođenosti na različitim tačkama. Potom genetski operatori, kao što su ukrštanje i mutacija, određuju puteve istraživanja tog pejzaža. Prosta mutacija, koja menja jedan bit genetskog koda, dovodi do pomeranja za jedan korak u pejzažu prilagođenosti. Ukrštanje i složene mutacija dovode do skokova (ponekad velikih skokova) kroz pejzaž prilagođenosti. Iz svega ovog se vidi zašto pejzaži prilagođenosti predstavljaju odličan alat za vizuelizaciju težine problema, kao i za vizuelizaciju ponašanja GA pri rešavanju konkretnog problema (videti [Forest92], [Jones95]). Isti ovakav pristup bi se mogao primeniti i na EA, ali je tu implementacija nešto teža, jer je teže odrediti kako da se računa rastojanje između jedinki kada njihova reprezentacija nije binarna, tj. teže je odrediti kako sve moguće jedinke postaviti u dvodimenzionalnu ravan da bliže jedinke budu i bliže postavljene.

2.5.3. Model kockarove propasti

Ako se vratimo na primer sa početka poglavlja 2.4. (slučajni prelazak čestice sa apsorbirajućim barijerama), uočava se mogućnost primene modela opisanog u tom primeru na domen shema i gradivnih blokova. Naime, procesiranje shema od strane EA (videti [Harik97]) se može posmatrati kao slučajni prelazak čestice sa apsorbirajućim barijerama, pri čemu barijera $x = 0$ označava konvergenciju ka nekorektnom rešenju, dok barijera $x = n$ predstavlja konvergenciju ka korektnom rešenju. Inicijalna pozicija čestice je data sa x_0 i to je očekivani broj korektnih gradivnih blokova u slučajno inicijalizovanoj populaciji.

Lako se pokazuje (korišćenjem osnovnih tvrđenja kombinatorike) da očekivani broj korektnih gradivnih blokova u slučajno inicijalizovanoj populaciji od n jedinki, gde je korišćena binarna reprezentacija jedinki i dužina jedinki je l, iznosi $x_0 = n/2^l$.

U izvođenju koje sledi, pretpostavlja se da je samo slučajna inicijalizacija jedini izvor gradivnih blokova, tj. da ukrštanje i mutacija ne kreiraju, niti uništavaju značajan broj korektnih gradivnih blokova(tj. onih gradivnih blokova koji se sadrže u optimalnom rešenju problema). Usvajanje ove pretpostavke povlači da, kada korektan gradivni blok nestane iz potpopulacije više nema mogućnosti da se ponovo napravi. Sa druge strane, to znači da korektni gradivni blokovi ne mogu ni da se razgrade.

Nadalje, pretpostavlja se da se kao operator selekcije koristi turnirska selekcija sa veličinom turnira 2, gde se pseudoslučajno biraju dve jedinke iz populacije i bolje prilagođena jedinka biva izabrana.

Neka je verovatnoća da EA konvergira ka optimalnom rešenju označena sa P_{bb} i neka je sa p označena verovatnoća da se dobije najbolji gradivni blok. Tada, s obzirom da je P_{bb} verovatnoća da se stiglo do apsorbujuće barijere $x = n$, važi:

$$P_{bb} = \frac{1 - \left(\frac{q}{p}\right)^{x_0}}{1 - \left(\frac{q}{p}\right)^{n}}$$

Odavde se, ako uočimo da je $p > 1 - p = q$ (jer najbolji gradivni blok ima veću prosečnu prilagođenost od svih ostalih), i da je obično x_0 mnogo manje od n, dobijamo da se izraz u imeniocu prethodnog razlomka brzo približava jedinici. Odavde se, zamenom x_0 sa $n/2^l$ dobija:

$$P_{bb} \approx 1 - \left(\frac{1-p}{p}\right)^{n/2^l}$$

Iako pretpostavka o invarijantnosti gradivnih blokova u odnosu na ukrštanje i mutaciju ne odgovara realnosti, poređenja dobijenih rezultata (opisana u [Cantu00]) između izračunatih i eksperimentalno dobijenih vrednosti su pokazala da je razlika između njih veoma mala.

2.6. Ubrzanje paralelnih EA

Teorijske karakteristike raznih vrsta paralelnih EA bile su, i još uvek su, centralni domen proučavanja velikog broja autora (videti [Petey89], [Baluja93], [Cantu00]). Jedna od najvažnijih karakteristika paralelnog EA, kao i svakog drugog paralelnog algoritma je ubrzanje.

Definicija 2.6.1. Neka je sa T_s označeno vreme izvršavanja najboljeg sekvencijalnog algoritma, a sa T_p vreme izvršavanja paralelnog algoritma. Tada je ubrzanje paralelnog algoritma količnik ove dve veličine, tj.

$$S_p = T_s / T_p .$$

U slučaju EA, razmatraju se najbolji sekvencijalni algoritam koji dostiže rešenje sa populacijom najmanje moguće veličine i istim genetskim operatorima koji bivaju primenjeni i kod paralelnog EA. Izuzetno velike populacije kod sekvencijalnog EA kao rezultat pokazuju „naduvano" ubrzanje (videti [Cantu00]), pa ih treba izbegavati pri poređenjima. Pri osmišljavanju poređenja i proračunu ubrzanja mora se prepoznati da se distribuisani paralelni EA razlikuju od prostih EA koji imaju jedinstvenu populaciju, pa stoga oni imaju drugačije ponašanje (među paralelnim EA samo globalne paralelne EA karakteriše isto ponašanje kao sekvencijalne EA). Dakle, proračunata vrednost predstavlja relativnu meru ubrzanja paralelnog algoritma u odnosu na sekvencijalni i ni jednog trenutka u analizama ne treba gubiti iz vida kvalitet dobijenog rešenja. Uopšteno gledano, da bi se izvršilo korektno poređenje između ma kojeg sekvencijalnog i paralelnog programa, potrebno je da ta dva programa daju potpuno isti rezultat. U slučaju stohastičkih algoritama kao što je EA, korektno poređenje se mora zasnivati na očekivanom kvalitetu rešenja.

U zajednici EA istraživača je postojalo dosta kontroverzi o paralelnom ubrzanju kod EA sa većim brojem potpopulacija. Primarno neslaganje potiče od čestih tvrdnji istraživača o superlinearnom ubrzanju (videti npr. [Tanese89], [Koza95]), tj. tvrdnji da je vreme izvršavanja paralelnog EA smanjeno u odnosu na sekvencijalni EA za faktor koji je veći od broja procesora na kojima je paralelni EA izvršavan. Sumnja u neke od tvrdnji o superlinearnom ubrzanju potiče iz činjenice da su sekvencijalni i paralelni EA poređeni bez eksplicitnog razmatranja kvaliteta dobijenih rešenja. U [Cantu00] su, polazeći od dve krajnosti (potpuno povezanih potpopulacija i potpuno izolovanih potpopulacija) izvedene aproksimacije koje dovode u vezu ubrzanje, veličinu potpopulacija i kvalitet dobijenog rešenja. Tu je pokazano da, ako nema komunikacije među potpopulacijama, za očuvanje kvaliteta rešenja je potrebno da potpopulacije ostanu relativno velike, pa je ubrzanje malo. Sa druge strane, u situaciji kada je svaka potpopulacija direktno povezana sa svim ostalim, dolazi do značajnog poboljšanja performansi (jer potpopulacije mogu biti značajno manje).

Superlinearno ubrzanje se može tumačiti i sa stanovišta teoreme o shemama i hipoteze o gradivnim blokovima. Naime, jedino logično objašnjenje za superlinearno ubrzanje je da na neki način paralelni EA odradi manje operacija od sekvencijalnog EA – jasno je da, kada bi se oba algoritma izvršavala na potpuno isti način, maksimalno ubrzanje bi moglo biti samo linearno. U toj situaciji, uočavaju se dva moguća razloga za takvo ponašanje algoritma:

- Veoma je verovatno da veći broj potpopulacija tokom izvršavanja algoritma nalazi različite jedinke-rešenja i migracija (koja će biti detaljno opisana u sledećem poglavlju) u potpopulacije uvodi jedinke koje se razlikuju od svih postojećih. Ova povećana raznovrsnost potpopulacija, sa jedne strane, usporava konvergenciju algoritma, čime se možda operatoru ukrštanja daje dovoljno vremena da rekombinuje gradivne blokove u rešenje koje je kvalitetnije od rešenja do kojih dolazi sekvencijalni

EA. Sa druge strane, uvođenje različitih jedinki u potpopulacije omogućuje nezavisnu evoluciju delimičnih rešenja i njihovu integraciju posle migracije. Tako je u radu [Starkw91] opisano kako je prilikom optimizacije razdvojive funkcije (razdvojive funkcije će biti opisane u poglavlju 5.3.1) paralelnim EA moguće smanjiti veličinu potpopulacija, jer nije neophodno da se ceo problem rešava odjednom.

- Moguće je da na smanjenje posla kod distribuisanog EA utiče politika migracije koja uvećava intenzitet selekcije. O intenzitetu selekcije i uticaju migracije na intenzitet selekcije biće reči u poglavlju 3.

Na kraju, ne bi se moglo zaključiti da sve tvrdnje o superlinearnom ubrzanju potiču od povećanog selekcionog pritiska usled migracije. Rezultati koji ukazuju na superlinearno ubrzanje mogu biti dobijeni i zbog implementacije EA (npr. manje potpopulacije možda mogu da se kompletno smeste u procesorski keš, pa se algoritam brže izvršava ili neodgovarajuće veličina potpopulacija, koja ne garantuje isti kvalitet rešenja itd.).

2.7. Analiza rada potpuno povezanih paralelnih EA pomoću Markovljevih lanaca

Isto kao što je slučajno kretanje čestice sa apsorbujućim barijerama samo jedna ilustracija Markovljevih lanaca, tako i prethodno opisani rezultati primene modela kockarove propasti mogu da se dobiju i korišćenjem Markovljevih lanaca. Štaviše, analiza pomoću Markovljevih lanaca može da pruži i neke odgovore o ponašanju algoritma koje model kockarove propasti ne može da pruži.

U izvođenju koje sledi pretpostavlja se (isto kao i u izvođenjima u sekciji 2.5.3) da ukrštanje i mutacija ne kreiraju niti uništavaju značajan broj korektnih gradivnih blokova.

Nadalje, pretpostavljamo da se EA izvršava nad n_d potpuno povezanih potpopulacija, gde se posle svakog niza iteracija algoritma na potpopulacijama (tj. posle svake epohe) razmene najbolje jedinke. Dakle, epoha je period između dve uzastopne razmene najboljih jedinki potpopulacija. U tom slučaju broj korektnih gradivnih blokova na početku neke epohe zavisi isključivo od broja potpopulacija koje su konvergirale ka korektnom rešenju na kraju prethodne epohe. Ako i potpopulacija konvergira ka korektnom rešenju, tada će svaka od potpopulacija otpočeti novu τ-tu epohu sa χ_i kopija korektnih gradivnih blokova, a verovatnoća korektne konvergencije je $P_{bb}(\chi_i)$.

Na osnovu stavova u poglavlju 2.5.3, jasno je da izolovana potpopulacija (a sve potpopulacije su izolovane na početku izvršavanja paralelnog EA) posle prve epohe konvergira ka korektnom rešenju sa verovatnoćom $P_{bb}(x_0)$, gde je $x_0 = n/2^l$. Dakle, posle prve epohe, (zbog prethodno istaknute nezavisnosti), broj potpopulacija koje konvergiraju ka korektnim gradivnim blokovima je $V_1 = \{V_1(i)\}_{i=\overline{1,n_d}}$ i ima binomnu raspodelu:

$$V_1(i) = \binom{n_d}{i} P_{bb}(x_0)^i (1 - P_{bb}(x_0))^{n_d - i}$$

U ovom trenutku dolazi do razmene jedinki i početka sledeće epohe. Da bi se pravilno izračunala verovatnoća korektne konvergencije posle druge epohe, moraju se razmotriti svi mogući rezultati prve epohe. Neka je V_τ vektor čija i-ta koordinata $V_\tau(i)$ predstavlja verovatnoću da tačno i potpopulacija konvergira ka korektnom rešenju posle τ-te epohe. Tada je verovatnoća korektne konvergencije data sa $P_{bb_\tau} = V_{\tau-1}U$, gde je U matrica-kolona čiji je i-ti elemenat $U(i) = P_{bb}(\chi_i)$. Zadatak je da se izračuna raspodela korektnih potpopulacija posle τ epoha, tj. $V_\tau(i)$.

Lako se može pokazati da prethodno opisano izračunavanje, uz već uvedenu pretpostavku da ukrštanje i mutacija ne utiču na gradivne blokove, predstavlja homogen Markovljev lanac. Početno stanje ovog sistema je $V_1 = \{V_1(i)\}_{i=\overline{1,n_d}}$

U takvom lancu, elemenat matrice prelaska P na poziciji (i, j) dat je sledećom formulom:

$$p_{i,j} = \binom{n_d}{j} (P_{bb}(\chi_i))^j (1 - P_{bb}(\chi_i))^{n_d - j}$$

Gornja formula, kao što je i pokazano u poglavlju 2.4, predstavlja verovatnoću prelaska (tokom jedne epohe) iz stanja u kome je i potpopulacija korektno prema stanju u kome će j potpopulacija korektno konvergirati.

Po određivanju početnog stanja sistema, utvrđivanju da se radi o Markovljevom lancu i određivanju matrice prelaska, nije teško izračunati raspodelu potpopulacija koje korektno konvergiraju: $V_\tau = V_1 P^{\tau-1}$.

S obzirom da je matrica prelaska P u ovom slučaju primitivna i stohastička, to je (na osnovu stava 2.4.11) Markovljev lanac ergodičan. Ergodičnost Markovljevog lanca ukazuje da verovatnoća prelaska u velikom broju koraka ne zavisi od početnog stanja, tj. da vrednosti finalnih verovatnoća ne zavise od početnog stanja sistema i slobodno se može pustiti da $\tau \to \infty$.

Sada može da se odredi i finalna raspodela za broj potpopulacija koje sadrže korektne gradivne blokove, kada se algoritam dugo izvršava, kao i očekivani broj epoha pre nego što sve potpopulacije dostignu korektno rešenje.

Postupak određivanja je u velikoj meri analogan postupku koji je opisan u poglavlju 2.4.3, s tim što se sada prelazak sistema iz stanja u stanje u jednom koraku ne odnosi na jednu generaciju (kako je to ranije bio slučaj), već na jednu epohu (period između dve uzastopne razmene jedinki između potpopulacija).

I ovde se, dakle, mora proširiti matrica prelaska - dodaje se prva vrsta koja dogovara apsorbirajućem stanju 0, kada nijedna od potpopulacija ne sadrži korektne gradivne blokove. Ta prva vrsta ima na prvom mestu jedinicu, a svi ostali elementi su nule. Takođe se dodaje i poslednja vrsta, koja odgovara (takođe apsorbirajućem) stanju n_d kada svaka potpopulacija sadrži korektne gradivne blokove. Poslednja vrsta ima na poslednjem mestu jedinicu, a sve ostale vrednosti su nule. Ove dve novododate vrste predstavljaju (jedina dva) neprolazna stanja sistema.

Preuređivanjem oznaka stanja sistema, matrica prelaska može da se napiše u obliku:

$$P = \begin{bmatrix} P_1 & 0 & 0 \\ 0 & P_2 & 0 \\ R_1 & R_2 & Q \end{bmatrix}$$

Ovde su P_1 i P_2 podmatrice sa verovatnoćama prelaska unutar dva zatvorena skupa stanja (u našem slučaju su ti skupovi jednočlani – stanje 0 i stanje n_d), pa su stoga matrice dimenzije 1×1 i važi $P_1 = P_2 = 1$. Matrica Q je podmatrica sa verovatnoćom prelaska između prolaznih stanja, a R_1 i R_2 su podmatrice sa verovatnoćama prelaska iz prolaznih stanja do prvog i drugog neprolaznog stanja, respektivno.

Neka je matrica N sopstvena matrica za matricu Q, tj. $N = (I - Q)^{-1}$.

Tada je očekivano vreme apsorpcije kod svakog prolaznog stanja i dato sa i-tim elementom vektora-kolone $T = N \cdot 1$, pri čemu 1 predstavlja vektor-kolonu sastavljenu od jedinica. Da bi se odslikala dodata apsorbujuća stanja, potrebno je vektor T uključiti nulu na početku i na kraju (jer su to vremena apsorpcije za stanja 0 i n_d). Ovako proširen vektor biće označen sa T', a prosečno vreme potrebno da se apsorbuju sva prolazna stanja se dobija množenjem inicijalne raspodele potpopulacija V_1 proširenim vektorom-kolonom T'.

Sa druge strane, verovatnoća apsorpcije od prolaznog stanja i do neprolaznog stanja l data je elementom na poziciji (i,l) matrice $N \cdot R$, gde je R matrica nastala „slepljivanjem“ matrica R_1 i R_2. U ovako oformljenom proizvodu, prva kolona predstavlja verovatnoće apsorpcije u stanje 0, a druga kolona verovatnoću da prolazno stanje bude apsorbovano u stanje n_d. Neka je ta druga vektor-kolona označena sa A. I u ovom slučaju treba vektor A proširiti nulom na početku i jedinicom na kraju - oni predstavljaju verovatnoće konvergiranja prema korektnom rešenju za ova dva stanja. Prošireni vektor A biće označen sa A'. Sada je, slično prethodnoj diskusiji, verovatnoća apsorbovanja u stanje n_d data sa: $P_{bb\infty} = V_1 A'$.

3. OPERATORI SELEKCIJE I MIGRACIJE

U radu se posebna pažnja posvećuje operatorima selekcije i operatorima migracije kod paralelnih Evolutivnih algoritama.

3.1. Karakteristike operatora selekcije

Tema proučavanja ovog poglavlja je operator selekcije i kriterijumi za upoređivanje operatora selekcije kod EA. U proučavanju operatora selekcije autori polaze od različitih prethodno definisanih veličina.
Tako su Goldberg i Deb uveli pojam vremena preuzimanja (eng. takeover time) - broj generacija potrebnih da se jedna jedinka proširi preko cele populacije, pri čemu se EA izvršava bez rekombinacije.
Neri je pri analizi operatora selekcije takođe proučavao EA kod kog se koristi isključivo selekcija, čime se jasnije video efekat određenih selekcionih mehanizama ([Neri95]). Bäck je, takođe, analizirao najčešće korišćene selekcione sheme u odnosu na njihovo vreme preuzimanja i izveo verovatnoće izbora jedinki kod velikog broja često korišćenih selekcionih shema ([Bäck91a], [Bäck92b], [Bäck95]). Mühlenbein je u analizi operatora selekcije proučavao uticaj ovog operatora na prosečnu prilagođenost populacije. Tidor i De La Maza su analizirali invarijantnost pri transliranju i pri skaliranju za nekoliko metoda selekcije (videti [DeLaMa93]).
Lako se da uočiti da se u ovim, prethodno pobrojanim, pristupima, bez obzira da li se radi o analizi baziranoj na ponašanju najbolje jedinke ili na prosečnoj prilagođenosti populacije, opisuje samo mali aspekat operatora selekcije.
Mnogo preciznija slika o ponašanju EA i efektu operatora selekcije se dobija preko raspodele prilagođenosti pre i posle selekcije ([Blickl95], [Blickl95a], [Blickl97]).
Nadalje će se pretpostaviti (mada nije toliko uobičajeno u korišćenju EA), da se najpre izvršava selekcija, a zatim rekombinacija. Ovakav pristup je (videti [Blickl95a], [Blickl97]) matematički ekvivalentan onom uobičajenijem, a dopušta odvojenu analizu selekcionog metoda.

Definicija 3.1.1. Funkcija raspodele prilagođenosti populacije je funkcija $s: R \to Z_0^+$ koja svakoj vrednosti prilagođenosti $f \in R$ dodeljuje broj jedinki u populaciji koje imaju tu vrednost prilagođenosti.

Definicija 3.1.2. Neprekidni analogon funkcije raspodele prilagođenosti je neprekidna realna funkcija, nazvana takođe raspodela prilagođenosti populacije - i ona se označava sa $\bar{s}(f)$. Domen ove funkcije je interval $[f_1, f_{n_f}]$, gde je f_{n_f} najveća, a f_1 najmanja vrednost funkcije prilagođenosti za populaciju.

Definicija 3.1.3. Neka je n_f broj međusobno različitih vrednosti funkcije prilagođenosti i neka su vrednosti prilagođenosti jedinki sortirane u rastućem poretku, tj. $f_1 < f_2 < .... < f_{n-1} < f_{n_f}$. Tada je kumulativna raspodela prilagođenosti, u oznaci $S(f_k)$, definisana na sledeći način:

$$S(f_k) = \begin{cases} 0 & , \quad k < 1 \\ \sum_{j=1}^{k} s(f_j) & , \quad 1 \le k \le n_f \\ n & , \quad k > n_f \end{cases}$$

Lako se uočava da kumulativna raspodela prilagođenosti neke vrednosti f_k predstavlja broj jedinki u populaciji čija je vrednost prilagođenosti f_k ili gora.

Definicija 3.1.4. Neka je $\bar{s}(f)$ neprekidna funkcija koja predstavlja raspodelu prilagođenosti populacije. Tada je neprekidni analogon kumulativne raspodele prilagođenosti, u oznaci $\bar{S}(f)$:

$$\overline{S}(f) = \int_{f_1}^{f} \overline{s}(x)dx$$

S obzirom na ove definicije, metod selekcije može biti opisan kao funkcija koja transformiše raspodelu prilagođenosti u neku drugu raspodelu prilagođenosti.

Definicija 3.1.5. Metod selekcije **Ω** je funkcija koja preslikava raspodelu prilagođenosti *s* u neku drugu raspodelu prilagođenosti *s'*. Dakle, *s'*=**Ω**(*s,param*), gde je *param* opciona lista parametara za konkretan metod selekcije.

Već je istaknuto da su genetski operatori, pa i operator selekcije, najčešće probabilistički. Stoga je neophodno da se uvede očekivana raspodela prilagođenosti.

Definicija 3.1.6. Očekivana raspodela prilagođenosti po primeni metoda selekcije **Ω**, u oznaci **Ω***, se definiše na sledeći način:

$$\Omega^*(s, param) = E(\Omega(s, param))$$

Često će se ova jednakost zapisivati i na sledeći način: $s^* = \Omega^*(s, param)$.

Ovakvim pristupom se ne posmatra samo najbolja jedinka niti samo prosečna prilagođenost populacije, već se vodi računa o prilagođenosti svih jedinki populacije. Cilj analize je da se predvidi s* u zavisnosti od date raspodele prilagođenosti s.

Stav 3.1.1. Disperzija dobijene raspodele prilagođenosti s^* računa se kao:

$$\sigma_s^2 = s^* \left(1 - \frac{s^*}{n} \right)$$

Dokaz: Neka $S^*(f_k)$ označava očekivani broj jedinki sa vrednošću prilagođenosti f_k posle selekcije. Ta veličina je dobijena izvođenjem eksperimenata oblika: izaberi jedinku iz populacije koristeći izvesni selekcioni mehanizam. Stoga je verovatnoća izbora za jedinku sa vrednošću prilagođenosti f_k jednaka p_k, gde je:

$$p_k = \frac{s^*(f_k)}{n}.$$

Dakle, za svaku vrednost prilagođenosti f_k, $S^*(f_k)$ je slučajna promenljiva sa Bernouli-jevom raspodelom. Disperzija takve slučajne promenljive je:

$$np_k(1 - p_k) = s^*(f_k)\left(1 - \frac{s^*(f_k)}{n} \right).$$

Odavde direktno sledi tvrđenje. ∎

Indeks s kod disperzije σ_s^2 ukazuje da se radi o istom sempliranju kao i kod raspodele prilagođenosti. Do ovolike disperzije je došlo zato što je selekcioni metod izvršen u n nezavisnih eksperimenata.

Definicija 3.1.7. Očekivana prosečna prilagođenost populacije pre selekcije, u oznaci *M*, je:

$$M = \frac{1}{n} \sum_{j=1}^{n} f_j$$

Očekivana prosečna prilagođenost populacije posle selekcije, u oznaci M^*, je:

$$M^* = \frac{1}{n_f} \sum_{j=1}^{n_f} s(f_j)$$

Definicija 3.1.8. Disperzija raspodele prilagođenosti pre selekcije, u oznaci $(\overline{\sigma})^2$, je:

$$(\overline{\sigma})^2 = \frac{1}{n_f} \sum_{j=1}^{n_f} (s(f_j) - M)^2$$

Disperzija raspodele prilagođenosti posle selekcije, u oznaci $\left(\overline{\sigma}^*\right)^2$, je:

$$\left(\overline{\sigma^*}\right)^2 = \frac{1}{n}\sum_{j=1}^{n}\left(f_j - M^*\right)^2$$

Nadalje će se, zbog jednostavnosti zapisa, raditi sa raspodelom prilagođenosti koja je neprekidna funkcija prilagođenosti. Može se pokazati da važe i diskretni ekvivalenti definicija i tvrđenja dobijenih u kontinualnom slučaju.

Definicija 3.1.9. Neprekidni analogon očekivane prosečne prilagođenosti populacije pre selekcije, u oznaci $\overline{M}$, je:

$$\overline{M} = \frac{1}{n}\int_{f_1}^{f_{nf}} \bar{s}(f)\, f df$$

Neprekidni analogon očekivane prosečne prilagođenosti populacije posle selekcije, u oznaci $\overline{M^*}$, je:

$$\overline{M^*} = \frac{1}{n}\int_{f_1}^{f_{nf}} \overline{s^*}(f)\, f df$$

Definicija 3.1.10. Neprekidni analogon disperzije raspodele prilagođenosti pre selekcije, u oznaci $\left(\overline{\sigma}\right)^2$, je:

$$\left(\overline{\sigma}\right)^2 = \frac{1}{n}\int_{f_1}^{f_{nf}} \bar{s}(f)(f-\overline{M})^2 df = \frac{1}{n}\int_{f_1}^{f_{nf}} f^2 \bar{s}(f) df - \overline{M}^2$$

Neprekidni analogon disperzije raspodele prilagođenosti posle selekcije, u oznaci $\left(\overline{\sigma^*}\right)^2$, je:

$$\left(\overline{\sigma^*}\right)^2 = \frac{1}{n}\int_{f_1}^{f_{nf}} \overline{s^*}(f)(f-\overline{M^*})^2 df = \frac{1}{n}\int_{f_1}^{f_{nf}} f^2 \overline{s^*}(f) df - \overline{M^*}^2$$

Lako se uočava razlika između disperzija definisanih u definiciji 3.1.10. i disperzije iz stava 3.1.1.

Definicija 3.1.11. Stopa reprodukcije, tj. stepen reprodukcije (eng. reproduction rate) datog selekcionog metoda, u oznaci $\overline{R(f)}$, je funkcija koja prostor vrednosti prilagođenosti preslikava u skup realnih brojeva. Stopa reprodukcije za neku vrednost prilagođenosti se određuje kao količnik broja jedinki sa određenom prilagođenošću pre i posle selekcije:

$$\overline{R(f)} = \begin{cases} \dfrac{\overline{s^*}(f)}{\bar{s}(f)} & ,\bar{s}(f) > 0 \\ 0 & ,\bar{s}(f) = 0 \end{cases}$$

Razuman metod selekcije treba da favorizuje dobre jedinke, dodeljujući im stopu reprodukcije veću od 1, i da kažnjava loše, tj. slabije prilagođene, jedinke postavljajući njihovu stopa reprodukcije na vrednost manju od 1. Tokom svake selekcije u iterativnom postupku, pojedine jedinke se gube i bivaju zamenjene kopijama drugih jedinki. Stoga se izvesna količina genetskog materijala, koja se nalazi u nekim (pretežno lošim) jedinkama, može izgubiti. Za kvantifikovanje gubitaka te vrste uvodi se nova veličina.

Definicija 3.1.12. Gubitak raznovrsnosti (eng. loss of diversity) datog selekcionog metoda, u oznaci p_d, je količnik između broja jedinki u populaciji koji ne biva izabran tokom faze selekcije i ukupnog broja jedinki u populaciji.

Gubitak raznovrsnosti je, kao što se iz definicije 3.1.12. vidi, broj između 0 i 1. Gubitak raznovrsnosti treba biti što je moguće manji, jer veliki gubitak raznovrsnosti povećava rizik prerane konvergencije.

U uskoj vezi sa pojmom gubitka raznovrsnosti je kriterijum poređenja nazvan broj izgubljenih gena. Taj, prevashodno empirijski kriterijum je, u svojoj doktorskoj disertaciji, uveo De Jong (videti [Goldbe89]).

Još jedan kriterijum ekvivalentan gubitku raznovrsnosti je procenat jedinki koje su izabrane za reprodukovanje, u oznaci RR. Tu meru je u svojoj doktorskoj disertaciji, 1989. godine, uveo Baker. Lako se da pokazati ekvivalentnost ovih mera: $RR = 100(1 - p_d)$.

Sledeća veličina koja se koristi za karakterisanje operatora selekcije je intenzitet selekcije (eng. selection intensity). Intenzitet selekcije i selekcioni pritisak su sinonimni termini koji se često koriste u različitim

kontekstima i za različite osobine selekcionog metoda. U nastavku ove sekciji će biti izvedene formule koje određuju selekcioni pritisak najpopularnijih selekcionih metoda. Važno je istaći da kod distribuisanih EA na selekcioni pritisak ne utiče samo selekcija, već i migracija, pa će kasnije u ovom radu (poglavlje 3.3.1) pažnja biti posvećena dejstvu migracije na selekcioni pritisak.

U ovom radu se termin intenzitet selekcije koristi na isti način kao u klasičnoj genetici. Izmena prosečne prilagođenosti populacije je razumna mera za ovu osobinu metoda selekcije. U genetici je izraz intenzitet selekcije uveden radi dobijanja normalizovane, nedimenzionirane mere. Ideja je da se meri napredak tokom selekcije pomoću selekcione razlike - razlike između prosečne prilagođenosti populacije pre i posle selekcije. Količnik selekcione razlike i srednje disperzije populacije predstavlja željenu nedimenzioniranu meru, i ta mera se naziva intenzitet selekcije.

Definicija 3.1.13. Intenzitet selekcije $I(\Omega,\bar{s})$ za selekcioni metod Ω i raspodelu prilagođenosti $\bar{s}(f)$ je veličina $I(\Omega,\bar{s}) = \dfrac{\overline{M^*} - \overline{M}}{\sigma}$.

Važno je istaći da intenzitet selekcije zavisi od raspodele prilagođenosti kod inicijalne populacije, pa će različite raspodele prilagođenosti dovesti do različitih intenziteta selekcije za isti selekcioni metod.

Definicija 3.1.14. Standardizovani intenzitet selekcije za dati selekcioni metod Ω, u oznaci $I(\Omega)$, je:

$$I(\Omega) = \int_{-\infty}^{+\infty} \overline{\Omega^*}\big(N(0,1)(f)\big) f df$$

Dakle, standardizovani intenzitet selekcije je matematičko očekivanje prosečne vrednosti prilagođenosti populacije posle primene metoda selekcije Ω na normalizovanu normalnu raspodelu prilagođenosti $N(0,1)(f)$, datu formulom $(1/\sqrt{2\pi})e^{-(f^2/2)}$.

Prethodna definicija intenziteta selekcije može biti primenjena samo ako je metod selekcije invarijantan u odnosu na skaliranje i translaciju.

Definicija 3.1.15. Disperzija selekcije za dati selekcioni metod Ω, u oznaci $V(\Omega)$, je:

$$V(\Omega) = \int_{-\infty}^{+\infty} (f - I(\Omega))^2 \overline{\Omega^*}\big(N(0,1)(f)\big) df$$

Dakle, disperzija selekcije je disperzija raspodele prilagođenosti populacije po primeni metoda selekcije Ω na normalizovanu normalnu raspodelu $N(0,1)(f)$.

Postoji važna razlika između disperzije selekcije i gubljenja raznovrsnosti. Gubljenje raznovrsnosti daje odnos neizabranih jedinki u ukupnoj populaciji, bez obzira na vrednost funkcije prilagođenosti za jedinke iz populacije. Zbog toga je moguće odrediti gubljenje raznovrsnosti nezavisno od inicijalne raspodele prilagođenosti. Disperzija selekcije se definiše kao nova disperzija raspodele prilagođenosti, uz pretpostavku da je inicijalna raspodela prilagođenosti normalizovana i normirana tj. $N(0,1)(f)$.

3.1.1 Gradivni blokovi, šum prilagođenosti, veličine populacije i selekcija

Prema hipotezi o gradivnim blokovima, mehanizam selekcije treba da izabira one jedinke koje imaju korektne gradivne blokove i da eliminiše ostale. Međutim, ponekad dolazi do pogrešnih izbora.

Goldberg je, zajedno sa svojim saradnicima, analizirao ovu pojavu (videti [Goldbe92]).

Definicija 3.1.16. Označimo sa p verovatnoću da se korektno odluči u izboru između jedinke i_1 koja „nosi" najbolji gradivni blok H_1 i jedinke i_2 koja „nosi" drugi po kvalitetu gradivni blok H_2.

Stav 3.1.1. Verovatnoća da se korektno odluči pri izboru data je sa:

$$p = \Phi\left(\frac{d}{\sqrt{{\sigma_{H_1}}^2 + {\sigma_{H_2}}^2}}\right)$$

, gde je $d = \overline{f_{H_1}} - \overline{f_{H_2}}$.

Dokaz: Budući da su raspodele prilagođenosti f_1 i f_2 za jedinke i_1 i i_2 normalne (tj. $f_1 \approx N(\overline{f_{H_1}}, \sigma_{H_1})$ i $f_2 \approx N(\overline{f_{H_2}}, \sigma_{H_2})$), to i razlika prilagođenosti $f_1 - f_2$ ima normalnu raspodelu, tj. $f_1 - f_2 \approx N(\overline{f_{H_1}} - \overline{f_{H_2}}, \sigma_{H_1} - \sigma_{H_2})$. Odavde se, uvođenjem $d = \overline{f_{H_1}} - \overline{f_{H_2}}$ i normalizacijom dobija da je verovatnoća pravilnog izbora

$$p = \Phi\left(\frac{d}{\sqrt{{\sigma_{H_1}}^2 + {\sigma_{H_2}}^2}}\right)$$

, pri čemu je Φ kumulativna funkcija raspodele za normalizovanu normalnu raspodelu $N(0,1)$ ∎

Stav 3.1.2. Verovatnoća da se pravilno izabere u problemu sa m nezavisnih i jednako-skaliranih particija je

$$p = \Phi\left(\frac{d}{\sqrt{2m'}\,\sigma_{bb}}\right)$$

Dokaz: Da bismo izračunali disperziju za sheme H_1 i H_2 pretpostavićemo da je funkcija prilagođenosti suma m nezavisnih podfunkcija F_i (koje su sve iste dužine k) za svaku od particija koja maksimalno obmanjuje. Pod ovim pretpostavkama, ukupna disperzija prilagođenosti je

$${\sigma_F}^2 = \sum_{i=1}^{m} {\sigma_{F_i}}^2 .$$

U domenima gde su svih m particija uniformno skalirane (tj. imaju istu težinu), prosečna disperzija svakog gradivnog bloka ${\sigma_{F_i}}^2$ je ista i iznosi ${\sigma_{bb}}^2$. U takvom slučaju, ukupan šum dolazi iz $m' = m - 1$ particija koje se ne takmiče je ${\sigma_F}^2 = m'{\sigma_{bb}}^2$.

Stoga, verovatnoća da se pravilno izabere u problemu sa m nezavisnih i jednako-skaliranih particija je

$$p = \Phi\left(\frac{d}{\sqrt{2m'}\,\sigma_{bb}}\right)$$

, čime je dokaz završen∎

Evolutivni algoritmi se često koriste u uslovima kada se prilagođenost jedinke ne može tačno odrediti zbog šuma. Šum može poticati iz problemskog domena koji je inherntno šuman, ili od aproksimacije funkcije prilagođenosti (kada je tačno određivanje prilagođenosti jedinke previše skupo).

U ovoj sekciji se ispituje kako šum prilagođenosti utiče na veličinu populacije. Taj problem su razmatrali Goldberg, Miler, Deb i Cantu-Paz (videti [Goldbe92], [Miller97] i [Cantu00]).

Definicija 3.1.17. Prilagođenost sa šumom neke jedinke, u oznaci f', je data sa $f' = f + n$, gde je f tačna vrednost prilagođenosti, a n je šum koji je prisutan tokom evaluacije prilagođenosti jedinke.

Efekat dodatog šuma je povećanje disperzije prilagođenosti u populaciji, što povećava težinu pravilnog izbora među jedinkama koje se takmiče u procesu selekcije. Stoga, jednačine iz prethodnih stavova treba da budu modifikovane kako bi uključile efekat eksplicitnog šuma.

Ako se pretpostavi da šum ima normalnu raspodelu $N(0, {\sigma_N}^2)$, tada disperzija prilagođenosti postaje ${\sigma_F}^2 + {\sigma_N}^2$. Stoga je verovatnoća korektnog izbora između jedinke sa optimalnim gradivnim blokom i jedinke sa drugim po redu gradivnim blokom u šumovitom okruženju:

$$p = \Phi\left(\frac{d}{\sqrt{{\sigma_{H_1}}^2 + {\sigma_{H_2}}^2 + 2{\sigma_N}^2}}\right)$$

U slučaju uniformno – skaliranih problema to postaje

$$p = \Phi\left(\frac{d}{\sqrt{2\left(m'\sigma_{bb}{}^2 + \sigma_N{}^2\right)}}\right).$$

3.2. Operatori selekcije kod evolutivnih algoritama

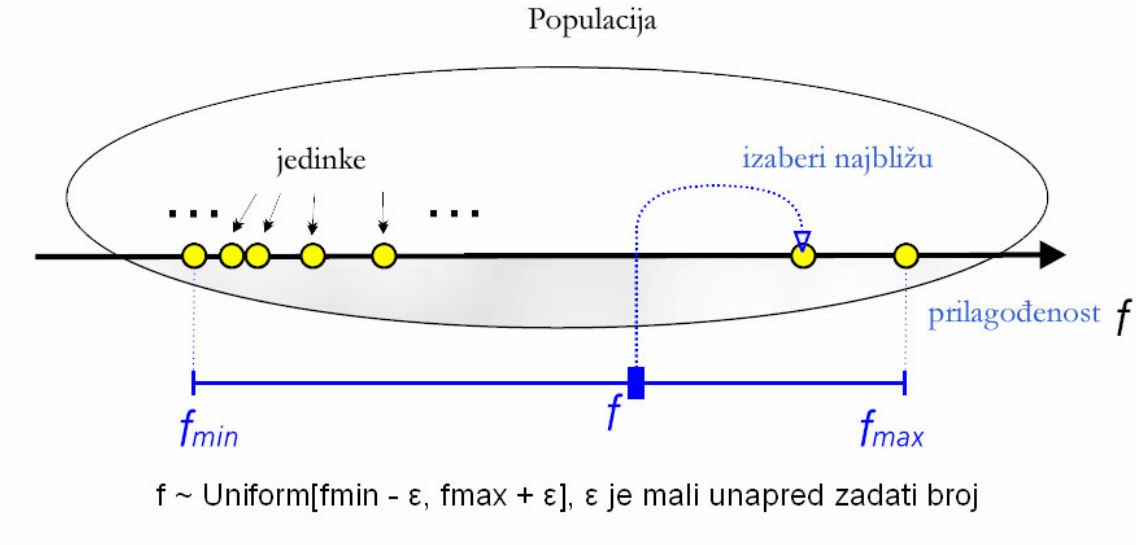
Operator selekcije se u biološki inspirisanoj literaturi naziva i operatorom odabiranja. To je postupak odlučivanja koje će jedinke nestati, a koje preživeti i preneti svoj genetski materijal u sledeće generacije.
Obično se operator selekcije dizajnira tako da i „sreća" ima uticaja, tj. tako da operator selekcije ne funkcioniše potpuno deterministički. Stoga, slabije jedinke ponekad mogu pobediti, tj. biti izabrane, i preneti svoje gene u nove generacije. Tako dizajniran operator selekcije preciznije emulira pojave i procese u prirodi. Naime, i u prirodi ne preživi uvek najbolji, niti trenutno najbolji nose sav kvalitetan genetski materijal. Često se dogodi da slabije jedinke nose genetski kod koji dovodi do toga da njihovi potomci imaju bolje karakteristike od potomaka trenutno najboljih jedinki. Ovako napravljen operator selekcije doprinosi očuvanju raznovrsnosti populacije, što je ključna stvar za kvalitetno traženje najboljeg rešenja i za izbegavanje prerane konvergencije.
Operator selekcije treba da poboljša prosečni kvalitet populacije davanjem veće mogućnosti za kopiranje u sledeću generaciju onim jedinkama koje su kvalitetnije, tj. koje su bolje prilagođene. Na taj način se eksploatišu prethodno stečene informacije o prilagođenosti jedinki u adaptivnom procesu traženja rešenja. Operatori ukrštanja i mutacije, jednom rečju nazvani operatorima rekombinacije, menjaju genetski materijal populacije (bilo na način opisan u poglavlju 1.5, bilo nekim drugim, egzotičnijim postupkom). Tako se u populaciju uključuju nove jedinke, čime bivaju istražene nove alternative, nove mogućnosti za rešenje.
EA se (videti [Bauer84]) susreću sa inherentnim konfliktima između eksploatisanja (eng. exploatation) i istraživanja (eng. exploration).
Eksploatisanje znači da se tokom traženja rešenja koriste informacije dobijene kod prethodno posećenih tačaka, kako bi se odredilo koja mesta mogu biti uspešna za sledeću posetu. Način rada metoda nazvanog penjanje uz brdo predstavlja tipičan primer eksploatisanja. Tehnike pretrage kod kojih je dominantno eksploatisanje postižu dobre rezultate pri traženju lokalnih maksimuma.
Istraživanje predstavlja postupak pribavljana novih informacija, tj. posećivanja sasvim novih regiona pretrage, da bi se videlo da li se u tim regionima može naći bilo šta obećavajuće. Za razliku od eksploatisanja, istraživanje uključuje skokove u nepoznato. Multimodalni problemi, tj. problemi sa više lokalnih ekstremuma, ponekad mogu biti rešeni isključivo korišćenjem tehnika pretrage kod kojih je dominantno istraživanje.
Sledeći primer ukazuje na značaj dobrog i preciznog postavljanja odnosa između eksploatacije i istraživanja.
Primer. Pretpostavimo da je izvršenjem prethodnih koraka genetskog algoritma otkriveno da niske u populaciji sa obrascem *1010* na bitovnim pozicijama *5-8* (tj. shema *****10101**...*) imaju mnogo bolje vrednosti prilagođenosti. Ako algoritam dovodi do brzog fiksiranja u bit-pozicijama *5-8* na vrednosti *1010*, tada se znanje eksploatiše kako bi se popravila prosečna prilagođenost populacije. Rizik od prebrzog fiksiranja bitova je sužavanje pretrage. Jer, dalja ispitivanja mogu pokazati da fiksirani obrazac *1010* na tim pozicijama i nije tako dobar, tj. da postoje bolje alternative□
Ma koja krajnost u podešavanju odnosa između eksploatacije i istraživanja nije dobra: ako bi eksploatacija bila dominantna, tada bi mogućnost vezivanja procesa pretrage za neki od lokalnih ekstremuma (nazvana prerana konvergencija - eng. prematural convergence) bila neprihvatljivo velika; u slučaju dominacije istraživanja proces traženja rešenja će trajati neprihvatljivo dugo i po svojim karakteristikama će se približiti čisto slučajnom pretraživanju.
Optimalan odnos eksploatacije i pretraživanja je različit za različite EA. Balans između eksploatacije i istraživanja može biti podešen određivanjem veličina stohastičkih parametara koje kontrolišu izvršenje genetskih operatora. U nekim drugim pristupima pokušava se postizanje balansa između eksploatisanja i istraživanja tako što se koriste dodatne informacije - npr. jedinke koje se dobro ponašaju mogu biti primorane da pređu u sledeću generaciju.
Kako je balans između eksploatacije i istraživanja kritičan za efikasnost EA, neophodno je poznavanje osobina rekombinacije, selekcije i mutacije, kao i razumevanje uticaja ovih operatora na brzinu konvergencije genetskog algoritma.

Mnogi autori su predlagali razna poboljšanja, dakle nove varijante operatora selekcije, ukrštanja, mutacije, nove modalitete njihove primene, kao i druge izmene koje još temeljnije zadiru u prost evolutivni algoritam. Neke od predloženih izmena su bile skoncentrisane i efikasno primenljive samo na problem kojim se autor-predlagač bavio, dok su drugi predlozi bili opšte primenjivi.

Predloga i poboljšanja kod operatora selekcije ima mnogo. U ovom poglavlju će biti opisane: proporcionalna tj. rulet selekcija, selekcija isecanjem, linearno rangiranje, eksponencijalno rangiranje i stohastičko univerzalno sampliranje. Pored opisa samog operatora selekcije, odrediće se i vreme potrebno za izvršenje jednog koraka selekcije, kao i neke osnovne osobine svakog konkretnog selekcionog metoda. Turnirska i fino gradirana turnirska selekcija, koje su najvažniji predmet proučavanja u ovom radu, biće detaljno opisani i istraženi u ovom poglavlju.

Pored prethodno pobrojanih, postoje i drugi operatori selekcije: selekcija pomoću očekivane vrednosti, selekcija pomoću faktora gomilanja (eng. crowding factor), Riplijevo uniformno rangiranje (eng. uniform ranking), operator stohastičkog ostatka, (μ,λ) selekcija, Bolcmanova turnirska selekcija itd. Ali, kako su oni ipak manje značajni, i budući da se ređe koriste, u ovom radu o njima neće biti više reči.

Među operatorima selekcije kojima se u ovom radu ne bavimo podrobno, ipak se izdvaja operator uniformne selekcije po prilagođenosti (eng. Fitness Uniform Selection), definisan i opisan u radu [Hutter02]. Ovaj operator selekcije je naročito interesantan zato što je njegov autor došao do iste dijagnoze situacije i problema koji stoje pred klasičnim selekcionim operatorima, do koje je došao i autor ovog rada. Naime, Hutter u svom radu ističe da, ako je selekcioni pritisak previsok, tada se EA „zaglavi" uz lokalni optimum, jer je genetska raznovrsnost rapidno umanjena. U tom slučaju, suboptimalni genetski materijal, koji može pomoći u pronalaženju globalnog optimuma je prerano nestao, tj. nastupila je prerana konvergencija . Sa druge strane, selekcioni pritisak ne sme biti previše slab, ako se želi da EA bude efikasan.



slika 3.1 Shematski prikaz operatora uniformne selekcije po prilagođenosti

Pristup koji je Hutter izabrao za prevazilaženje ovog problema je radikalno drugačiji operator selekcije, koji za razliku od dotad poznatih operatora, ne pokušava da poveća prosečnu prilagođenost celokupne populacije, već se orjentiše na očuvavanje genetske raznovrsnosti u populaciji. Takav pristup je, na prvi pogled, u koliziji sa hipotezom o gradivnim blokovima i teoremom o shemama. Međutim, operator je prilikom primene u rešavanju realnih problema, videti [Legg04] dao veoma dobre rezultate (pokazao se boljim od turnirske selekcije). Na osnovu ovih rezultata i analize koja je izvršena u drugim problemima implementiran je EA sa uniformnom slekcijom po prilagođenosti u softverskom sistemu GANP. Izvršeni eksperimenti ukazuju da je (na posmatranim problemima SPLP i ISP) fino gradirana turnirska selekcija pokazala bolje performanse od uniformne selekcije po prilagođenosti.

3.2.1. Proporcionalna selekcija

Proporcionalna selekcija (eng. proportional selection) je bila jedina predložena od strane Holland-a.

Kod ovog operatora selekcije verovatnoća preživljavanja za neku jedinku direktno je proporcionalna vrednosti prilagođenosti te jedinke. Alternativni naziv za proporcionalnu selekciju, uveden je zbog načina realizacije ovog operatora: kreira se rulet sa onoliko pregrada (eng. slot) koliko ima jedinki u populaciji, pri čemu su veličine pregrada proporcionalne veličinama prilagođenosti jedinke. Po oformljenju ruleta, vrši se n puta simuliranje bacanja kuglice: pri svakom simuliranju bacanja, u sledeću generaciju prelazi ona jedinka u čiju pregradu upadne kuglica.

Preciznije iskazano, proporcionalna selekcija se realizuje sledećim algoritmom (zapisanim u pseudo PASCAL-u):

```
Ulaz: Populacija a
Izlaz: Populacija posle selekcije a'
proporcionalna_selekcija:
begin
  s0 := 0
  /* formiranje ruleta */
  for i := 1 to n do
    si := si-1+fi /n
  end
  /* simulacija bacanja kuglice */
  for i := 1 to n do
    r := random( [0, sn] )
    ai' := ak za k takvo da je sk-1≤r<sk
  end
  return a'
end
```

algoritam 3.1. – Proporcionalna selekcija

Vremenska složenost ovog algoritma je $O(n)$.

Proporcionalna selekcija ostavlja otvorenim dva izvora problema. Prvo, kako se za fiksiranu shemu ne može praktično izračunati njen aktuelni prosek prilagođenosti (prosek prilagođenosti svih jedinki koje odgovaraju shemi), to se vrednost prilagođenosti sheme procenjuje kroz konačno sekvencijalno sampliranje. Drugo, proporcionalna selekcija je sama po sebi proces sa velikim odstupanjima, pa mogu postojati velika odstupanja između očekivanog i stvarnog broja kopija izabrane jedinke.

Dakle, proporcionalna selekcija, s obzirom na efikasnost i korektnost, tj. ravnopravnost jedinki, nije najbolja vrsta selekcije. Naime, metod nije uvek fer, jer fluktuacije slučajnih brojeva (koje se nazivaju stohastičke greške, ili greške sampliranja) mogu prouzrokovati da broj izabranih jedinki bude bitno manji ili veći od očekivanog (videti [ManEAn93]). Ipak, zbog svog značaja i korišćenja u teorijskim razmatranjima, ovaj operator selekcije se ne može zaobići.

Još se može primetiti ([Blickl95], [Blickl97]) da će prethodno prezentirani mehanizam funkcionisati samo ukoliko su vrednosti funkcije prilagođenosti pozitivne. Stoga se, pre simulacije bacanja kuglice na ruletu, često vrši skaliranje vrednosti funkcije prilagođenosti. Verovatnoća izbora jako zavisi od načina skaliranja funkcije prilagođenosti,, jer je dokazano (videti [DeLaMa93]) da proporcionalna selekcija nije invarijantna u odnosu na translaciju. Sledeći primer pojašnjava i ilustruje ovo tvrđenje:

Primer. Neka se populacija sastoji od *10* jedinki, pri čemu najbolja jedinka ima vrednost prilagođenosti *11*, a najgora *1*. Tada je verovatnoća izbora za najbolju jedinku $p_b \approx 16.6\%$, a za najgoru $p_w \approx 1.5\%$. Ako se vrednost prilagođenosti translira za *100*, tj. doda se broj *100* svakoj od vrednosti prilagođenosti, dobija se da je $p_b' \approx 10.4\%$ i $p_w' \approx 9.5\%$.□

Zbog prethodno pobrojanih nedostataka, razni autori su predložili veći broj metoda skaliranja: statično linearno skaliranje, dinamično linearno skaliranje, eksponencijalno skaliranje, logaritamsko skaliranje ([Grefen89]), sigma isecanje itd. Drugi način za popravljanje proporcionalne selekcije, čiji je autor John Koza, je primoravanje da se određen procenat populacije, npr 80%, bira između 20% najbolje prilagođenih u populaciji ([Koza91]).

Mühlenbein i Schilerkamp-Voosen su istakli (videti [Blickl95a], [Blickl97]) da ove modifikacije jesu neophodne da bi mehanizam selekcije funkcionisao, a ne trikovi koji ubrzavaju njegovo izvršavanje.

Potrebno je još odrediti verovatnoću izbora za jedinku u proporcionalnoj selekciji. Direktno iz samog postupka izbora, tj. iz algoritma 3.1, proizilazi da jedinkama populacije u proporcionalnoj selekciji bivaju dodeljene verovatnoće izbora proporcionalne njihovoj prilagođenosti, tj. po sledećoj formuli:

$$p_s(a_k) = f(a_k) \Big/ \sum_{j=1}^{n} f(a_j) = f_k \Big/ \sum_{j=1}^{n} f_j$$

3.2.2. Selekcija isecanjem

U selekciji isecanjem (eng. truncation selection) sa vrednošću isecanja *T* samo deo od *T* najboljih jedinki može biti izabran, i sve jedinke u tom delu imaju istu verovatnoću izbora.

Formalnije iskazano, selekcija isecanjem se realizuje sledećim algoritmom (zapisanim u pseudo PASCAL-u):

```
Ulaz: Populacija a, velicina isecanja T∈[0,1]
Izlaz: Populacija posle selekcije a'
selekcija_isecanjem:
begin
  a* := populacija a sortirana u rastucem redosledu po prilagodjenosti
  for i := 1 to n do
       r := random( {[(1-T)N],...,N} )
       ai' := ar*
  end
  return a'
end
```

algoritam 3.2. – Selekcija isecanjem

Budući da algoritam zahteva sortiranje populacije, izvršenje selekcije isecanjem će u najgorem slučaju imati red $O(n \log n)$.

Mühlenbein je (videti [Blickl95a], [Blickl97]) uveo ovu selekcionu shemu u domen EA (preciznije rečeno, u domen evolutivnih algoritama). Bäck je (videti [Bäck95]) pokazao da je ovaj metod ekvivalentan (μ,λ) selekciji koja se primenjuje u evolutivnim strategijama za $T=\mu/\lambda$.

Može se odrediti i verovatnoća izbora konkretne jedinke pri selekciji isecanjem:

Stav 3.2.1. Verovatnoća izbora za jedinku a_k u selekciji isecanjem sa parametrom *T*, a uz pretpostavku da su jedinke u populaciji sortirane po prilagođenosti u neopadajući niz, je

$$p_s(a_k) = \begin{cases} 1 - \left(\dfrac{\lceil nT \rceil - 1}{\lceil nT \rceil} \right)^n & , k \geq n + 1 - \lceil nT \rceil \\ 0 & , k < n + 1 - \lceil nT \rceil \end{cases}$$

Dokaz: Dokazano u [Filipo98] ∎

3.2.3. Linearno rangiranje

Linearno rangiranje (eng. linear ranking) je prvi predložio Baker, kako bi se eliminisali ozbiljni nedostatci proporcionalne selekcije ([Grefen89], [Whitle89]).

Ovaj metod selekcije ima jedan parametar: stepen reprodukcije najgore prilagođene jedinke, u oznaci η-, koji se nalazi u intervalu $[0,1]$.

Kod selekcije linearnim rangiranjem, jedinke se sortiraju u neopadajućem redosledu po prilagođenosti. Po sortiranju svakoj jedinki se dodeli njen rang - najgora će imati rang 1, a najbolja rang *n*. Potom se svakoj jedinki a_k linearno dodeli verovatnoća izbora po sledećoj formuli:

$$p_s(a_k) = \frac{1}{n}\left(\eta^- + \left(\eta^+ - \eta^-\right)\frac{k-1}{n-1} \right)$$

U gornjoj formuli figuriše i veličina η^+, koja predstavlja stepen reprodukcije najbolje prilagođene jedinke u populaciji. Međutim, ova veličina nije parametar selekcionog metoda, ona se može iskazati preko η^-, tj. važi

Stav 3.2.2. Kod linearnog rangiranja je $\eta^+ = 2 - \eta^-$.

Dokaz: Dokazano u [Whitle89] ∎

Linearno rangiranje se realizuje sledećim algoritmom:

```
Ulaz: Populacija a, stepen reprodukcije najgore jedinke η⁻∈[0,1]
Izlaz: Populacija posle selekcije a'
linearno_rangiranje:
begin
  a* := populacija a sortirana u rastucem redosledu po prilagodjenosti
  /* pravljenje ruleta sa velicinama slotova jednakim verovatnocama izbora */
  s0 :=0
  for i := 1 to n do
    si := si-1 + ps(ai) /* ps(ai) se odredjuje po gornjoj formuli */
  end
  /* simulacija bacanja kuglice na takvom ruletu */
  for i := 1 to n do
      r := random( [0,sn])
      ai' := ak* za k takvo da je sk-1≤r<sk
  end
  return a'
end
```

algoritam 3.3. – Linearno rangiranje

Linearno rangiranje zahteva sortiranje populacije, pa složenošću algoritma dominira složenost sortiranja, tj. broj operacija u algoritmu je bar $O(n \log n)$.

Kao i u drugim rang-selekcijama, jedinkama se dodeljuju verovatnoće izbora koje ne zavise od veličine prilagođenosti, već samo od relativne pozicije jedinke u populaciji. Stoga će i jedinke sa istom prilagođenošću imati različite verovatnoće izbora.

Postoji još nekoliko rangiranja koja su ekvivalentna ovako definisanom linearnom. Tako je Koza predložio metod selekcije (videti [Blick95a]) koji se svodi na prethodnu formu. Pored toga, u literaturi se sreće Whitley-evo rangiranje. To je selekcioni metod slične vrste. Za parametar c selekciona predrasuda (eng. selection bias) koji figuriše u Whitley-evom rangiranju, pokazano je da, kada je $c \in [1,2]$, ovakva selekcija za $\eta^+ = c$ biva skoro svuda svedena na linearno rangiranje (videti [Bäck92b]). Ipak, u praksi se često koristi Whitley-eva forma operatora, zato što se pri njenoj realizaciji produkuje znatno manja greška sampliranja - nema n odvojenih simulacija bacanja kuglice.

3.2.4. Eksponencijalno rangiranje

Eksponencijalno rangiranje (eng. exponential ranking) se razlikuje od linearnog zato što verovatnoće izbora za rangirane jedinke imaju eksponencijalnu težinsku funkciju. Osnova te eksponencijalne funkcije, u oznaci c, nalazi se u intervalu $[0,1]$ i predstavlja parametar selekcionog metoda. Što je c bliže jedinici, to je eksponencijalnost selekcionog metoda manja.

Kod selekcije eksponencijalnim rangiranjem, jedinke se sortiraju u neopadajući redosled po prilagođenosti. Po sortiranju se svakoj jedinki dodeli njen rang - najgora će imati rang 1, a najbolja rang n. Potom se svakoj jedinki a_k linearno dodeli verovatnoća izbora - po sledećoj formuli:

$$p_s(a_k) = \frac{c^{n-k}}{\sum_{j=1}^{n} c^{n-j}}$$

Suma koja se javlja u imeniocu razlomka služi za normalizaciju verovatnoća, tj. obezbeđuje da je $\sum_{k=1}^{n} p_s(a_k) = 1$. Ekvivalentna jednakost za verovatnoću izbora jedinke se dobija ako se izračuna suma geometrijskog reda u imeniocu razlomka. Tada je:

$$p_s(a_k) = \frac{c^{n-k}(c-1)}{c^n - 1}$$

Jasno je da je algoritam za eksponencijalno rangiranje sličan algoritmu za linearno rangiranje. Preciznije rečeno, jedina razlika između ova dva algoritma je izraz koji na osnovu ranga jedinke određuje verovatnoću njenog izbora.

Algoritam za eksponencijalno rangiranje je sledeći:

```
Ulaz: Populacija a, stepen reprodukcije najgore jedinke η-∈[0,1]
Izlaz: Populacija posle selekcije a'
eksponencijalno_rangiranje:
begin
  a* := populacija a sortirana u rastucem redosledu po prilagodjenosti
  /* pravljenje ruleta sa velicinama slotova jednakim verovatnocama izbora */
  s0 :=0
  for i := 1 to n do
    si := si-1 + ps(ai) /* vrednost ps(ai) se odredjuje po gornjoj formuli */
  end
  /* simulacija bacanja kuglice na takvom ruletu */
  for i := 1 to n do
    r := random( [0,sn] )
    ai' := ak* za k takvo da je sk-1≤r<sk
  end
  return a'
end
```

algoritam 3.4. – Eksponencijalno rangiranje

Eksponencijalno rangiranje zahteva sortiranje populacije pa je broj operacija u algoritmu bar $O(n \log n)$.

I kod ove metode selekcije verovatnoće izbora ne zavise od veličine prilagođenosti, već samo od relativne pozicije jedinke u populaciji. Stoga će i jedinke sa istom prilagođenošću imati različite verovatnoće izbora.

3.2.5. Turnirska selekcija

Turniri su mala takmičenja između jedinki populacije, koji se nadmeću radi preživljavanja i učešća u sledećoj generaciji. Turniri mogu uključivati dve ili više jedinki.

Dakle, u turnirskoj selekciji (eng. tournament selection) jedinka biva izabrana ukoliko bude bolja od određenog broja slučajno izabranih protivnika. Veličina turnira, u oznaci t, je parametar ove selekcije.

Strožije iskazano, turnirska selekcija se realizuje sledećim algoritmom (zapisanim u pseudo PASCAL-u):

```
Ulaz: Populacija a, velicina turnira t, t∈N
Izlaz: Populacija posle selekcije a'
turnirska_selekcija:
begin
  for i := 1 to n do
     ai' := najbolja jedinka medju t jedinki slucajno izabranih iz skupa {ak }k=1,n'
  end
  return a'
end
```

algoritam 3.5. – Turnirska selekcija

Vreme izvršavanja za ovaj algoritam je $O(n\,t)$. Kako je obično t<<n, to je vreme izvršavanja linearno, tj. $O(n)$. Dakle, za razliku od shema sa eksplicitnim rangiranjem, kao što su linearno rangiranje i eksponencijalno rangiranje, turnirska selekcija nema potrebu za dodatnim sortiranjem populacije po prilagođenosti.

Turnirska selekcija, takođe, dopušta uobičajeno paralelno izvršavanje tokom konstrukcije novoizabranih jedinki populacije.

Ova selekcija je invarijantna u odnosu na translaciju i skaliranje(videti [DeLaMa93]). Za razliku od drugih operatora invarijantnih u odnosu na translaciju i skaliranje, turnirska selekcija ne zahteva da populacija bude prvo sortirana po prilagođenosti.

Iako kod turnirske selekcije nema eksplicitne simulacije bacanja rulet-kuglice, i kod nje se, ipak, javlja dosta velika greška sampliranja - jedinke nove generacije određuju se u n nezavisnih izbora, pa disperzija nije zanemarljiva.

Popularnost turnirske selekcije je porasla kada je utvrđeno da se ovaj operator može uspešno koristiti kod paralelnih EA ([DeJong95]). Da bi se mogle efikasno eksploatisati neke paralelne arhitekture, potrebno je da kontrolna struktura EA bude decentalizovana. Dakle, treba umanjiti komunikaciju tako što će se redukovati uloga procesa-gospodara i kontrola prebaciti na proces-rob. Tradicionalni selekcioni algoritmi, opisani u poglavlju 4, zahtevaju globalna izračunavanja, čije izvođenje uključuje veliku komunikaciju. Pokušaji decentralizacije ovih selekcionih algoritama, u cilju povećanja efikasnosti i smanjenja komunikacije, dovode do distribuisanog selekcionog procesa, čije se performanse (vrednosti parametara koji opisuju selekcioni proces) bitno razlikuju od performansi centralizovane verzije. Dakle, centralizovana i distribuisana verzija istog selekcionog metoda imaće različito evolutivno ponašanje, što je neprihvatljivo. Kod turnirske selekcije,

međutim, nema tog problema i moguće je lako napraviti efikasnu decentralizovanu kontrolnu proceduru, koja će se ponašati na isti način kao njen centralizovani ekvivalent.
I za turnirsku selekciju se može odrediti verovatnoća preživljavanja konkretne jedinke (videti [Bäck94], [Bäck95]):

Stav 3.2.3. Verovatnoća izbora za jedinku a_k u turnirskoj selekciji sa parametrom t, uz pretpostavku da sve jedinke imaju različite vrednosti prilagođenosti i da su sortirane po prilagođenosti u rastućem poretku, je data sledećom formulom:

$$p_s(a_k) = \frac{1}{n^t}\left(k^t - (k-1)^t\right)$$

Dokaz: Dokazano u [Blickl95a] ∎
Teorijske karakteristike raznovrsnih operatora selekcije, pa i operatora turnirske selekcije, proučavali su mnogobrojni autori (videti [Bäck92], [Bäck94], [Bäck95], [Blickl95], [Blickl95a], [DeJong95], [Eselhm89], [Neri95], [Thier94] itd.).

Stav 3.2.4. Očekivana raspodela prilagođenosti $s^* = \Omega^*_{tur}(s,t)$ po izvršenju turnirske selekcije sa turnirom veličine t na raspodelu s je:

$$s^*(f_k) = \Omega^*_{tur}(s,t)(f_k) = n\left(\left(\frac{S(f_k)}{n}\right)^t - \left(\frac{S(f_{k-1})}{n}\right)^t\right)$$

Dokaz: Dokazano u [Blickl95a] ∎

Stav 3.2.5. Neka je $\bar{s}(f)$ neprekidna raspodela prilagođenosti u populaciji. Tada je očekivana raspodela prilagođenosti po izvršenju turnirske selekcije sa turnirom veličine t, u oznaci $\overline{\Omega^*_{tur}}(s,t)$, data sledećom formulom:

$$\overline{s^*}(f) = \overline{\Omega^*_{tur}}(s,t)(f) = t\bar{s}(f)\left(\frac{\bar{S}(f)}{n}\right)^{t-1}$$

Dokaz: Dokazano u [Blickl95a] ∎

Stav 3.2.6. Stopa reprodukcije kod turnirske selekcije iznosi $\overline{R_{tur}}(f) = t\left(\frac{\bar{S}(f)}{n}\right)^{t-1}$.

Dokaz: Dokazano u [Blickl95a] ∎
Iz gore izvedene formule je jasno da jedinke sa najnižom prilagođenošću imaju najnižu stopu reprodukcije, dok jedinka sa najvećom prilagođenošću ima stepen reprodukcije t.

Stav 3.2.7. Stopa reprodukcije kod turnirske selekcije je rastuća funkcija prilagođenosti. Ova funkcija je neprekidna.
Dokaz: Dokazano u [Filipo98] ∎

Stav 3.2.8. Gubitak raznovrsnosti kod turnirske selekcije, u oznaci $p_{d,tur}$, je :

$$p_{d,tur} = t^{-\frac{1}{t-1}} - t^{-\frac{t}{t-1}}$$

Dokaz: Dokazano u [Filipo98] ∎
Dakle, sa porastom veličine turnira dolazi do povećanja gubitka raznovrsnosti. Primećuje se i još jedna važna osobina ([Blickl95a]): gubitak raznovrsnosti je nezavistan od inicijalne raspodele prilagođenosti.
Naravno, veličina turnira utiče i na intenzitet selekcije, odnosno selekcioni pritisak – sa povećanjem veličine turnira t povećava se i intenzitet selekcije.
Pored veličine turnira, na intenzitet selekcije utiče i veličina populacije n. U svrhu analiziranja uticaja veličine populacije na intenzitet selekcije kod turnirske selekcije, pretpostavićemo da se korektan gradivni blok takmiči protiv $t-1$ kopije prvog do najboljeg. Tada je $P\{prava\ odluka\} = 1/t$. Ova procena je konzervativna – u realnosti je verovatnoća često veća od $1/t$, zato što turnir može uključiti više od jedne

kopije najboljeg gradivnog bloka (naročito u daljem izvršavanju algoritma). Ipak i ta procena predstavlja dobru polaznu tačku za dalju analizu.

Povećana teškoća u donošenju ispravne odluke sa povećanjem veličine turnira može da se iskaže i preko sažimanja signala koji EA pokušava da detektuje (videti [Cantu00]). Podešavanjem novog signala na $d' = d + \Phi^{-1}\left(1/t\right)\sigma_{bb}$, gde je $\Phi^{-1}\left(1/t\right)$ inverzna funkcija kumulativna raspodele za normalnu normalizovanu raspodelu, sa argumentom $1/t$. Drugi izraz u prethodnoj jednačini je negativan, pa sa većom veličinom turnira signal postaje manji a populacija veća. I drugi autori su uočili da veličina populacije raste sa selekcionim pritiskom, odnosno intenzitetom selekcije (videti [Bäck96], [Cantu00]).

Prethodna razmatranja imaju veliki značaj za dizajn paralelnih evolutivnih algoritama, ne samo zato što izbor selekcionog metoda utiče na veličinu populacije, već i zbog toga što neke implementacije paralelnih EA mogu promeniti selekcioni pritisak. Često se događa da ti efekti na selekcioni pritisak budu implicitni i da implementator nema nameru da ih napravi.

Stav 3.2.9. Standardizovani intenzitet selekcije za turnirsku selekciju sa parametrom t se računa kao

$$I(tur,t) = I_{tur}(t) = \int_{-\infty}^{+\infty} tx \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right)^{t-1} dx$$

Dokaz: Dokazano u [Blickl95a] ∎

Stav 3.2.10. Disperzija selekcije za turnirsku selekciju sa parametrom t se računa kao:

$$V(tur,t) = V_{tur}(t) = \int_{-\infty}^{+\infty} t\left(x - I_{tur}(t)\right)^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right)^{t-1} dx$$

Dokaz: Dokazano u [Blickl95a] ∎

3.2.6. Fino gradirana turnirska selekcija

EA su, kako je već istaknuto, veoma robustan algoritam za rešavanje raznih klasa (obično veoma teških) problema. Postupak rešavanja problema nužno sadrži fazu podešavanja vrednosti stohastičkih parametara koji određuju stepen primene izabranog operatora u dizajniranom iterativnom postupku. Dakle, kada se autor odluči za vrstu selekcije, vrstu ukrštanja i/ili mutacije, kao i za kriterijum završetka iterativnog postupka, on nije fiksirao jedan algoritam, već celu familiju algoritama međusobno različitih po veličini stohastičkih parametara. Takva familija algoritama naziva se reproduktivni plan.

Prema tome, svako rešavanje problema ovom tehnikom zahteva pažljivo eksperimentisanje kako bi se našle najbolje vrednosti parametara. Odnos između istraživanja i eksploatacije treba da bude optimalan, a on direktno zavisi od problema koji se rešava.

Greffenstatte je svojim eksperimentima (videti [Grefen86], [Grefen89]) pokazao da to podešavanje parametara ne mora biti isuviše fino i da će EA, zbog svoje robusnosti, skoro jednako dobro funkcionisati za široki opseg vrednosti parametara - parametri se ponekad mogu razlikovati i za decimalu, a da se to ne odrazi na brzinu izvršavanja algoritma.

Ipak, kod turnirske selekcije je nešto drugačiji slučaj. Parametri svih ostalih operatora su realni brojevi, koji mogu uzimati neku od beskonačno mnogo vrednosti. Veličina turnira kod turnirske selekcije može uzimati vrednost nekog od prirodnih brojeva.

Za turnirsku selekciju je izbor parametra sužen na 2-3 logične vrednosti veličine turnira, pa se teško može podesiti odnos istraživanja i eksploatacije. Često se dešava da sa jednom veličinom turnira konvergencija bude isuviše spora, a da, ako se veličina turnira uveća za jedan, algoritam konvergira suviše brzo i proces pretrage se veže za lokalno najbolje rešenje. Stoga je poželjno dizajnirati takav operator selekcije koji će delovati slično kao turnirska selekcija, ali čije ponašanje može da se fleksibilnije podesi. Ovaj operator selekcije nazvaćemo Fino gradirana turnirska selekcija.

Fino gradirana turnirska selekcija ima jedan parametar: to je željena srednja veličina turnira, u oznaci t. Isto kao u turnirskoj selekciji, jedinka biva izabrana ukoliko bude bolja od određenog broja slučajno izabranih protivnika. Ali, veličina svakog od turnira koji se održavaju radi izbora jedinki koje prelaze u sledeću

generaciju nije više jedinstvena u okviru populacije, tj. u okviru jednog koraka selekcije biće održani turniri sa različitim brojem učesnika.

Mnogobrojni autori (npr. [Goldbe89], [Hollan92]) su primetili da među svim modifikacijama EA najviše uspeha, odnosno najveću opštu primenljivost, imaju one modifikacije za koje postoji analogija u prirodi koja nas okružuje. Takav je slučaj npr. sa paralelnim EA na MIMD ili SIMD računarskim sistemima gde se komunikacija među procesorima realizuje prosleđivanjem poruka. Razmena podataka među čvorovima takvog računarskog sistema je mnogo sporija i mnogo manjeg intenziteta nego unutar čvora. Tu EA funkcionišu na način koji je analogan toku evolucije na nekim međusobno dosta udaljenim ostrvima - potpopulacija na svakom od čvorova tog MIMD evoluira skoro nezavisno od potpopulacija na susednim čvorovima, s tim što povremeno neke jedinke potpopulacije na jednom čvoru emigriraju.

I turnirska i fino gradirana turnirska selekcija imaju svoj analogon u prirodi: poznato je da veći broj životinjskih vrsta (viši primati) živi u grupama, nazvanim krda, odnosno čopori, u kojim samo najjači, najprilagođeniji mužjak prenosi svoje gene potomstvu. Postupak kojim se određuje koji je od mužjaka vođa grupe umnogome odgovara turniru. U prirodi brojnost grupe nije fiksna: za svaku vrstu i lokalitet na kom se ta vrsta nalazi, broj životinja u grupi i broj odraslih mužjaka koji se bore za vođstvo varira, s tim da ne može izaći izvan nekih granica.

Fino gradirana turnirska selekcija se realizuje na sledeći način: na osnovu broja jedinki u populaciji n i željene srednje veličine turnira t, odrede se veličine turnira t^+ i t^-, kao i broj održavanja ovih turnira n^+ i n^-. Ove veličine zavise od broja jedinki u populaciji n i željene prosečne veličine turnira t, tj. $t^+ = t^+(n,t)$, $t^- = t^-(n,t)$, $n^+ = n^+(n,t)$, $n^- = n^-(n,t)$. Vrednosti za t^+ i t^-, te n^+ i n^- određuju se tako da bude $n^+ + n^- = n$ i da je srednja vrednost veličine održanih turnira što je moguće bliža t. Potom se. izvrši n^+ turnira sa t^+ članova i n^- turnira sa t^- članova. Vrednosti veličina turnira t^+ i t^- se (u najjednostavnijem slučaju) određuju tako da to budu prirodni brojevi manji i veći od željene srednje veličine turnira t. Po određivanju vrednosti za t^+ i t^-, vrednosti za n^+ i n^- se mogu odrediti iz gornjih uslova: njihov zbir je n, a željena srednja veličina turnira je t.

Ova jednostavna verzija operatora će biti realizovana na najjednostavniji mogući način. Izvršavaće se jednako brzo kao i klasični operator turnirske selekcije i, što je takođe izuzetno važno, ne dolazi do povećanja greške sampliranja u odnosu na klasičnu turnirsku selekciju. Može se očekivati da fluktuacija zbog izbora slučajnih brojeva ipak bude nešto veća, iako nema novih sampliranja. To je stoga što su jedinke koje učestvuju u turnirima manje kardinalnosti „ravnopravnije" od ovih drugih. Ali, kako se veličine turnira razlikuju najviše za jedan, to eventualno povećanje stohastičke greške mora biti minimalno, tj. doprinos ovog uzroka stohastičkoj grešci može da se zanemari.

Ukoliko se, dakle, uzme da je $t^- = \lfloor t \rfloor$ i $t^+ = \lfloor t \rfloor + 1$ rešavanjem sistema:

$$\begin{cases} n^+ + n^- = n \\ nt = n^+ t^+ + n^- t^- \end{cases}$$

dobijaju se vrednosti t+ i t-, te n+ i n- u zavisnosti od n i t:

$$\begin{cases} t^- = \lfloor t \rfloor \\ t^+ = \lfloor t \rfloor + 1 \\ n^- = \lfloor n(1 - (t - \lfloor t \rfloor)) \rfloor \\ n^+ = n - \lfloor n(1 - (t - \lfloor t \rfloor)) \rfloor \end{cases}$$

Prema tome, fino gradirana turnirska selekcija se realizuje sledećim algoritmom (zapisanim u pseudo PASCAL-u):

```
Ulaz: Populacija a, zeljena srednja velicina turnira t, t∈R
Izlaz: Populacija posle selekcije a'
fino_gradirana_turnirska_selekcija:
begin
  t⁻ := trunc( t )
  t⁺ := trunc( t ) + 1
  n⁻ := trunc ( n * ( 1 - ( t - trunc ( t ) ) ) )
  n⁺ := n - trunc ( n * ( 1 - ( t - trunc ( t ) ) ) )
  /* turniri velicine t⁻ */
  for i := 1 to n⁻ do
     a_i' := najbolja jedinka medju t⁻ slucajno izabranih iz skupa {a_k}k=1,n'
  end
  /* turniri velicine t⁺ */
  for i := n⁻+1 to n do
     a_i' := najbolja jedinka medju t⁺ slucajno izabranih iz skupa {a_k}k=1,n'
  end
  return a'
end
```

algoritam 3.6. – Fino gradirana turnirska selekcija

Vreme izvršavanja za ovaj algoritam je $O(n\,t)$. Kako je obično $t << n$, to je vreme izvršavanja linearno, tj. $O(n)$.

Kao što je u uvodu i istaknuto, ovako dizajniran operator predstavlja uopštenje turnirske selekcije.

Stav 3.2.11. Ako je željena veličina turnira fino gradirane turnirske selekcije ceo broj, tada se fino gradirana turnirska selekcija poklapa sa klasičnom turnirskom selekcijom.

Dokaz: Dokazano u [Filipo03] ∎

Ukoliko željena srednja veličina turnira t nije ceo broj, onda se, po već opisanom postupku, formiraju turniri tako da njihova prosečna veličina bude što je moguće bliža parametru t. Pri tome se prosečna veličina održanih turnira ponekad ne poklapa sa željenom veličinom turnira t.

Stav 3.2.12. Prosečna veličina održanih turnira fino gradirane turnirske selekcije jednaka parametru t, tj. željenoj prosečnoj veličini turnira, zaokruženom na tačnost $\frac{1}{n}$.

Dokaz: Dokazano u [Filipo03] ∎

Stav 3.2.13. Fino gradirana turnirska selekcija je invarijantna u odnosu na translaciju i skaliranje.

Dokaz: Dokazano u [Filipo03] ∎

Fino gradirana turnirska selekcija je dizajnirana tako da se, slično kao kod klasične turnirske selekcije, selekcioni algoritam može lako i prirodno decentralizovati. Dobijeni distribuisani algoritam će imati isti selekcioni pritisak kao njegov centralizovani analogon, što ga čini pogodnim za primenu kod paralelnih EA.

I za fino gradiranu turnirsku selekciju se može odrediti verovatnoća preživljavanja konkretne jedinke:

Stav 3.2.14. Verovatnoća izbora za jedinku a_k u turnirskoj selekciji sa parametrom *t*, uz pretpostavku da sve jedinke imaju različite vrednosti prilagođenosti i da su sortirane po prilagođenosti u rastućem poretku, data je sledećom formulom:

$$p_s(a_k) = \frac{n^+}{n^{1+t^+}}\left(k^{t^+} - (k-1)^{t^+}\right) + \frac{n^-}{n^{1+t^-}}\left(k^{t^-} - (k-1)^{t^-}\right).$$

Dokaz: Dokazano u [Filipo03] ∎

U ovom poglavlju se izvode teorijske karakteristike prethodno dizajniranog operatora selekcije. Izvedene karakteristike ukazuju da se klasična turnirska selekcija može posmatrati kao specijalni slučaj fino gradirane, i da novi operator ima ista ona dobra svojstva koja karakterišu turnirsku selekciju.

Stav 3.2.15. Očekivana raspodela prilagođenosti $s^* = \Omega^*_{ftur}(s,t)$, po izvršenju fino gradirane turnirske selekcije sa prosečnom veličinom turnira *t* i raspodelom *s*, data je sledećom formulom:

$$s^*(f_k) = \Omega^*_{ftur}(s,t)(f_k) = \frac{n^+}{n^{t^+}}\left(\left(S(f_k)\right)^{t^+} - \left(S(f_{k-1})\right)^{t^+}\right) + \frac{n^-}{n^{t^-}}\left(\left(S(f_k)\right)^{t^-} - \left(S(f_{k-1})\right)^{t^-}\right).$$

Dokaz: Dokazano u [Filipo03] ∎

Stav 3.2.16. Neka je $\overline{s}(f)$ neprekidna raspodela prilagođenosti u populaciji. Tada je očekivana raspodela prilagođenosti po izvršenju fino gradirane turnirske selekcije sa srednjom veličinom turnira *t*, u oznaci $\overline{\Omega^*_{ftur}}(s,t)$, data sledećom formulom:

$$\overline{s^*}(f) = \overline{\Omega^*_{ftur}}(s,t)(f) = \overline{s}(f)\left(\frac{n^+t^+}{n^{t^+}}\left(\overline{S}(f)\right)^{t^+-1} + \frac{n^-t^-}{n^{t^-}}\left(\overline{S}(f)\right)^{t^--1}\right).$$

Dokaz: Dokazano u [Filipo03] ■

Stav 3.2.17. Stopa reprodukcije kod fino gradirane turnirske selekcije iznosi

$$\overline{R_{ftur}}(f) = \frac{n^+t^+}{n^{t^+}}\left(\overline{S}(f)\right)^{t^+-1} + \frac{n^-t^-}{n^{t^-}}\left(\overline{S}(f)\right)^{t^--1}.$$

Dokaz: Dokazano u [Filipo03] ■

Stav 3.2.18. Stopa reprodukcije kod fino gradirane turnirske selekcije je rastuća funkcija prilagođenosti. Ova funkcija je neprekidna.

Dokaz: Dokazano u [Filipo03] ■

Stav 3.2.19. Neka je data jednačina:

$$\frac{n^+t^+}{n^{t^+}}x^{t^+-1} + \frac{n^-t^-}{n^{t^-}}x^{t^--1} = 1.$$

Ta jednačina ima formu: $ax^{q-1}+bx^{q-2}=1$. Neka je sa $x = G(t^+,t^-,n^+,n^-) = H(t,n)$ označeno rešenje jednačine dobijeno nekim od numeričkih postupaka. Tada je gubitak raznovrsnosti kod fino gradirane turnirske selekcije, u oznaci $p_{d,ftur}$, dat sledećom formulom:

$$p_{d,ftur} = \left(H(t,n)\right)^{[t]} + \frac{n+1-\left[n\left(1-\left(t-[t]\right)\right)\right]}{n}H(t,n)$$

Dokaz: Dokazano u [Filipo98] ■

Stav 3.2.20. Standardizovani intenzitet selekcije, za fino gradiranu turnirsku selekciju sa parametrom *t*, se računa po sledećoj formuli:

$$I(ftur,t) = I_{ftur}(t) = \int_{-\infty}^{+\infty} x\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}\left(\frac{n^+t^+}{n^{t^+}}\left(\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}dy\right)^{t^+-1} + \frac{n^-t^-}{n^{t^-}}\left(\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}dy\right)^{t^--1}\right)dx$$

Dokaz: Dokazano u [Filipo98] ■

Gornja integralna jednačina se može rešavati odgovarajućim numeričkim metodama i dobiti rešenje sa potrebnom tačnošću.

Stav 3.2.21. Disperzija selekcije za FGTS *t* se računa kao:

$$V(ftur,t) = V_{ftur}(t) =$$

$$= \int_{-\infty}^{+\infty} t\left(x - I_{tur}(t)\right)^2\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}\left(\frac{n^+t^+}{n^{t^+}}\left(\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}dy\right)^{t^+-1} + \frac{n^-t^-}{n^{t^-}}\left(\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}dy\right)^{t^--1}\right)dx$$

Dokaz: Dokazano u [Filipo98] ■

I ova integralna jednačina se, za konkretne vrednosti n i t može, sa potrebnom tačnošću, rešiti odgovarajućim numeričkim metodama.

Operator FGTS je veliki broj puta primenjen u okviru EA kojim su uspešno i efikasno rešavani raznovrsni EA problemi (videti [Kratic99a], [Kratic00a], [Kratic01], [Kratic01a], [Kratic01b], [Kratic01c], [Filipo01], [Kratic02], [Filipo03], [Kratic03], [Stan04], [Kratic05], [Stan06], [Kratic06a]). O efektima njegove praktične primene biće više reči u kasnijim poglavljima.

3.2.7. Disperzivna fino gradirana turnirska selekcija

Operator disperzivne FGTS je nadogradnja operatora FGTS. On se može primenjivati i kod klasičnih sekvencijalnih EA, ali očekivano polje njegove primene je selekcija u grubo usitnjenom paralelnom EA.

Slično kao što je to slučaj kod FGTS, i ovaj selekcioni operator ima jasan analogon u prirodi (koji u velikoj meri podseća na situaciju sa pejzažom prilagođenosti): nisu svi prostori u kojima žive pojedine zajednice viših primata podjednako pogodna za život i za razmnožavanje – negde su uslovi života lakši, a negde su teži.

Motivacija za uvođenje Disperzivne fino gradirane turnirske selekcije (nadalje će biti korišćena skraćenica DFGTS) je omogućavanje preciznog podešavanja odnosa između istraživanja i eksploatacije. U isto vreme, operator DFGTS se dizajnira sa ciljem da zadrži sve dobre teorijske osobine koje su krasile FGTS.

DFGTS se implementira na najjednostavniji način (alternativno bi se, isto kao i kod FGTS, pri retroaktivnom kreiranju populacije pored željene sredine populacije u obzir uzimala i disperzija). Ovaj, najjednostavniji, način određivanja veličina turnira dovodi do toga da stohastička greška sampliranja bude minimalna.

Isto kao u turnirskoj selekciji i u FGTS, jedinka biva izabrana ukoliko bude bolja od određenog broja slučajno izabranih protivnika. Ali, veličina svakog od turnira koji se održavaju radi izbora jedinki koje prelaze u sledeću generaciju nije više jedinstvena u okviru populacije, tj. u okviru jednog koraka selekcije na različitim potpopulacijama biće održani turniri sa različitim brojem učesnika.

Realizacija DFGTS algoritma se oslanja na dva parametra: željene srednje veličine turnira t i podele populacije na l potpopulacija, čije su veličine date uređenom l-torkom $(n_i)_{i=1}^{l}$. Bez gubljenja opštosti pretpostavićemo da su potpopulacije uređene u nerastućem poretku po svojoj veličini, tj. da je $n_1 \geq n_2 \geq n_3 \geq \cdots \geq n_l$. U i-toj potpopulaciji ($i \in \{1,\ldots,l-1\}$) sa brojem članova n_i biće održana klasična turnirska selekcija sa veličinom turnira $t_i \in N$, a u l-toj potpopulaciji će biti održana FGTS sa željenom srednjom veličinom turnira $t_l \in R$. Veličine turnira po potpopulacijama određuju se tako da se međusobno razlikuju najviše za jedan i da srednja vrednost veličine svih turnira bude što je moguće bliža parametru t.

Pre formalnog opisa algoritma DFGTS, sam postupak određivanja veličina turnira će biti prikazan kroz jedan konkretan slučaj.

Primer. Neka je populacija od $n = 200$ jedinki razdeljena na šest potpopulacija čije su veličine $n_1 = 50, n_2 = 50, n_3 = 30, n_4 = 30, n_5 = 20, n_6 = 20$ i neka je srednja veličina turnira $t = 3.4$. Tada će biti $t_1 = round(t) = 3$, i u tom trenutku će srednja veličina turnira biti $\frac{3 \cdot 50}{50} = 3 < 3.4$. Kako je prethodni izraz manji od parametra t, to će u sledećoj potpopulaciji veličina turnira biti veći ceo broj, tj. $t_2 = \lceil t \rceil + 1 = 4$. Prosečna veličina održanih turnira je sada $\frac{3 \cdot 50 + 4 \cdot 50}{50 + 50} = 3.5 > 3.4$. Sada je vrednost izraza veća od parametra, pa je $t_3 = \lceil t \rceil = 3$ i vrednost izraza (zaokružena na tri decimale) je $\frac{3 \cdot 50 + 4 \cdot 50 + 3 \cdot 30}{50 + 50 + 30} = 3.384 < 3.4$.

Nadalje je $t_4 = \lceil t \rceil + 1 = 4$ i $\frac{3 \cdot 50 + 4 \cdot 50 + 3 \cdot 30 + 4 \cdot 30}{50 + 50 + 30 + 30} = 3.5 > 3.4$.

Istom logikom kao i kod prethodnih slučajeva dobija se da je $t_5 = \lceil t \rceil = 3$ i da je prosečna veličina održanih turnira $\frac{3 \cdot 50 + 4 \cdot 50 + 3 \cdot 30 + 4 \cdot 30 + 3 \cdot 20}{50 + 50 + 30 + 30 + 20} = 3.444 > 3.4$. Na kraju, u poslednjoj potpopulaciji se izvršava FGTS sa parametrom $t_6 = 3.4 + (3.4 - 3.444) = 3.4 - 0.044 = 3.356$.

Dakle, DFGTS se realizuje sledećim algoritmom (zapisanim u pseudo PASCAL-u) :

```
Ulaz: Populacija a razdvojena na potpopulacije a1,a2,...al, velicine n1>=n2>=...>=nl
zeljena srednja velicina turnira t, t∈R
Izlaz: Populacija posle selekcije a'
Disperzivna_fino_gradirana_turnirska_selekcija:
begin
  t[1] := trunc( t )
  s := n[1] * t[1]
  n = n[1]
  for i := 2 to l do
    if( s/n < t )
      t[i] := trunc(t) +1
    else
      t[i] := trunc(t)
    s := s + n[i]*t[i]
    n := n + n[i]
  end
  /* turniri u svakoj od potpopulacija*/
  for i := 1 to l do
    for j := 1 to n[i] do
     a1i' := najbolja jedinka medju t[i] slucajno izabranih iz potpopulacije ai
  end
  return a'
end
```

algoritam 3.7. – Disperzivna fino gradirana turnirska selekcija

S obzirom da je ovaj selekcioni metod ciljno oformljen za primenu u paralelnim EA a da do sada nije teorijski analizirano dejstvo migracije kod paralelnih grubo usitnjenih EA, to će se u ovom poglavlju izvršiti analiza onih aspekata operatora koji nisu povezani sa paralelnim radom algoritma. Ostali aspekti korišćenja ovog operatora će biti analizirani u kasnijim poglavljima.

U slučaju kada sve potpopulacije egzistiraju na istom računaru, vreme izvršavanja za ovaj algoritam je $O(l + n\,t)$. Kako je obično $l << n$ i $t << n$, to se vreme izvršavanja svodi na linearno, tj. $O(n)$.

Stav 3.2.22. Disperzivna fino gradirana turnirska selekcija je invarijantna u odnosu na translaciju i skaliranje.

Dokaz: Tvrđenje stava proizilazi iz činjenice da izbor jedinke zavisi od njenog relativnog položaja u populaciji (odnosno potpopulaciji), tj. od broja jedinki u odnosu na koje je bolje prilagođena, a ne od vrednosti prilagođenosti. Kako ni translacija, ni skaliranje ne menjaju relativnu poziciju jedinki, to je disperzivna fino gradirana turnirska selekcija invarijantna u odnosu na njih ■

Na kraju, ali ne najmanje važno, jedna od prednosti DFGTS je to što se podešavanjem različitih veličina potpopulacija može podesiti da grubo usitnjeni sinhronizovani paralelni EA jako malo vremena provodi u sinhronizaciji (jer su svi čvorovi približno podjednako opterećeni – brži računari će dobiti veću potpopulaciju za rad, a sporiji manju) i da se kod takvog algoritma povežu povoljne karakteristike sinhronizovane verzije (tj. bolja teorijska podloga i veća predvidljivost dobijenih rezultata) sa povoljnim karakteristikama asinhrone verzije (veća efikasnost rezultujućeg algoritma – ne troši se vreme u čekanju).

3.3. Operatori migracije kod grubo usitnjenih paralelnih evolutivnih algoritama

Migracija je, kao što je ranije opisano, razmena jedinki koja je kontrolisana sa nekoliko parametara – brojem migranata (jedinki koje se premeštaju), vremenom koje protiče između migracija, topologijom veza između suseda (npr. zvezda, hiperkocka, mreža, torus...), politika izbora i zamene migranta (npr. najbolji zamenjuje slučajno izabranog, slučajno izabrani zamenjuje najgoreg...), sinhronizovanošću migracije itd.

Broj migranata je jednak proizvodu veličine potpopulacije i nivoa migracije (broja koji određuje koliki će deo potpopulacije da učestvuje u migraciji).

Operatorima migracije se bavio veliki broj autora (videti radove [Muneto93], [Beldin95], [Alba99], [Cantu00]). Jedan od pionira u toj oblasti bila je Tanese, koja je inicirala sistematsko proučavanje migracije i njenih efekata na efikasnost i kvalitet dobijenih rešenja (videti [Tanese87], [Tanese89]). Njen eksperimentalni metod se sastojao u podeli određenog broja jedinki na potpopulacije jednake veličine i variranje nivoa migracije i intervala migracije (tj. broja generacija između dva uzastopna premeštanja jedinki).

Njeni eksperimenti su pokazali da su za dobijanje rešenja koje je istog kvaliteta kako kod sekvencijalnih EA potrebni nivoi migracije koji nisu ekstermno veliki niti suviše mali. Tanese je ispitivala i kompletno izolovane potpopulacije i uočila da su dobijena rešenja inferiorna u odnosu na rešenja koja dobija sekvencijalni EA ili paralelni EA kod kojeg postoji migracija.

Tokom svojih eksperimenata, Tanese je implicitno pretpostavljala da se nekoliko manjih populacija može korektno porediti naspram monolitne populacije. Ukupan broj jedinki je tokom eksperimenata održavan na konstantnom nivou. Međutim, budući da kvalitet rešenja i cena nisu linearno zavisni od veličine populacije, to ni kvalitet rešenja ni cena izvršavanja nisu bili konstanti tokom izvršenja eksperimenata. Dakle, pitanja efikasnosti algoritma i kvaliteta rešenja su odvojeno tretirana, što je dovelo do zaključka da je pri rešavanju pojedinih problema postignuto superlinearno ubrzanje, o čemu je bilo reči u poglavlju 2.6.

3.3.1. Uticaj migracije na intenzitet selekcije kod paralelnih EA

Intenzitet selekcije, onako kako je dat definicijom 3.1.13, meri pritisak prema konvergenciji koji populacija trpi tokom izvršavanja algoritma. Iako je taj pritisak nazvan intenzitet selekcije, selekcija nije jedini operator koji pravi ovaj pritisak – do pritiska prema konvergenciji može doći i primenom operatora migracije.
Sledi stroga definicija intenziteta selekcije koji generiše migracija.

Definicija 3.3.1. Selekciona razlika $I(\Omega)$ za metod migracije Ω je veličina $I(\Omega) = \overline{f^*} - \overline{f}$, tj. razlika između prosečne prilagođenosti populacije posle migracije i prosečne prilagođenosti populacije pre migracije.

Definicija 3.3.2. Prosečna prilagođenost populacije pre migracije, u oznaci $\overline{f}$, je:

$$\overline{f} = \frac{1}{n}\sum_{j=1}^{n} f_j .$$

Što se očekivane prosečne prilagođenosti populacije posle migracije tiče, to je uprosečena suma proseka prilagođenosti migranata i prosečne prilagođenosti jedinki koje su preživele migraciju (tj. nisu bile zamenjene migrantima).

Definicija 3.3.3. Neka δ predstavlja stepen topologije (tj. broj suseda fiksirane potpopulacije) i neka je $m = \rho\, n$ broj migranata iz jedne potpopulacije (ovde je nivo migracije označen sa ρ). Neka je, dalje, $\overline{f_{mig}}$ prosek prilagođensti migranata, a $\overline{f_{sur}}$ prosek prilagođenosti skupa od $n - \delta\, m$ jedinki koje su preživele migraciju. Tada je prosečna prilagođenost populacije posle migracije data sa:

$$\overline{f^*} = \frac{1}{n}\left(\delta\, m\, \overline{f_{mig}} + (n - \delta\, m)\overline{f_{sur}}\right).$$

Primer. U skladu sa prethodnom definicijom, prosečna prilagođenost populacije pre migracije može da se definiše kao:

$$\overline{f} = \frac{1}{n}\left(\delta\, m\, \overline{f} + (n - \delta\, m)\overline{f}\right).$$

Sada se selekciona razlika može izraziti kao suma selekcionih razlika emigranata I_e i selekcionih razlika zameni I_r:

$$I(\Omega) = \overline{f^*} - \overline{f} = I_e + I_r = \frac{1}{n}\delta\, m\left(\overline{f_{mig}} - \overline{f}\right) + \frac{1}{n}(n - \delta\, m)\left(\overline{f_{sur}} - \overline{f}\right)$$

Na ovaj način se izračunavanje selekcione razlike može podeliti u dva nezavisna koraka. Pri izvođenju se pretpostavlja da se za migrante biraju najbolje jedinke iz potpopulacije i da se prilikom migracije uklanja najgora jedinka.

Glavna pretpostavka koja smanjuje opštost, jeste da vrednosti prilagođenosti jedinki f_i predstavljaju realizaciju slučajnih promenljivih F_i koje imaju istu raspodelu. Slučajne promenljive se mogu urediti u neopadajući redosled: $F_{1:n} \le F_{2:n} \le \cdots \le F_{n:n}$.

Bez gubitka opštosti, može se uvesti normalizovana slučajna promenljiva $Z_{i:n} = \dfrac{F_{i:n} - \overline{f}}{\sigma}$.

Prosečna prilagođenost za $m = \rho\, n$ najboljih jedinki koje su izabrane da migriraju iz jedne potpopulacije je

$$\overline{f_{mig}} = \frac{1}{m}\sum_{i=n-m+1}^{n} E\left(F_{i:n}\right)$$

što može biti iskazano pomoću normalizovanih promenljivih

$$\overline{f_{mig}} = \frac{1}{m}\sum_{i=n-m+1}^{n}\left(E(Z_{i:n})\sigma + \overline{f}\right) = \sigma\frac{1}{m}\sum_{i=n-m+1}^{n}E(Z_{i:n}) + \overline{f}\ .$$

Sada je:

$$I_e = \frac{1}{n}\delta m\left(\overline{f_{mig}} - \overline{f}\right) = \frac{1}{n}\sigma\delta\sum_{i=n-m+1}^{n}E(Z_{i:n}).$$

Kako se (što je i elaborirano u poglavlju 3.1) selekciona razlika može iskazati kao proizvod intenziteta selekcije i disperzije prilagođenosti($I = IS\,\sigma$), to je

$$IS_e = \frac{1}{n}\delta\sum_{i=n-m+1}^{n}E(Z_{i:n}).$$

Očekivanje i-tog momenta za uzorak čija je veličina n slučajne promenljive Z je:

$$\mu_{i:n} = E(Z_{i:n}) = n\binom{n-1}{i-1}\int_{-\infty}^{\infty} z\phi(z)\Phi^{i-1}(z)[1-\Phi(z)]^{n-i}\,dz$$

gde $\phi(z)$ i $\Phi(z)$ predstavljaju gustinu raspodele i kumulativnu funkciju raspodele za slučajnu promenljivu Z sa neprekidnom raspodelom. Za približno izračunavanje ove veličine mogu da se koriste tabele odgovarajuće raspodele.

Zamena najgorih jedinki u potpopulaciji sa migrantima dovodi do povećanja prosečne prilagođenosti potpopulacija. Prilagođenost jedinki koje preživljavaju može da se izračuna na sledeći način:

$$\overline{f_{sur}} = \frac{1}{n-\delta m}\sum_{i=\delta n+1}^{n}E(F_{i:n}) = \sigma\frac{1}{n-\delta m}\sum_{i=\delta n+1}^{n}\mu_{i:n} + \overline{f}^{t}$$

Selekciona razlika u zameni je, stoga,

$$I_r = \frac{1}{n}(n-\delta m)\left(\overline{f_{sur}} - \overline{f}\right) = \sigma\frac{1}{n}\sum_{i=\delta n+1}^{n}\mu_{i:n} = \sigma\, IS_r$$

Odavde je $IS_r = \frac{1}{n}\sum_{i=\delta n+1}^{n}\mu_{i:n}$, a vrednost $\mu_{i:n}$ može da se približno odredi na isti način kao u prethodnom izvođenju.

Na kraju, uticaj migracije na selekcioni intenzitet se dobija sabiranjem IS_e i IS_r .

3.3.2. Analiza uticaja nivoa migracije na paralelne EA pomoću Markovljevih lanaca

U poglavlju 2.7 je pomoću Markovljevih lanaca analiziran paralelni EA sa maksimalnim nivoom migracije. Takvi algoritmi se lakše analiziraju, ali se izuzetno retko sreću u praksi. Stoga, rezultati dobijeni analizom takvog algoritma mogu samo predstavljati graničnan slučaj za oblast koja se proučava.

U ovom izvođenju se proširuju rezultati iz poglavlja 2.7 jer se dopuštaju niži nivoi migracije.

Primer. Razmotrimo sledeću situaciju: Neka u datoj epohi dve od tri potpopulacije korektno konvergiraju i neka svaka od potpopulacija šalje ostalim dvema deo veličine $\rho = 0.05$ od populacije. na početku sledeće epohe postoje dve mogućnosti za konkretnu potpopulaciju:

- ako je potpopulacija korektno konvergirala u prethodnoj epohi, tada 95% jedinki sadrži korektne gradivne blokove (90% je već bilo ovde, a 5% je pristiglo od druge korektne potpopulacije)
- ako potpopulacija nije korektno konvergirala u prethodnoj epohi, tada će samo 10% potpopulacije imati korektne gradivne blokove i to je onaj deo populacije koji predstavlja doprinos dobijen od preostale dve.

Da bi se odslikale situacije sliče onoj koja je opisana u prethodnom primeru, Markovljev lanac zahteva dvostruko više stanja nego što je to slučaj u izvođenju u poglavlju 2.7: za svako od korektnih potpopulacija Markovljev lanac zahteva dva stanja – jedno da predstavlja slučaj kada najveći deo potpopulacije sadrži korektne gradivne blokove (zato što je ta potpopulacija korektno konvergirala u prethodnoj epohi) i drugo da predstalja situaciju kada najveći deo potpopulacije ne sadrži korektne gradivne blokove. Uobičajeno je da se stanja označe brojevima od 0 do $2n_d - 1$ i urede tako da stanja od 0 do $n_d - 1$ označavaju situacije kada je

većina potpopulacije nekorektna, a stanja od n_d do $2n_d - 1$ označavaju sitacije kada je većina potpopulacije korektna.

Kao i u poglavlju 2.7, postoji jedno apsorbirajuće stanje za slučaj kada sve potpopulacije konvergiraju nekorektno (tzv. stanje 0) i jedno apsorbirajuće stanje za slučaj kada sve potpopulacije konvergiraju korektno (stanje $2n_d - 1$). Ostala stanja sistema su prelazna stanja.

U ovom scenariju, inicijalna raspodela će biti $V_1 = \{V_1(i)\}_{i=\overline{1,2n_d-1}}$ i elementi vektora su:

$$V_1(i) = \begin{cases} \binom{n_d - 1}{i} P_{bb}(x_0)^i \left(1 - P_{bb}(x_0)\right)^{n_d - i} & , \quad i < n_d \\ \binom{n_d - 1}{i - n_d} P_{bb}(x_0)^{i - n_d + 1} \left(1 - P_{bb}(x_0)\right)^{2n_d - i - 1} & , \quad i \geq n_d \end{cases}$$

Matrica prelaska je:

$$p_{i,j} = \begin{cases} \binom{n_d - 1}{j} \left(P_{bb}(\chi_i)\right)^j \left(1 - P_{bb}(\chi_i)\right)^{n_d - j} & , \quad i < n_d \\ \binom{n_d - 1}{j - n_d} \left(P_{bb}(\chi_i)\right)^{j - n_d + 1} \left(1 - P_{bb}(\chi_i)\right)^{2n_d - j - 1} & , \quad i \geq n_d \end{cases}$$

Pri tome je χ_i polazna tačka za slučajno kretanje kod svakog stanja i, a njena vrednost zavisi od nivoa migracije ρ i od toga koliko je potpopulacija korektno konvrgiralo.

Dalje izvođenje je potpuno isto kao izvođenje koje je opisano u poglavlju 2.7, samo što dobijeni rezultati imaju mnogo dužu u mnogo neelegantniju formulaciju.

4. EA SISTEMI

4.1. Razvijeni EA sistemi

U ovom poglavlju se opisuju EA sistemi koje je razvio autor kao podršku (potporu) ovom radu. Tako razvijeni EA sistemi su dalje korišćeni tokom empirijskih proučavanja EA i za rešavanje raznovrsnih problema, najčešće problema iz oblasti operacionih istraživanja.
Treba istaći da je na osnovi u prethodno pomenutim sistemima bilo praktično nemoguće napraviti nadogradnju u pravcu web servisa i Internet procesiranja. Zbog toga je jedan pravac autorovog delovanja išao prema proširenju i nadogradnji ovih EA sistema novodizajniranim operatorima, dok se drugi pravac usmerio prema web servisima.

4.1.1. GANP

Jedna grupa istraživača (kojoj pripada i autor ovog rada) je razvila softver (tj. algoritamski orjentisani softverski sistem) radi ispitivanja karakteristika raznih GA operatora i radi rešavanja problema pomoću GA. S obzirom da se razvijeni softver u najvećoj meri koristio za rešavanje instanci NP teških problema, on je dobio ime GANP. Softver je napravljen u ANSI C programskom jeziku.
U jezgru algoritamski orjentisanog GANP sistema su implementirane ([Kratic97a], [Filipo98]):

- proporcionalna selekcija, linearno rangiranje, turnirska selekcija i, naravno, fino gradirana turnirska selekcija;
- jednopoziciono, dvopoziciono, višepoziciono i uniformno ukrštanje;
- prosta mutacija realizovana po definiciji (izvršava se relativno sporije), prosta mutacija primenom Puasonove raspodele i prosta mutacija primenom normalne raspodele;
- linearno skaliranje, skaliranje oko srednje vrednosti, skaliranje u jedinični interval, sigma isecanje (eng. sigma truncation); sva skaliranja su implementirana kako za probleme u kojima se traži maksimum, tako i za probleme u kojima se traži minimum;
- generacijska politika zamene jedinki, politika zamene nazvana stabilno stanje i elitna strategija;
- raznovrsni kriterijumi za završetak EA;
- određivanje maksimalne i minimalne prilagođenosti populacije, off-line i on-line performanse, te broja izgubljenih gena u svakoj iteraciji algoritma (izračunavanje i prikaz vrednosti se vrši ukoliko je korisnik iskazao potrebu za tim);
- određivanje prosečne prilagođenosti i disperzije populacije, selekcionog intenziteta i selekcione disperzije, te gubitka raznovrsnosti u svakoj iteraciji algoritma (pri čemu se izračunavanje i prikaz vrednosti vrši ukoliko su postavljene odgovarajuće vrednosti u konfiguracionu datoteku);
- zadavanje parametara EA pomoću konfiguracione datoteke;
- mogućnost korišćenja unapred datog optimalnog rešenja za analizu izvršavanja algoritma.

GANP sistem se i dalje razvija i nadograđuje, u skladu sa potrebama autora. Tako je, sa stanovišta selekcije (a ista je situsacija i kod drugih aspekata algoritma), u okviru GANP implementiran svaki od prethodno opisanih selekcionih operatora. Ovaj sistem je dosad korišćen za rešavanja raznovrsnih problema i za eksperimente opisane u sledećim radovima: [Filipo96], [Filipo96a], [Filipo97a], [Filipo03], [Kratic96], [Kratic97], [Kratic97a], [Vugdel96]. Pažljivo razmatranje tokom dizajniranja GANP sistema, naročito njegovog jezgra, omogućilo je da se sva dosadašnja proširenja i poboljšanja (bez obzira koliko "egzotična" bila) uklope u sistem na izuzetno jednostavan i lak način.
Karakteristike implementacije jezgra GANP sistema su:

- formira se struktura koja sadrži sve informacije o EA;

- izbor raznih varijanti operatora selekcije, ukrštanja i mutacije, kao i izbor funkcije prilagođenosti i kriterijuma za kraj izvršavanja EA vrši se pomoću konfiguracione datoteke;
- koriste se funkcijski pokazivači, što očuvava brzinu izvršavanja algoritma, uz mogućnost variranja raznih varijanti operatora promenom u konfiguracionoj datoteci (nema potrebe za ponovnim prevođenjem i povezivanjem programa);
- način čitanja podataka iz konfiguracione datoteke je veoma fleksibilan; naime, konfiguraciona datoteka ima strukturu kao inicijalizacione datoteke kod Windows-a;
- jezgro GANP sistema je moguće lako nadograditi u skladu sa nekim specifičnijim potrebama (meta EA, paralelni EA, itd.);
- koristi se generator slučajnih brojeva dat u izvornom kodu, što doprinosi nezavisnosti koda od platforme na kojoj se izvršava, omogućuje determinisano testiranje programa i jednake rezultate izvršavanja na svim platformama;
- jezgro GANP sistema se lako dopunjava funkcijama koje zavise od prirode problema.

Funkcije zavisne od prirode problema, neophodne za uspešno izvršavanje algoritma, su:

- inicijalizacija podataka zavisnih od prirode problema;
- učitavanje i štampanje podataka karakterističnih za dati problem;
- konverzija genetskog koda jedinke u argumente problema;
- računanje funkcije objekcije na osnovu argumenata problema.

Jezgro GANP sistema omogućava lako uključivanje i drugih, neobaveznih, funkcija zavisnih od prirode problema, kao što su:

- heuristike za dobijanje nekih jedinki početne populacije (u slučaju da se date jedinke ne dobijaju na slučajan način);
- heuristike za poboljšavanje vrednosti nekih jedinki (u slučaju da ne postoji biunivokna korespodencija između prostora niski i prostora pretraživanja);
- operatori ukrštanja i mutacije zavisni od prirode problema.

Posle zajedničkog koncipiranja centralne strukture i ključnih delova algoritma, svako od autora se posvetio dizajnu pojedinih delova jezgra, kao i čitavog sistema.

Arhitektura sistema i način njegove implementacije su orjentisani prema efikasnosti izvršavanja EA, tako da brzina izvršavanja predstavlja veliku komparativnu prednost ovog softverskog sistema u odnosu na neke druge. Rezultati ukazuju da ovaj pristup i ovaj sistem predstavljaju izuzetno uspešnu platformu za eksperimentisanje i analizu radi kreiranja GA za efikasno rešavanje NP -teških problema.

S obzirom da je prošlo već više od deset godina od dizajna ovog sistema, moglo se očekivati da se ovakav softver ne može potpuno uklopiti u novonastale prodore i trendove koji su nastali u polju EA (kao što je GP, napredna primena EA u veštačkoj inteligenciji itd.), u polju distribuisanog izračunavanja i mreža (web servisi, procesiranje korišćenjem Interneta itd.) i u polju softverskog inženjerstva (šabloni obrazaca, višeslojne aplikacije, kolaboracija između aplikacija itd.)

4.1.2. PGANP

Ovde se radi o nadogradnji i proširenju prethodno opisanog GANP sistema, dodavanjem paralelizacije. Autor Jozef Kratica je za komunikaciju između čvorova koristio MPI standard – što omogućuje prenosivost sistema na razne platforme. Arhitektura ovog paralenog EA softverskog sistema je zadata od strane korisnika i ona, kao što je već pokazano, znatno utiče na performanse paralelnog EA. Ukoliko se svi procesi koji se izvršavaju na paralelnom računaru posmatraju kao čvorovi, a procesi koji razmenjuju jedinke se obeleže kao grane između prethodno uočenih čvorova, time se definiše graf razmene jedinki tokom izvršavanja paralelnog EA.

Postupak izvršavanja paralelnog algoritma u ovom EA sistemu je dat sledećim algoritmom (videti [Kratic00]), pri čemu prefiksi M, S i A redom označavaju da se odgovarajuća funkcija izvršava samo na glavnom procesu (eng. main - glavni), na ostalim procesima (eng. slave - rob) ili na svim procesima(eng. all - svi):

```
Paraleni_Evolutivni_Algoritam:
begin
  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() )
  begin
    gener := gener+1;
    Prijem_Jedinki_od_Suseda();
    for (i=Nelite+1; i<Npop; i++)
      if (! Primljen_od_Suseda(i))
        pi = Vrednosna_Funkcija(i);
    Računanje_Funkcije_Prilagođenosti();
    if(gener mod frazmene = 0)
    begin
      Izaberi_Jedinke_za_Slanje();
      Pošalji_Jedinke_Susedima();
    end
    Prijem_Poruke_O_Kraju_GA();
    if( not Kraj_Lokalnog_GA())
    begin
      Selekcija();
      Ukrštanje();
      Mutacija();
    end
    else
      Slanje_Poruke_O_Kraju_Lokalnog_GA();
  end
  M_Štampanje_Rešenja();
end
```

algoritam 4.1. – Opsta shema paralenog EA

Za sve modele paralelizacije je značajan način prosleđivanja EA i problema od osnovnog procesa ka svim ostalim procesima. Ovakav način prosleđivanja se može formalno opisati nekom drvoidnom strukturom, tzv. drvo prosleđivanja. U takvoj formalizaciji, za svaki proces se definišu dva pojma: sused roditelj i susedi potomci. Svaki proces, osim osnovnog, ima tačno jednog suseda roditelja od koga na početku dobija EA i problem 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.

U distribuiranom modelu, koji je (kako je već istaknuto) primarna oblast istraživanja ovog rada, jedinke se razmenjuju relativno retko, najčešće samo jednom u nekoliko generacija, i to samo između potpopulacija koje se izvršavaju na susednim procesima. Razlozi za takvu odluku su podrobnije objašnjeni u poglavlju rada koje se bavi operatorom migracije. Zbog toga, u PGANP 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. Dobar izbor učestanosti razmene jedinki, omogućuje dobar kompromis između istraživanja i eksploatacije. To doprinosi punom korišćenju prednosti paralelizacije, koje se ogledaju u boljem kvalitetu rešenja, uz istovremeno kraće vreme izvršavanja, u odnosu na sekvencijalni algoritam.

Po izvršenju EA na lokalnoj potpopulaciji potrebno je proslediti i objediniti dobijena rešenja. Programska struktura 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 EA dopunjuje rezultatima dobijenim od suseda potomaka, i na taj način se formira rešenje koje se dalje prosleđuje susedu roditelju.

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

U slučaju lokalnog kriterijuma završetka, dati proces prekida izvršavanje lokalnog EA, 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.

4.2. EA Web servis i aplikacije koje ga koriste

4.2.1. Motivacija

Motivacija za ovakav pristup je korišćenje računarskih resursa dostupnih na Internet-u kako bi se obezbedila računarska snaga neophodna za rešavanje različitih problema. Web servis i aplikacije su razvijene na Microsoft .NET platformi, korišćenjem programskog jezika C#. Ovakav pristup podržava različite tipove evolutivnih algoritama (sa potpuno različitim vrstama reprezentacije, ukrštanja, selekcije, mutacije, politike zamene jedinki, različitim mrežnim topologijama, itd.). U osmišljavanju sistema je korišćena objektno orjentisana paradigma i obrasci dizajna, a u realizaciji osobine, interfejsi i indekseri. Stoga uvođenje i implementacija novih modifikacija evolutivnog algoritma zahteva minimalne izmene u kodu.
U dizajnu velikih i složenih softverskih sistema, izuzetno je veliki značaj pažljive analize i dizajna, a takođe i potreba da se u taj proces uključe nove metodologije i moderne tehnologije. Zato su, pre opisa samog sistema, opisani tehnologije i metodologije koje su korišćene: .NET okvir i jezik C#, jezik modeliranja UML i obrasci dizajna, XML, web servisi i SOAP.

4.2.2. .NET Okvir

.NET okvir (eng. .NET framework) je nova računarska platforma dizajnirana od strane Microsoft-a, da bi, po rečima kreatora, uprostila razvoj aplikacija u široko distribuisanom Internet okruženju.
.NET okvir je dizajniran sa sledećim namerama (videti [Bott02]):

- Zajedničko izvršavanje. .NET okvir obezbeđuje metode za zajedničko izvršavanje novog koda i prethodno kreiranih COM biblioteka.
- Zajedničko okruženje za izvršavanje. Slično kao kod Jave, programski jezici se kod .NET-a prevode na međujezik koji se zove zajednički međujezik (eng. Common Intermediate Language - CIL) Microsoft-ova implementacija CIL-a ima naziv Microsoft-ov međujezik (eng. Microsoft Intermediate Language - MSIL). Instrukcije MSIL-a se, za razliku od Jave, ne interpretiraju, već se prevode u mašinski kod na način „u pravo vreme“ (eng. just in time compilation - JIT). Obuhvatni naziv za celokupnu strukturu je zajednička jezička infrastruktura (eng. Common Language Infrastructure - CLI), a Microsoft-ova implementacija CLI-a se naziva zajednički jezički izvršni sistem (eng. Common Language Runtime CLR). CLR može biti posmatran kao agent koji upravlja kodom tokom izvršenja, obezbeđujući osnovne servise kao što su upravljanje memorijom, upravljanje nitima, pri čemu se istovremeno obezbeđuje striktna sigurnost koda. Koncept upravljanja kodom je fundamentalni princip modula za izvršavanje. Kod koji se izvršava pomoću izvršnog modula naziva se upravljani kod, a kod čije izvršavanje zaobilazi izvršni modul se naziva neupravljani kod.
- Nezavisnost od jezika. Za razliku od Java platforme i od COM-a, .NET okvir uvodi zajednički sistem tipova (eng. Common Type System - CTS). CTS specifikacija definiše sve moguće tipove podataka i programske konstrukcije koje podržava CLR i definiše kakvu međusobnu interakciju oni mogu imati. Zbog ove osobine, .NET okvir podržava razvoj u većem broju programskih jezika, što će u daljem tekstu biti detaljnije opisano.
- Osnovna biblioteka klasa. Osnovna biblioteka klasa (eng. Base Class Library - BCL) koja se u literaturi još naziva i biblioteka klasa za okvir (eng. Framework Class Library - FCL) je biblioteka sa tipovima koji su na raspolaganju svim .NET programskim jezicima. BCL sadrži klase preko kojih su realizovane osnovne funkcionalnosti, kao što je čitanje iz datoteke i upis u datoteku, crtanje

grafičkih objekata, interakcija sa bazama podataka, manipulacija sa XML dokumentima itd. Najkraće, biblioteka klasa .NET okvira je obimna objektno-orjentisana kolekcija klasa raspoloživih za ponovno korišćenje, koje se koriste u razvoju različitih tipova aplikacija, počev od klasičnih aplikacija koje se izvršavaju u komandnoj liniji, preko GUI aplikacija, pa sve do aplikacija zasnovanih na ASP.NET-u i web servisima.

- Olakšan razvoj. Instalacija i održavanje Windows aplikacija ranije nije bilo lako. Problemi kao što je „DLL pakao" koji je zahtevao ručna direktna podešavanja registra, a čak i tada bio teško rešiv je sa .NET-om skoro u potpunosti eliminisan.
- Sigurnost. Kako je Internet postajao sve više integrisan u svakodnevni rad i programiranje, to je pitanje kom kodu se može verovati sve više dobijalo na značaju. Da bi razrešio ovo pitanje, .NET okvir je uveo sistem sigurnog pristupa kodu (eng. code access security). O sigurnosnim mehanizmima .NET-a i njihovoj nadogradnji više informacija u [Meier02].

Zahvaljujući svojoj arhitekturi, činjenici da koristi zajednički međujezik i zajednički sistem tipova, .NET okvir je i nezavistan od platforme. Naime, pored Windows OS-a, sam Microsoft obezbeđuje verzije CLI za FreeBSD i za Mac OS X. Nadalje, s obzirom da je zajednička jezička infrastruktura ECMA standard, postoji veći broj projekata sa otvorenim izvornim kodom (eng. open source) koji obezbeđuju podršku za dodatne platforme. Najpoznatiji takvi projekti su Mono, DotGNU i Portable .NET.

Pored standardnog .NET okvira, Microsoft obezbeđuje i .NET kompaktni okvir – manje zahtevnu verziju koja je pogodna za korišćenje na PDA-ovima, organizatorima, „pametnim" telefonima itd.

Mada .NET sadrži i kompajlere i druge alate za razvoj aplikacija koji su dostupni programerima, Microsoft nudi i dodatne aplikacije koje olakšavaju proces razvoja – najčešće se radi o integrisanom razvojnom okruženju Visual Studio .NET. Takođe postoje i alati za razvoj .NET aplikacija sa otvorenim izvornim kodom, kao što je SharpDevelop.

Avgusta 2000. kompanije Microsoft, Hewlett-Packard i Intel ulažu napore u standardizaciju CLI i C#. Uloženi napori dovode do rezultata pa obe specifikacije bivaju u decembru 2002. prihvaćeni ECMA standardi (ECMA 335 i ECMA 334). Aprila 2003 i ISO ih standardizuje (ISO/IEC 23271 i ISO/IEC 23270).

.NET tehnologije

Windows forme

Windows forme su deo .NET okvira koji obezbeđuje omotače za elemente koji se nalaze unutar postojećeg Win32 API-ja.

ASP.NET

ASP.NET, Microsoft-ova zamena za tehnologiju web programiranja aktivne serverske strane (eng. Active Server Pages - ASP) ima .NET biblioteku klasa. ASP.NET je, kao i sve ostale .NET tehnologije, nezavistan od jezika. Detaljnije informacije o ASP .NET mogu se naći u [Ander02].

ADO .NET

ADO .NET je nova verzija aktivnih objekata sa podacima (eng Active Data Object – ADO) i služi za olakšavanje rada sa podacima. Rašireni jezik za označavanje XML igra veoma značajnu ulogu kod ADO .NET-a, jer se u ADO .NET-u podaci sa kojima se radi faktički čuvaju kao XML dokumenti. Za detaljno razmatranje ADO .NET-a konsultovati npr. knjigu [Joshi02].

Udaljeni .NET

Infrastruktura udaljenog .NET-a (eng. :NET Remoting) je apstraktan pristup međuprocesorskoj komunikaciji. Udaljeni .NET obezbeđuje bogat i proširiv okvir za laku i efikasnu međusobnu komunikaciju objekata koji "žive" u različitim aplikativnim domenima, u različitim procesima, ili na različitim računarima. Udaljeni .NET nudi jednostavan a moćan programski model i podršku za izvršavanje kako bi ove interakcije ostale transparentne.

Web servisi

XML web servisi su osnovni gradivni blokovi pri premeštanju prema distribuisanom izračunavanju na Internet-u. Otvoreni standardi i usmeravanje prema komunikaciji i saradnji među ljudima i aplikacijama je kreiralo okruženje u kome web servisi postaju platforma za integraciju aplikacija. Aplikacije se konstruišu

korišćenjem više web servisa iz različitih izvora, tako da svi oni rade zajedno bez obzira na to gde se nalaze ili kako su implementirani.
.NET okvir poštuje sve komunikacione standarde na kojima se zasnivaju web servisi, što omogućuje interoperabilnost sistema. Drugim rečima, klase iz skupa koje obezbeđuje .NET okvir poštuju komunikacione standarde (SOAP, XML, WSDL), pa se programer može fokusirati na logiku servisa koji implementira, bez potrebe da brine o komunikacionoj infrastrukturi koju zahteva razvoj distribuisanog softver-a. Detaljni opis .NET okvira, arhitekture, metodologije programiranja i debagiranja sadrži se u knjigama [Ritche02] i [Robbin03].

.NET jezici

S obzirom na prethodno opisanu arhitekturu .NET okvira (CLI, MSLI, CLR) i na to da je ona zasnovana na međujeziku, svaki izvorni kod ili kompajler koji emituje MSIL može da se koristi za kreiranje sklopova (eng. assemblies) , a te sklopove potom može da izvršava CLR. CLR podržava i objektno-orjentisane i proceduralne jezike.
Mada trenutno postoji pedesetak jezika koji imaju prevodilac za .NET okvir, samo mali podskup njih je široko korišćen i podržan od Microsoft-a.
Jezici koje podržava Microsoft:

- C# - Jezik koji se pojavio sa .NET-om i koji ima brojne sličnosti sa jezicima C++ i Java. Kako je C# korišćen za programiranje EA sistema EaWebService, on će kasnije biti detaljno opisan.
- Visual Basic .NET – kompletno redizajnirana verzija jezika Visual Basic, za .NET okvir.
- C++ - upravljana verzija jezika C++.
- J# - Jezik koji olakšava prelaz sa Jave i J++ na .NET platformu.
- JScript.NET – verzija skript jezika JScript koja se prevodi.

Alternativni jezici:

- A#, Ada
- APL
- Boo, Python
- COBOL
- Delphi 8 i Delphi 2005
- Eiffel
- F#, iz porodice ML programskih jezika
- Forth
- FORTRAN
- IronPython, Python
- Lisp
- IKVM, Java
- Mercury
- Oberon
- Perl
- Phalanger, PHP
- RPG
- Smalltalk

Verzije

- .NET okvir 1.0 (puni broj je 1.0.3705). To je prvi .NET okvir, koji je isporučen 2002. godine. On je dostupan u dva oblika: kao paket za izvršavanje (tj. redistribuciju) ili kao alat za softverski razvoj (eng. Software Development Kit - SDK). Taj okvir je bio uključen u prvu verziju Visual Studio .NET, (ta verzija programa Visual Studio se označava sa Visual Studio .NET 2002).
- .NET okvir 1.1 (puni broj je 1.1.4322). Ovo je prva veća nadogradnja .NET okvira, isporučena u drugoj polovini 2003. I ova verzija se može dobiti samostalno, bilo kao paket za redistribuciju ili kao SDK. Nadalje, ovaj okvir je uključen u drugu verziju Visual Studio :NET (tzv. Visual Studio .NET 2003), a takođe je uključen i kao deo Microsoft-ovog operativnog sistema Windows Server 2003. najznačajnije razlike između ovog okvira i njegovog prethodnika su:
  - Ugrađena podrška za mobilne ASP .NET kontrole (što je ranije postojalo kao odvojen dodatak, koji se posebno uključivao).
  - Promene u sistemu sigurnosti - kod Windows formi gde se dopušta izvršavanje sa Interneta i omogućavanje da se u ASP. NET aplikacijama koristi sistem sigurnog pristupa kodu.
  - Ugrađena podrška za mobilne ODBC i Oracle baze podataka (što je ranije postojalo kao odvojen dodatak, koji se posebno uključivao).

- .NET kompaktni okvir – predstavlja oslabljenu verziju .NET okvira, pogodnu za izvršavanje na kompaktnim uređajima.
- Podrška za IPv6 (eng. Internet Protocol version 6).
- Veliki broj promena u API-ju (eng. Application Programming Interface - API)

- .NET okvir 2.0 (ili preciznije 2.0.50727). Ova verzija .NET okvira je isporučena krajem oktobra 2005. I nadalje se paket za redistribuciju .NET 2.0 i SDK za .NET okvir 2.0 mogu besplatno dovući sa web prezentacije proizvođača. .NET 2.0 je uključen i u Visual Studio 2005 i u SQL Server 2005. U periodu između isporučivanja verzija 1.1 i 2.0 napravljena su sledeća poboljšanja (pobrojana su samo najvažnija među njima):
  - Puna 64-bitna podrška za x64 i za IA64 hardverske platforme.
  - Veliki broj promena u API-ju.
  - Smanjena potreba za kodiranjem – manji broj linija koda proizvodi veću funkcionalnost.
  - Podrška za uopštavače (eng. generics) je prebačena direktno u CLR.

4.2.3. Jezik C#

C# (izgovara se „si šarp“, bukvalno prevedeno to bi bilo „povišeno C“ ili „C sa povisilicom“) je programski jezik dizajniran za gradnju širokog opsega aplikacija koje se izvršavaju u .NET okviru. Na sam dizajn jezika C# je, pored C++, najviše uticala Java, mada je uočljiv i uticaj Smalltalk-a i Pascal-a. C# is jednostavan, moderan, objektno-orjentisan jezik (videti [ReyHae02]). C# kod se prevodi do nivoa upravljanog koda, što znači da koristi servise CLR. Prednost ovakvog pristupa uključuje interoperabilnost sa drugim programskim jezicima, ugrađeni mehanizam sakupljanja otpadaka (eng. garbage collection), poboljšanu sigurnost i pojačanu podršku za upravljanje verzijama.
C# ima sledeće karakteristike:

- Podrška za punu interoperabilnost sa COM+ 1.0 i sa servisima .NET okvira, uz blizak pristup zasnovan na bibliotekama.
- XML podrška za interakciju Web-baziranih komponenti.
- Upravljanje verzijama, što olakšava administraciju, implementaciju i isporuku.
- U sam jezik su ugrađeni atributi, osobine, interfejsi, indekseri, refleksija, događaji, a u biblioteku klase i interfejsi za olakšanje višenitnog programiranja.

Jezik C# je, na neki način, programski jezik koji najdirektnije odslikava .NET okvir na kom se, kao što je već istaknuto, izvršavaju svi .NET programi. On jako zavisi od tog okvira – njegovi prosti tipovi su u suštini odgovarajući .NET tipovi, on ima sakupljač otpadaka, a mnoge njegove apstrakcije, kao što su klase, interfejsi, delegati, izuzetci (videti [Hejlsb03]) itd. eksplicitno izlažu osobine .NET izvršnog okruženja. Detaljne informacije o ovom jeziku se, pored ostale mnogobrojne literature, mogu naći u [Archer01] i [Archer02].
U poređenju sa programskim jezicima C i C++, ovaj jezik je negde ograničen a negde proširen na raznovrsne načine (pri čemu lista koja sledi sadrži samo najznačajnija ograničeja i proširenja):

- Klasični, sirovi pokazivači (u smislu adrese prostora u memoriji) mogu biti korišćeni samo u tzv. nesigurnom modu. Najveći broj pristupa objektima je rađen preko tzv. sigurnih referenci – koje ne mogu da postanu nekorektne. Kod najvećeg dela aritmetičkih operacija se proverava da li dolazi do prekoračenja. Pokazivači mogu biti samo tzv. vrednosnog tipa, a za objekte nad kojima radi sakupljač otpadaka je moguće samo da se referiše na njih.
- Objekti se ne oslobađaju eksplicitno, već se nepotrebni objekti (oni na koje niko ne referiše) recikliraju od strane sakupljača otpadaka, kad se započne taj proces. Za one objekte koji predstavljaju resurse nad kojima se ne upravlja automatski, programer može da (korišćenjem interfejsa `IDisposable`) definiše kako da se ti resursi oslobode.
- Postoji samo jednostruko nasleđivanje (kao u Javi i Smalltalk-u). Isto kao i u Javi, jedna klasa može implementirati veći broj interfejsa.
- C# je mnogo sigurniji pri konverziji tipova od C++. Jedine podrazumevane tj. implicitne konverzije su konverzije koje čuvaju tip (konverzija u kojoj se vrednosnom tipu proširuje opseg i konverzija iz izvedenog tipa u osnovni, tj. roditeljski tip). Nema implicitne konverzije između logičkih i

celobrojnih vrednosti, niti između enumerisanih i celobrojnih vrednosti, nema pokazivača na `void` (mada se referenca na `Objekat` može posmatrati i na taj način).
- Razlikuje se sintaksa pri deklarisanju niza (npr. `int a[] = new int[5]` umesto C-ovskog `int a[5]`).
- Enumerisane vrednosti se nalaze u sopstvenom prostoru imena (eng. namespace).
- Nema standardne biblioteke šablona (eng. Standard Template Library - STL), ali su kod C# 2.0 dodati uopštavači – koji su se pokazali toliko korisnim da je našto kasnije sličnu stvar Sun uveo u novu verziju Jave.
- Na raspolaganju su i osobine (eng. properties) što dopušta pozivanje metoda sintaksom koja se obično koristi za pristup podacima u okviru objekta.
- Uvedeni su i indekseri, što omogućuje programeru da elementima kolekcija snabdevenih indekserom pristupa kao da se radi o nizu.
- Dostupna je potpuna refleksija – može se pisati program koji tokom izvršavanja pita o objektima koji su kreirani i koji žive tokom izvršavanja programa, kreira nove objekte itd.

Prethodno pominjani C# standard (ECMA 334, odnosno ISO/IEC 23270) detaljno opisuje minimalni skup tipova i biblioteka klasa koje treba da su dostupne kompajleru. Taj skup predstavlja osnovni zahtev, a najveći broj implementacija se isporučuje sa većim skupom biblioteka.
BCL (odnosno FCL) u .NET okviru predstavlja biblioteku klasa koja se može koristiti iz ma kog .NET jezika za realizaciju zadataka, počev od proste reprezentacije podataka i manipulacije niskama, preko generisanja dinamičkih web strana, XML procesiranja i refleksije. Kod koji je na raspolaganju je organizovan u hijerarhiju prostora imena, gde su klase koje imaju sličnu funkciju smeštene u jedan prostor imena (npr. `System.Drawing` za grafiku, `System.Collection` za strukture podataka, `System.Windows.Forms` za forme itd.) .
Dalji, tj. opštiji nivo organizacije je obezbeđen pomoću sklopa. Sklop je jedna datoteka (ili više datoteka koje su međusobno povezane korišćenjem alata `al.exe`) koje mogu sadržati veći broj prostora imena i objekata. Programi kojima trebaju klase iz biblioteka da bi izvršili konkretnu funkciju tada referišu sklopove kao što je System.Drawing.dll ili System.Windows.Forms, kao i biblioteku koda (u datoteci mscorlib.dll kod Microsoft-ove implementacije).
Iz očiglednih razloga, mnogi Microsoft-ovi potezi i inicijative su privlačile veliku pažnju, pa ni C# nije u tome izuzetak. S obzirom na veliku bliskost između C# i komercijalne organizacije, otvorila se velika i oštra diskusija o legitimnosti standardizacije jezika C#, o sličnosti C# i Jave, o njenoj budućnosti, itd. U isto vreme, jezik je hvaljen zbog čistog dizajna, zbog gramatike koja je bliska programeru i zbog toga što je kod izvesnih vrsta aplikacija vreme razvoja na ovom novom jeziku mnogostruko skraćeno.
Za razliku od Microsoft Visual Basic-a, pa na neki način i Jave, sa kojima proizvođač neograničeno raspolaže, kod C# je Microsoft odlučio da otvori C# prema procesu standardizacije. Iako postoji veći broj implementacija za CLI i C#, mora se istaći da je Microsoft osnovna pokretačka snaga razvoja ovog jezika i da se, u krajnjoj instanci, u Redmondu odlučuje u kom će se pravcu C# dalje razvijati. Pored toga, Microsoft je jasno istakao da C#, kao i ostali .NET jezici, imaju važnu ulogu u njihovoj softverskoj strategiji i to kako kod internog razvoja aplikacija, tako i za spoljašnje korišćenje. Microsoft ima veoma aktivnu ulogu u promovisanju ovog jezika kao dela svoje celokupne poslovne strategije. Detaljne informacije o jeziku i njegovim konstrukcijama mogu da se nađu u [Lippma02], [Fergu02], [Robins02] i [Teilh04].

4.2.4. XML

XML tj. proširiv jezik za označavanje (eng. eXtensible Markup Language) je jezik za označavanje preporučen od W3C (eng. World Wide Web Consortium). On služi za kreiranje specijalizovanih jezika za označavanje, sposobnih da opišu mnogobrojne, veoma raznovrsne vrste podataka. XML predstavlja način opisivanja strukturisanih podataka (videti [W3C] i [W3CXML]). Njegova namena je da omogući razmenu podataka kroz raznovrsne sisteme, naročito sisteme povezane na Internet. Jezici koji su zasnovani na XML-u (kao što su GML, MathML, PML, XHTML, SVG, MusicXML itd.) su definisani na formalan način, čime dopuštaju programima da validiraju i modifikuju dokumente koji su zapisani u tim jezicima, a da programi nemaju prethodno znanje o dokumentima nad kojima operišu.

XML je dizajnirala radna grupa od 11 članova, koju je podržavalo oko 150 članova interesne grupe. Tehnička debata se vodila preko diskusione liste interesne grupe, a otvorena pitanja su razrešavana konsenzusom ili prostom većinom (u situacijama kada se nije moglo doći do konsenzusa). Radna grupa za XML se, tokom svog rada, nikada nije našla na okupu, već su dogovaranja i zajednički rad realizovani kombinacijom elektronske pošte i nedeljnih telekonferencija. Glave odluke oko dizajna novog jezika su donesene između jula i novembra 1996, a februara 1998 usaglašeni rezultat rada radne grupe postaje W3C preporuka – XML 1.0.

XML 1.0 je postigao ciljeve koji su radna grupa i grupa za podršku postavljali pred njega – primenljivost kod Internet-a, opšta primenljivost, kompatibilnost sa SGML-om, primena pri lakšem razvoju softvera, minimizacija opcionih karakteristika, formalnost, konzistentnost i lakoću kreiranja dokumenata.

XML obezbeđuje da se pomoću teksta opišu i modifikuju drvoidne strukture podataka. Na najnižem nivou, sve informacije se prikazuju kao tekst ispresecan konstrukcijama označavanja, koje ukazuju na strukturisanje informacija u hijerarhiju znakovnih podataka, elemenata koji služe kao kontejneri i atributa ovih elemenata (što donekle podseća na S-izraze u LISP-u).

Pre razvoja XML-a, bilo je veoma malo jezika za opis podataka opšte namene, koji su se dobro slagali sa Internet protokolima i koji su bili laki za rad i za učenje. U stvari, do nastanka XML-a, najveći broj formata za razmenu podataka je bio vlasništvo autora (autor ga je mogao menjati po svom nahođenju), bio je u binarnom obliku (pa se nije mogao lako deliti među različitim aplikacijama na različitim platformama i nije se mogao gledati niti menjati pomoću tekst editora).

Obezbeđivanjem otvorenosti za imena, hijerarhije i za značenje elemenata i atributa, kao i obezbeđivanjem da svi prethodno pobrojani objekti mogu biti definisani prilagodljivim shemama, XML je dopustio da se kreiraju novi, prilagođeni jezici za označavanje koji su zasnovani na XML-u. Opšta sintaksa ovakvih jezika je dosta rigidna – dokumenti pisani u tim jezicima moraju zadovoljavati opšta pravila XML-a, čime je obezbeđeno da softver koji čita XML može pročitati i razumeti redosled među informacijama koje su sadržane u takvom dokumentu.

XML ne propisuje nikakva ograničenja o tome kako će da bude korišćen. Iako je u svojim temeljima XML zasnovan na tekstu, postoji i veći broj programa gde se XML podaci apstrahuju u druge, bogatije formate, obično uz korišćenje shema orjentisanih na tipove podataka i uz korišćenje objektno-orjentisane paradigme (gde se dokumentima manipuliše kao objektima).

Kao što je već istaknuto, XML koristi skup tagova za izdvajanje elemenata u okviru podatka. Svaki elemenat enkapsulira deo podatka koji može biti veoma jednostavan ili vrlo složen.

XML je nezavistan od platforme, jednostavan i široko prihvaćen standard. On razdvaja podatke od prezentacije, što omogućuje integraciju podataka iz različitih izvora.

Sledeće osobine XML-a čine ga prilagođenim za prenos podataka:

- to je format koji sa lakoćom mogu čitati i ljudi i mašine;
- on ima podršku za Unicode, što omogućuje da se razmene ma kakve informacije na bilo kom pisanom jeziku koji koriste ljudi;
- on može predstavljati najopštije strukture koje se koriste u računarstvu: slogove, liste i drveća;
- sam format je samodokumentujući - on opisuje strukture, imena polja i konkretne vrednosti;
- striktna sintaksa i zahtevi za čitanjem omogućuju da algoritmi za manipulaciju budu jednostavni, efikasni i konzistentni.

Jezik XML se često koristi i za smeštaj i obradu dokumenata, kako on-line, tako i off-line. Za ovakav način korišćenja preporučuju ga sledeće njegove osobine:

- XML je robustan, logički proveriv format koji je baziran na međunarodnim standardima.;
- hijerarhijska struktura podataka je pogodna za najveći broj tipova dokumenata;
- XML je čisti tekst i nije opterećen licencama ili ograničenjima;
- on je nezavisan od platforme, pa je stoga relativno imun na promene tehnologije;
- on i njegovi prethodnik, SGML, se koriste počev od 1986. godine, pa postoji široko iskustvo u radu i mnogo dostupnog softvera.

Ipak, XML je kod nekih primena pokazao određene slabosti:

- Njegova sintaksa je dosta opširna i delimično redundantna. Ovo može negativno uticati na čitljivost konstrukcija i na efikasnost aplikacije, a takođe dovodi do većih troškova smeštaja

informacija. Ovo otežava i primenu XML-a u slučajevima gde je limitiran propusni opseg (mada u tim slučajevima kompresija podataka može da dovede do poboljšanja).

- Programi za čitanje XML-a se prave tako da rekurzivno očitavaju ugnježdene strukture i oni moraju da vrše dodatne provere radi otkrivanja neodgovarajuće formatiranog ili pogrešno uređenog dela podataka – što negativno utiče na performanse. Nadalje, pojavljuju se dodatni sigurnosni problemi kada se čita XML sa lokacije kojoj ne može da se veruje u potpunosti, jer čitanje takvih podataka može dovesti do iscrpljivanja resursa ili prekoračenja steka. Ipak, popularnost XML-a je toliko velika, da su se na tržištu pojavili hardverski uređaji čija je jedina svrha procesiranje XML-a. Kod ovih uređaja je kod za procesiranje implementiran direktno u hardveru, što dovodi do ubrzanja rada celokupnog sistema i održavanja performansi na zadovoljavajućem nivou.
- Pojedini autori su smatrali da XML sintaksa sadrži određene čudne i nepotrebne karakteristike koje potiču od SGML-a. Međutim, pokazalo se da nema konsenzusa oko toga koje su karakteristike XML-a stvarno čudne i nepotrebne.
- Osnovni zahtevi za XML čitače ne podržavaju previše tipova podataka, pa je posle čitanja potrebno uložiti dodatni napor da bi se pročitani podaci doveli u zadovoljavajući oblik. Ova funkcionalnost je dodata jezikom za XML sheme.
- Modeliranje preklapajućih struktura podataka (struktura koje nisu hijerarhijske), kao i uklapanje XML-a u relacionu ili objektno-orjentisanu paradigmu zahteva dodatni napor – mada u .NET-u za to uklapanje postoje funkcije čijim se pozivanjem obavi glavnina posla.

Da bi XML dokumenat bio korektan on mora da bude:

- Dobro formiran. Dobro formiran dokumenat se pokorava pravilima sintakse XML-a. Na primer, ako neprazan elemenat ima otvarajući tag, a nema zatvarajući, on nije dobro formiran.
- Validan. Validan dokumenat ima podatke koji se uklapaju u konkretan skup korisnički definisanih pravila o sadržaju, koja opisuju ispravne vrednosti za podatke i za lokacije. Na primer, ako se za elemenat u dokumentu zahteva da sadrži tekst koji se može interpretirati kao ceo broj, a taj element sadrži reč „Matematika“ ili je prazan, tada taj dokumenat nije validan.

Dobro formirani dokument

XML dokumenat je tekst, tj. sekvenca znakova. XML specifikacija zahteva podršku za Unicode kodiranja UTF-8 i UTF-16 (UTF-32 nije obavezan). Dobro formirani dokumenat mora da zadovoljava sledeće uslove:

- Postoji tačno jedan elemenat-koren u celom dokumentu. Moguće je da se XML deklaracija, instrukcije o procesiranju ili komentari nađu pre tog korenog elementa.
- Neprazni elementi sadrže početni tag i završni tag.
- Prazni elementi mogu biti označeni sa samozatvarajućim tagom. Tako je, na primer, tag `<PrazanTag/>` ekvivalentan sa `<PrazanTag></PrazanTag>`.
- Vrednosti atributa se uokviruju parom apostrofa ili parom navodnika.
- Tagovi se mogu umetati jedan u drugi, ali se ne smeju preklapati. Svaki elemenat koji nije koren mora da se potpuno sadrži unutar roditeljskog elementa.
- Sadržaj dokumenta mora da se pokorava definiciji skupa karaktera (eng. character set - charset). Skup karaktera se obično definiše u XML deklaraciji, ali može biti određen i transportnim protokolom (kao što je HTTP). Ako skup karaktera nije definisan, podrazumeva se da se radi o Unicode UTF-8.
- Kod imena elemenata se razlikuju velika i mala slova.. Tako, na primer, sledeća konstrukcija predstavlja dobro-formiran par:

  ```
  <Crossover> ... </Crossover>
  ```

  a ova konstrukcija nije dobro formirana:

  ```
  <crossover> ... </Crossover>
  ```

Pažljiv izbor imena XML elemenata opisuje značenje označenih podataka. Ovo povećava čitljivost dokumenta, pri čemu biva zadržana strogost koja je potrebna za uspešno softversko čitanje i manipulaciju.
Smislena imena elemenata čoveku sugerišu semantiku elemenata i atributa, bez potrebe da se referiše na spoljašnju dokumentaciju. Sa druge strane, preduga imena elemenata mogu dovesti do preopširnosti, koja komplikuje pisanje i uvećava veličinu fajla.

Valjani dokument

Ako je dobro formiran XML dokumenat u skladu sa konkretnom shemom, tada je taj dokumenat valjan.
Shema je opis tipa XML dokumenta, obično iskazan u obliku ograničenja postavljenih nad strukturom i sadržajem dokumenata te vrste (koja su dodata na ograničenja koja propisuje sam XML).
Pre razvoja opšteg jezika za opis podataka, kao što je SGML ili XML, dizajneri softvera su morali da definišu specijalne formate datoteka ili male jezike kako bi podaci bili preneseni iz programa u program. To je dalje zahtevalo pisanje detaljne specifikacije i specijalizovanih čitača i pisača. Regularna struktura, koja karakteriše XML, i striktna pravila čitanja dopuštaju programerima da posao čitanja ostave standardnim alatima, a kako XML obezbeđuje opšti okvir orjentisan prema podacima to se programer može koncentrisati samo na razvoj pravila za svoje podatke i to na relativno viskom nivou apstrakcije.
U manje-više svakom popularnom razvojnom okruženju su već razvijeni detaljno testirani alati za validiranje XML-a, a često su na raspolaganju i posebni XML editori.

DTD

Najstariji format za definisanje sheme u XML-u je DTD (eng. Document Type Definition), koji je nasleđen iz SGML-a. Kako je DTD uključen u XML 1.0 standard, podrška za njega je neophodna. Međutim, vremenom su uočena ograničenja DTD-a:

- On nema podršku za novije karakteristike XML-a (npr. za prostore imena).
- Nedostaje mu izražajnost – pojedini formalni aspekti XML dokumenta ne mogu biti obuhvaćeni sa DTD-om.
- Za opis sheme DTD koristi novu sintaksu, koja nije bazirana na XML-u

XML Schema

Noviji XML shema jezik, opisan od W3C kao naslednik DTD-a, je XML Shema, ili preciznije XSD (eng. XML Schema Definition). XSD su mnogo moćniji od DTD-a ,- oni koriste bogat sistem tipova podataka i dopuštaju mnogo detaljnije postavljanje uslova vezanih za logičku strukturu XML dokumenta. Stoga, kada treba vršiti robusniju validaciju XML-a, to se postiže korišćenjem XSD-a. Pored toga, XSD-ovi koriste XML sintaksu, što pruža dodatne prednosti.
Sheme retko uključuju sintaksna pravila u skup uslova. One obično ograničavaju imena elemenata i atributa i dopuštenu hijerarhiju pripadanja (u shemi se, na primer, specificira da XML elemenat `Rođendan` sadrži tačno jedan elemenat `Mesec` i jedan elemenat `Dan`). Uslovi koji se sadrže u shemi mogu takođe uključivati i tipove podataka – takvi uslovi utiču na način procesiranja informacija.

Internacionalno korišćenje

XML u potpunosti podržava Unicode karaktere u imenima elemenata, u atributima i podacima. Stoga, dokument koji sledi predstavlja dobro formiran XML dokumenat, čak iako istovremeno uključuje i kineske znake i rusku ćirilicu.

```
<?xml version="1.0" encoding="UTF-8"?>
<俄语>Данные</俄语>
```

Verzije XML-a

Postoje dve aktuelne verzije XML-a. Prva, XML 1.0 je inicijalno definisana početkom 1998. Od tada je ta verzija imala nekoliko manjih revizija koje nisu dobijale posebne verzije brojeva. Poslednja od tih revizija je objavljena u februaru 2004. Taj XML 1.0 je široko prihvaćen i još uvek se on preporučuje za opštu upotrebu.
Verzija XML 1.1 se pojavila istog dana kada se pojavili treća revizija XML 1.0. Ova nova verzija sadrži donekle sporne karakteristike, čija je namera da načine XML lakšim za korišćenje kod pojedinih klasa korisnika. Nadalje, XML 1.1 dopušta korišćenje većeg broja kontrolnih karaktera nego što je to slučaj kod verzije 1.0.
Način korišćenja XML-a iz .NET-a i C# su potpunije opisani u [Baarts00], a iz Jave u [McLaug00].

Primena u EA sistemu

Kao što je već istaknuto, XML nije odgovarajući za svaku situaciju. XML dokumenti imaju tendenciju da budu mnogo opširniji od binarnih formata koje zamenjuju. Stoga, oni zahtevaju veći propusni opseg mreže i zauzimaju veći memorijski prostor, ili se vrši njihova kompresija, što zahteva veće procesorsko vreme.

Nadalje, parsiranje XML-a je obično sporije i memorijski zahtevnije od parsiranja visoko optimizovanih binarnih formata. Pažljiv dizajn aplikacije i korišćenja XML-a u njoj može zaobići neke od ovih problema.
Što se prethodno uočenog problema tiče, treba istaći da način izvršavanja EA ne generiše veliki mrežni saobraćaj - klijent pošalje parametre algoritma (u XML formatu) servisu, servis izvrši algoritam i pošalje rezultate (opet u XML formatu) nazad klijentu. Dakle, u svim netrivijalnim slučajevima, vreme utrošeno na komunikaciju je skoro zanemarljivo mali deo vremena koje servis troši na izvršavanje algoritma.

4.2.5. UML

U skladu sa poznatom istočnjačkom poslovicom „jedna slika govori više nego hiljadu reči" razmatranje obrazaca dizajna i arhitekture EA sistema će često sadržavati dijagrame koji su realizovani u jeziku koji predstavlja standard za modeliranje – unificiranom jeziku modeliranja (eng. Unified Modelling Language - UML).
UML se definiše (videti [Weisf00]) kao „grafički jezik za vizuelizaciju, specifikaciju, konstruisanje i dokumentovanje činjenica o softverskom sistemu". UML pruža standardizovani pristup izradi planova softverskog sistema. U osnovi, UML omogućava grafičko predstavljanje objektno-orjentisanog softverskog sistema i manipulisanje njime.
UML je nastao sredinom 90-tih godina prošlog veka, kada su autori dve tada najpopularnije metodologije objektno-orjentisanog modeliranja Grady Booch (koji je kreirao Booch metodologiju) i James Rumbaugh (kreirao OMT) objedinili napore u cilju kreiranja unificiranog metoda modeliranja. Njima se u tim naporima ubrzo pridružio Ivar Jacobson, autor OOSE metoda – jednog od pedesetak pristupa modeliranju koji su tada postojali na tržištu.
Juna 1995 godine, Grupa za upravljanje objektima (eng. Object Management Group – OMG) je uputila zahtev za zajedničkim pristupom modeliranju softvera.
Pod vođstvom trojke Booch, Rumbaugh i Jacobson (poznatih kao „tri amigosa") tim eksperata koji su predstavljali veći broj kompanija su predali OMG-u svoj predlog za Unifikovani jezik modeliranja UML.
Do jeseni 1997, posle dužih usaglašavanja, javnih rasprava i komentara, OMG je odobrio standard UML-a (tzv. UML 1.1.).
UML je veoma koristan kod većeg broja inženjerskih problema, počev od jednoprocesnih i jednokorisničkih aplikacija, do distribuiranih sistema – što UML čini jako bogatim, ali i veoma obimnim.
Od svog nastanka UML se revidirao i rastao, pa je kreiran veći broj verzija (najznačajnije su bile 1.3 i 1.5), da bi danas aktuelna verzija bila 2.0.
Najveći broj današnjih uspešnih komercijalnih alata za modeliranje podržava najveći deo UML 2.0 specifikacije, a nivo podrške je iz dana u dan sve viši.
Važno je uočiti razliku između UML modela i skupa dijagrama datog sistema. Dijagram je delimična grafička reprezentacija modela sistema (videti [Booch99]). Model, pored dijagrama, sadrži „semantičku pozadinu" – tekstualni dokument sa opisom slučajeva korišćenja koji određuju elemente modela i dijagrama.
Model sistema se sastoji od tri dela:

- Funkcionalni model – prikazuje funkcionalnost sistema sa tačke gledišta korisnika; sadrži dijagrame korišćenja (eng. use case diagram).
- Objektni model – prikazuje strukturu sistema i podsistema korišćenjem objekata, atributa, operacija i asocijacija; sadrži klasne dijagrame.
- Dinamički model – prikazuje unutrašnje ponašanje sistema; tu spadaju sekvencni dijagram, dijagram aktivnosti i dijagram stanja konačnog automata.

Specifikacija UML 2.0 opisuje 13 tipova dijagrama, grupisanih u tri grupe:

- Strukturni dijagrami (klasni dijagram, dijagram komponenti, objektni dijagram, dijagram složene strukture, dijagram isporuke,dijagram paketa).
- Dijagrami ponašanja (dijagram aktivnosti, dijagram korišćenja potpunije opisan u radu [Schne01] i dijagram stanja konačnog automata).
- Dijagrami interakcije (sekvencni dijagram, dijagram saradnje i/ili komunikacije, dijagram pregleda interakcije i dijagram vremena), koji se mogu posmatrati i kao podskup dijagrama ponašanja, su koncentrisani na opis toka kontrole i toka podataka između stvari u sistemu koji se modelira.

U skladu sa inžinjerskom tradicijom, u svakoj vrsti UML dijagrama je dopušteno dodavanje napomena koje se odnose na opis načina korišćenja, uslov ili nameru i priključivanje napomena uz konkretne elemente dijagrama.
Oformljeni modeli sistema mogu biti korišćeni od strane raznih UML alata, ako ti alati podržavaju rad sa datotekama u XMI formatu (koji je zasnovan na XML-u) ili sa datotekama u DI (eng. Diagram Interchange) formatu.
Iako je UML široko prihvaćen i intenzivno korišćen standard, on je i kritikovan zbog neprecizne semantike koja proizvodi subjektivne interpretacije i zbog teškoća koje se javljaju u kasnijim fazama razvoja. UML se smatra i preopširnim, previše usitnjenim u mnogim aspektima – tako da se može dogoditi da se „od drveća ne vidi šuma". Ipak arhitektura upravljana modelom (eng. model-driven architecture) zahteva postojanje iscrpne opšte notacije, kao što je UML 2.0. Jedan od problema koji dovodi do kriticizma i nezadovoljstva sa UML-om je široko prihvatanje UML-a i njegovo korišćenje od strane ljudi koji nemaju adekvatna znanja i veštine(videti [Bell04]).
Tipovi UML dijagrama koji su korišćeni u ovom radu predstavljaju samo mali podskup vrsta UML dijagrama. Potpunije upoznavanje sa UML-om (istorijatom, verzijama, osnovnim pojmovima, svim tipovima dijagrama i elemenata, oblastima i načinima korišćenja, UML alatima itd.) zahteva jako mnogo prostora i uključivanje takvih sadržaja bi, po proceni autora, dovelo do prevelikog proširenja obima rada. Osnovne informacije o UML-u mogu da se nađu u [Weisf00], a potpunije u [Rumba98].

4.2.6. Obrasci dizajna

Obrasci dizajna (eng. design patterns) su rešenja problema u softverskom dizajnu koja se često ponavljaju prilikom razvoja aplikacija u realnom svetu. Obrasci dizajna se odnose na interakcije između objekata, kao i na obezbeđivanje platforma za komunikaciju pri razvoju elegantnih rešenja koja se mogu višestruko koristiti kao odgovor na široko rasprostranjene programerske izazove.
Obrasci dizajna predstavljaju najnaprednija tehnološka rešenja u objektno-orjentisanoj tehnologiji, pa alati za OO analizu, OO knjige i seminari danas nužno uključuju obrasce dizajna.
Pre nešto više od tri decenije, arhitekta Alexander se zapitao: „Da li je kvalitet objektivan?". Preciznije, njega je dominanto interesovala arhitektura, pa je pitanje (u tom domenu) bilo formulisano na sledeći način: „Otkuda se zna koji je arhitektonski dizan dobar? Da li postoji objektivna osnova za takvo prosuđivanje?" Jer, ako nema objektivne osnove za takva razmatranja, dalje istraživanje je uzaludno. S obzirom da je Alexander stekao zaključak da je kvalitet arhitektonskog dizajna objektivan, onda mu se prirodno nametnula sledeća pitanja: „Šta to postoji u kvalitetnom dizajnu, a ne postoji u nekvalitetnom?" i „Šta to postoji u lošem dizajnu, a ne postoji u dobrom?" (videti [Shallo03]). Alexander je rešavao ovaj problem pažljivo i intenzivno proučavajući zgrade, gradove, ulice i sve druge aspekte životnog prostora koji ljudi grade. U tim svojim proučavanjima uočio je da, za konkretnu arhitektonsku kreaciju, dobre konstrukcije imaju mnogo zajedničkih elemenata. Posmatrane arhitektonske strukture, čak iako su iste vrste, se međusobno razlikuju, a istovremeno sve te različite strukture odlikuje dobar dizajn i veliki broj zajedničkih elemenata.
Stoga Alexander uvodi obrazac (eng. pattern, što se ponegde prevodi i kao šablon) koji predstavlja „rešenje problema u kontekstu". Obrazac uključuje četiri elementa:

- naziv obrasca;
- svrha obrasca, tj. zadatak koji obrazac rešava;
- kako će se postići rešenje zadatka;
- ograničenja i uslovi koji se moraju razmotriti da bi se postigao cilj, tj. rešio zadatak.

Alexander je tvrdio da postoje obrasci dizajna koji rešavaju baš svaki arhitektonski problem na koji je on naišao. Nadalje, pri rešavanju složenih arhitektonskih problema moguće je zajedničko korišćenje (tj. kombinacija) obrazaca dizajna.
U prvoj polovini devedesetih godina prošlog veka, grupa iskusnih programera se upoznala sa delima Alexandera i zapitali da li ono šta važi u arhitekturi takođe važi u dizajnu softvera:

- Da li se softverski problemi koji se stalno ponavljaju mogu rešavati slično tome kako se radi u arhitekturi?
- Da li je moguć dizajn softvera u terminima obrazaca, uz kreiranje konkretnog rešenja onda kada se obrasci identifikuju?

Po njima, odgovor na oba pitanja je bio „bezuslovno da“, a sledeći korak koji su preduzeli je identifikovanje obrazaca i razvoj standarda za katalogizaciju novih obrazaca.
Knjiga koja je ostvarila najveći uticaj na dizajnere softvera i koja predstavlja ugaoni kamen za obrasce dizajna je [Gamma95] tj. „Obrasci dizajna: elementi višestruko korišćenog objektno-orjentisanog softvera“ autora Gamma, Helm, Johnson, Vlissides. U znak prepoznavanja i uvažavanja značaj njihovog rada, ova grupa autora se u literaturi često označava terminom „Četvoročlana banda“ (eng. Gang of Four - GoF).
Ova knjiga je:

- definisala poziciju obrazaca dizajna u softverskom inženjerstvu;
- opisala strukturu po kojoj se katalogizuju i opisuju obrasci dizajna;
- katalogizovala 23 takva obrasca;
- oformila objektno-orjentisane strategije i pristupe koji su bazirani na obrascima dizajna.

Važno je istaći da autori nisu kreirali obrasce koji su opisani u ovoj knjizi, već su identifikovali obrasce koji su već postojali u programerskoj zajednici i koji su odslikavali visokokvalitetni dizajn za konkretne probleme (što jako podseća na rad Alexander-a u domenu arhitekture).
Najčešće isticani razlozi za korišćenje obrazaca dizajna su (videti [Fowler02], [Teale03] i [Shallo03]):

- Ponovno korišćenje rešenja: ponovnim korišćenjem utvrđenog dizajna, autori softvera dobijaju brži i čistiji početak i izbegavaju korišćenje „prljavih“ trikova. Na taj način se tokom softverskog dizajna preuzima iskustvo drugih i ne pojavljuje se potreba za ponovnim otkrivanjem rešenja za česte ponavljajuće probleme.
- Uspostavljanje zajedničke terminologije: komunikacija i timski rad zahteva zajednički osnovni rečnik i zajednički pogled na problem. Obrasci dizajna obezbeđuju takvu referentnu tačku tokom faze analize i faze dizajniranja projekta.
- Obrasci dizajna daju autorima mogućnost proučavanja problema i objektno-orjentisanog pristupa iz više perspektive, oslobađajući autora od potrebe da suviše rano mora da vodi računa o detaljima.
- Obrasci dizajna obezbeđuju i mogućnost lakše modifikacije softvera. Razlog za to je što se radi o rešenjima koja su testirana u dužem vremenu, pa su strukturisana tako da lakše mogu istrpeti promenu nego ona rešenja koja autoru prvo padnu na pamet.

U knjizi [Gamma95] „četvoročlana banda“ sugeriše strategije za kreiranje dobrog objektno-orjentisanog koda:

- odrediti koji elementi se variraju i te elemente enkapsulirati objektima;
- favorizovati kompoziciju (tj. pripadanje) u odnosu na nasleđivanje.

Ove strategije se pojavljuju u skoro svim obrascima koji su opisani u literaturi. To dovodi do još jedne prednosti kod obrazaca dizajna: dopušta se kreiranje dizajna za kompleksne probleme a da ne bude formirana velika hijerarhija nasleđivanja.
Kako su obrasci dizajna korišćeni prilikom razvoja EA softverskog sistema, to će u ovom radu biti ukratko opisani oni obrasci koji su tu korišćeni (bilo da se radi o obrascima koji su uvršteni u katalog obrazaca u [Gamma95], bilo o onim opisanim u [Fowler02] i [Teale03]). Isti pristup koji je realizovan kod obrazaca dizajna će biti iskorišćen i za analizu i dizajn arhitektonskih aspekata sistema.
Opis svakog od obrazaca sadrži sledeće elemente:

- Kontekst: kontekst u kome se problem pojavljuje.
- Problem: problem koji obrazac pokušava da reši.
- Posledice: konsekvence korišćenja ovog konkretnog obrasca.
- Učesnici: objekti koji učestvuju u funkcionisanju obrasca.
- Rešenje: rešenje koje je realizovano.
- Implementacija: kako se obrazac može implementirati. Implementacije su konkretne manifestacije posledice primene obrasca.
- Prednosti i nedostatci: prednosti i nedostatci realizovanog rešenja.

Obrazac Fasada

Kontekst
Fasada (eng. facade) se obično javlja kod kompleksnih sistema kod kojih treba drugoj strani iskazati samo deo postojeće funkcionalnosti. Fasada je, na primer, potrebna tamo gde mnogo ljudi u timu treba da se obuči za novi sistem, iako svako od njih koristi samo mali deo funkcionalnosti. Dakle, fasada se koristi u situacijama kada je potrebno uprostiti način korišćenja postojećeg sistema i definisati sopstveni interfejs.

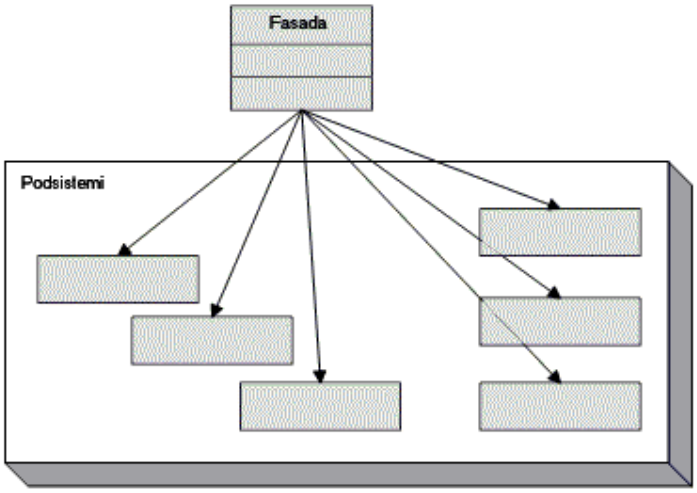
Problem
Potrebno je koristiti samo podskup kompleksnog sistema, ili treba kreirati interakciju sa sistemom na jedinstven, poseban način. Nadalje, ponekad je potrebno izvršiti enkapsulaciju sistema. Neki od razloga za enkapsulaciju su:

- Praćenje rada sistema – primoravanje da svo korišćenje sistema ide preko fasade, olakšava implementiranje mehanizama za nadgledanje i praćenje rada sistema.
- Promena sistema – ako se očekuje da će u budućnosti sistem biti promenjen, blagovremenim korišćenjem fasade se može postići da se ta promena izvede uz minimalnu utrošak truda i vremena.

Posledice
Obrazac fasada uprošćava korišćenje sistema. Međutim, s obzirom da fasada nije kompletna, određene funkcionalnosti sistema neće biti na raspolaganju klijentima.
Fasada se takođe može koristiti za sakrivanje, odnosno enkapsulaciju, sistema. Ona može sadržavati sistem koji se enkapsulira kao privatni član unutar fasada klase, koji nije dostupan korisnicima, tj. klijentima.

Učesnici
Klase i/ili objekti koji učestvuju kod ovog obrasca su `Fasada` i podsistemi. `Fasada` ima informaciju koje su klase u podsistemima odgovorne za realizaciju zahteva i ona usmerava klijentske zahteve prema odgovarajućim posistemima. Podsistemi implementiraju potrebnu funkcionalnost i odrađuju operacije koje im je prosledila fasada. Podsistemi nemaju znanja o fasadi niti sadrže reference na nju.

Rešenje
Ovaj obrazac obezbeđuje jedan interfejs za pristup skupu interfejsa sistema. Na taj način, fasada definiše interfejs višeg nivoa, koji olakšava korišćenje podsistema.



slika 4.1. Fasada

Implementacija
Implementacija je dosta direktna: kreira se fasadna klasa, koja sadrži javne metode koje koriste klijenti, a koji predstavljaju podskup celokupne funkcionalnost svih podsistema u sistemu kome se postavlja fasada.

Prednosti i nedostatci
Prednost korišćenja ovog obrasca dizajna ogleda se u činjenici da klijenti složenog sistema imaju jednu tačku pristupa, preko koje mogu na jednostavan način realizovati zahtevane operacije. S druge strane, ako je oblik i

način interakcije sa elementima sistema podložan čestim promenama, korišćenje ovog obrasca postaje veoma komplikovano.

Obrazac Filter za presretanje

Kontekst
Kao što je poznato (videti [Trowbr03]) web aplikacije su složenije od klasičnih klijent-server aplikacija i njihova izgradnja (naročito ako se zahteva skalabilnost rešenja) od nule bi zahtevala mnogo više truda nego što je to slučaj kod drugih vrsta aplikacija - treba prevazići probleme koje sa sobom nose protokoli HTTP, URL, MIME, eventualno SSL, potreba za autentifikacijom korisnika, za detekcijom Internet pregledača (eng. Internet browser), praćenjem rada korisnika itd. Tu se često koristi obrazac filter za presretanje (eng. intercepting filter).

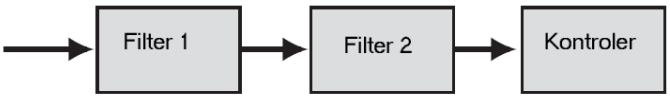
Problem
Kako realizovati zajedničko preprocesiranje i postprocesiranje vezano za web zahteve.

Posledice
Postoji veći broj načina za pristup ovom problemu, pa će stoga ukratko biti razmotreni elementi najboljeg pristupa i njihove posledice (prednosti i mane):

- Uobičajena je praksa da se funkcije niskog nivoa, kao što su obrada HTTP zaglavlja, rad sa kolačićima (eng. cookies) itd. odvoje od aplikativne logike. Ovo omogućuje da se aplikativna logika testira i ponovo koristi i u okruženjima koje ne koriste web klijente.
- Osobine preprocesiranja i postprocesiranja web zahteva mogu da se menjaju nezavisno od funkcionalnosti aplikacije, pa je dobro imati odvojene poglede na ova dva različita aspekta rada softverskog sistema. Ta odvojenost pogleda ograničava i situacije gde promena jednog dela aplikacije zahteva promene u mnogim drugim delovima.
- Mnoge operacije preprocesiranja i postprocesiranja su zajedničke pri obradi svih web zahteva, pa je poželjno da se te operacije implementiraju na jednoj, centralnoj lokaciji, kako bi se izbeglo dupliranje koda.
- Mnoge funkcije niskog nivoa ne zavise jedna od druge. na primer, detekcija pregledača i enkodiranje karaktera su dve nezavisne funkcije. Da bi se olakšalo ponovno korišćenje, te funkcije treba enkapsulirati u skup modula koji se mogu komponovati – dodavati i uklanjati po potrebi tako da ne utiču na ostale delove koda.
- Često je od velike koristi da se omogući dodavanje ili uklanjanje modula pri izvršavanju, a ne pri kompilaciji. Ta sposobnost aplikacije da se dodaju ili oduzmu moduli a da se ne menja kod se naziva mogućnost komponovanja pri izvršavanju.
- Budući da se funkcije niskog nivoa izvršavaju pri obradi svakog web zahteva, veoma su važne performanse – pa ove komponente, tj. zajedničke funkcije, ne smeju sadržavati karakteristike i tačke odluke koje nisu neophodne. Drugim rečima, svaki korak procesiranja treba da bude što kompaktniji i što efikasniji.
- Odvajanje preprocesiranja i postprocesiranja od aplikativne logike dopušta da ta funkcionalnost bude realizovana u drugom programskom jeziku – jeziku različitom od jezika u kom je programirana aplikativna logika.

Učesnici
Klase i/ili objekti koji učestvuju kod ovog obrasca su konkretni filteri i kontroler. Svaki od konkretnih filtera realizuje jedan od zadataka preprocesiranja. `Kontroler` sadrži realizaciju aplikativne logike.

Rešenje
Kreirati lanac filtera koji se mogu komponovati, a koji implementiraju zajedničko preprocesiranje i postprocesiranje web zahteva.

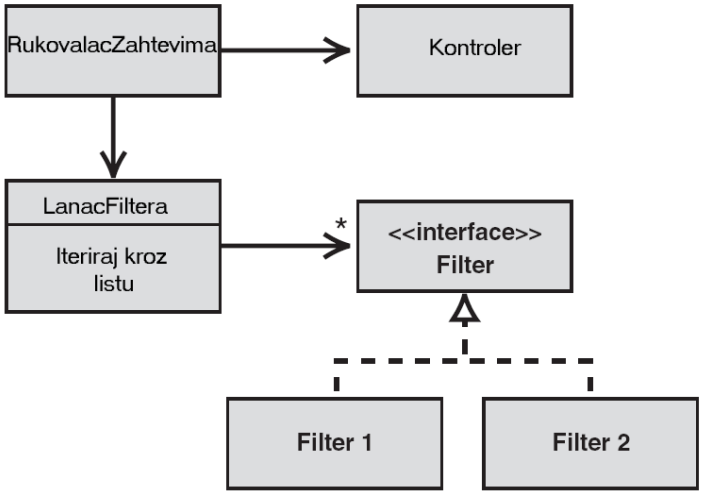


slika 4.2. Lanac filtera

Filteri su formirani od niza nezavisnih modula koji se mogu uvezati kako bi izvršili skup zajedničkih koraka procesiranja pre nego što kontrola bude prepuštena kontroleru. Kako pojedinačni filteri implementiraju isti interfejs, oni ne zavise eksplicitno jedan od drugog, pa se novi filteri mogu dodati bez uticanja na postojeće. Čak se ti novi filteri mogu dodati dinamički pri izvršavanju – konsultujući sadržaj konfiguracione datoteke. Pojedinačni filteri se trebaju dizajnirati na takav način da ne sadrže nikakve pretpostavke o prisustvu drugih filtera. Na taj način se omogućuje lako dodavanje, uklanjanje ili rearanžiranje filtera.

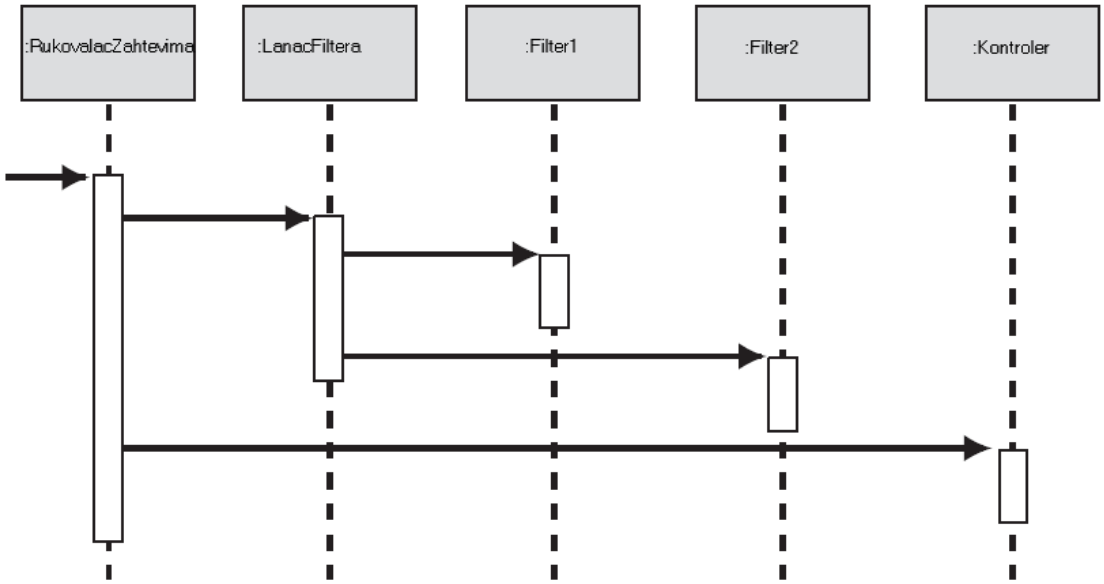
Implementacija

Direktna implementacija filtera za presretanje je lanac filtera koji iterira kroz listu svih filtera (kao na slici 4.3).



slika 4.3. Implementacija lanca filtera

Po prijemu zahteva (obično se radi o web zahtevima), `RukovalacZahtevima` prvo prosledi kontrolu do objekta koji predstavlja lanac filtera. `LanacFiltera` čita sekvencu filtera iz konfiguracione datoteke kako bi se postigla mogućnost komponovanja pri izvršavanju. Po izvršenju svih filtera, rukovalac zahtevima prenosi kontrolu na `Kontroler`, koji izvršava funkcije aplikativne logike.



slika 4.4. Sekvencni dijagram za lanac filtera

Jedna od ključnih prednosti ovakvog dizajna je da su filteri samostalne komponente koje nemaju direktnu zavisnost od drugih filtera ili od kontrolera, zato što lanac filtera izvršava svaki od filtera. Stoga, filteri ne sadrže reference na sledeći filter.

Kako se filteri za presretanje izvršavaju neposredno pre i neposredno posle kontrolera, ponekad nije lako odrediti da li da se funkcionalnost implementira unutar filtera ili unutar kontrolera. Sledeće teze predstavljaju uputstva koja se trebaju uzeti u obzir pri rešavanju ove dileme:

- Filter više odgovaraju za funkcije nižeg nivoa koje su povezane sa prenosom – te funkcije su obično dobro enkapsulirane, efikasne i bez stanja, pa nema potrebe da jedna drugoj prenose dodatne informacije - lako ih je ulančati.
- Prava funkcionalnost aplikacije treba da bude realizovana unutar kontrolera (ili pomoćnika za kontrolera), a ne unutar filtera, jer ta funkcionalnost obično ne može da se komponuje na način kako to zahtevaju filteri.
- U najvećem broju slučajeva, procesiranje unutar filtera ne zavisi od stanja aplikacije (ili sesije) već se odigrava bez obzira na sve.
- Zbog čestog izvršavanja (pri obradi svakog web zahteva), od suštinske je važnosti efikasnost izvršavanja, tj. performanse. :NET okvir, kao što je ranije istaknuto, dopušta da se u okviru istog softverskog sistema nalaze delovi (tj. komponente) napisane u različitim programskim jezicima – pa se može omogućiti da filteri budu realizovani u efikasnijem jeziku u kome je teže programirati (kao što je C++ koji može kreirati i pravi izvršni kod), a da komponente za realizaciju aplikativne logike budu realizovane u izražajnijim, a manje efikasnim jezicima (kao što je C#, VB, J#) gde se prevođenje vrši do nivoa upravljanog koda – što je manje efikasno, ali programer ima pun pristup svim uslugama koje daje .NET okvir.

Jedan od obrazaca dizajna koji se takođe često koristi u slučajevima u kojima se koristi i filter za presretanje je dekorator (videti [Gamma95]), ali s obzirom da je u EA sistemu koji se opisuje korišćen filter za presretanje, to se u ovom radu dekorator neće opisivati.

Prednosti i nedostatci

Kao i svi drugi obrasci dizajna, i filter za presretanje ima svoje prednosti i mane. Prvo o prednostima:

- Odvojenost pogleda. Logika koja se nalazi u filterima je potpuno razdvojena od aplikativne logike, pa kod za realizaciju aplikativne logike nije pogođen pri promeni karakteristika niskog nivoa (kao što je promena transportnog protokola ili promena mehanizma za čuvanje stanja aplikacije i sesije između korisnika).

- Fleksibilnost. Filteri su međusobno nezavisni, pa se (bez ikakve izmene u kodu filtera) mogu ulančati u ma kakvu zahtevanu kombinaciju.
- Centralizovana konfiguracija. S obzirom na mogućnost komponovanja filtera, može se koristiti konfiguraciona datoteka za učitavanje lanca filtera. Tada se za određivanje koji će filteri biti uključeni u procesiranje treba samo napraviti promena u toj konfiguracionoj datoteci, bez potrebe da se bilo šta menja u kodu i da se aplikacija ponovo prevodi.
- Mogućnost komponovanja pri izvršavanju. Filteri za presretanje se mogu kreirati tokom izvršavanja, shodno informacijama zapisanim u konfiguracionoj datoteci – pa se sekvenca izvršavanja filtera može menjati tokom rada aplikacije, a da se ne menja kod.
- Ponovno korišćenje. S obzirom na samodovoljnost i nezavisnost filtera, oni lako mogu biti ponovo korišćeni u drugim aplikacijama.

Naravno, ovaj pristup ima i određene mane:

- Zavisnost od redosleda. Kod filtera za presretanje nema eksplicitne zavisnosti jednog filtera od drugog, pa se može javiti problem ako filteri očekuju da je izvesno procesiranje izvršeno pre nego što su pozvani. U takvim slučajevima je ipak bolje u kodu fiksirati redosled operacija, nego koristiti konfigurabilni lanac filtera.
- Deljeno stanje. Filteri nemaju eksplicitni mehanizam za delenje informacija o stanju, osim da manipulišu sadržajem koji procesiraju – a to takođe važi i za prosleđivanje informacija iz filtera do kontrolera. Za informacije o stanju koje se između filtera i do kontrolera prenose pomoću manipulacije obrađivanim sadržajem novi problem predstavlja nemogućnost provere podataka u vremenu prevođenja i nemogućnost provere tipova.

Obrazac Posmatrač

Kontekst

U objektno orjentisanom programiranju, objekti sadrže podatke i ponašanja. Jedna od prednosti korišćenja objektnog pristupa je to što sva manipulacija nad podacima može biti enkapsulirana unutar objekta, čime objekat postaje samodovoljan, a time se povećava mogućnost njegovog ponovnog korišćenja.

Međutim, objekti ne mogu uvek da rade u takvoj izolaciji – oni moraju da sarađuju kako bi realizovali složenije zadatke. Kada objekti sarađuju, oni moraju da obaveštavaju jedan drugog o promeni sopstvenog stanja. Jedan od najčešće korišćenih načina za međusobno informisanje objekata je korišćenje obrasca posmatrač (eng. observer).

Problem

Kako objekat da obavesti druge objekta o promeni stanja, a da ne bude zavisan od klasa drugih objekata?

Posledice

Rešavanje ovog problema zahteva da se razmotre sledeće alternative, stavovi i posledice:

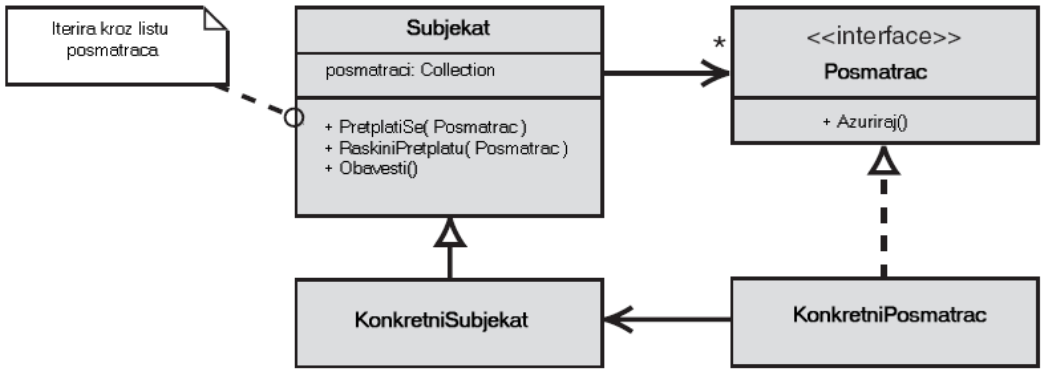
- Najlakši način da se zavisni objekti informišu o promeni stanja je direktan poziv metode. Međutim, takva direktna saradnja objekata kreira zavisnost između odgovarajućih klasa. Takvo direktno ili čvrsto povezivanje (eng. direct coupling, tight coupling) u velikoj meri smanjuje mogućnost ponovnog korišćenja tako napravljenih klasa.
- Potreba da klase budu razdvojene se često javlja u radnim okruženjima koja su upravljana događajima (eng. event-driven frameworks). Tamo samo okruženje mora biti sposobno da informiše aplikaciju o događaju, ali ne sme da zavisi od konkretnih klasa u aplikaciji.
- Aplikacije koje sadrže mnogo čvrsto povezanih klasa su obično teške za održavanje, jer promena u jednoj klasi može uticati na sve klase koje su čvrsto povezane sa njom.
- Ako se direktno pozivaju zavisni metodi, svaki put kad a se doda nova zavisnost, mora da se modifikuje ne samo način na koji zavistan objekat reaguje na promenu stanja, već i sadržaj klase izvornog objekta – objekta koji je promenio stanje.
- U nekim situacijama, broj zavisnih objekata ne može biti pozvan u vremenu dizajniranja aplikacije.
- Direktan poziv funkcije je najefikasniji način za prenos informacija između dva objekta (jedino što je brže je smeštanje funkcionalnosti oba objekta unutar jednog objekta). Stoga će razdvajanje objekata negativno uticati na performanse cele aplikacije, pa se treba pažljivo izvagati odluka.

Učesnici

Klase i/ili objekti koji učestvuju kod ovog obrasca su `Subjekat` (on zna sve svoje posmatrače i obezbeđuje interfejs za pretplaćivanje i raskidanje pretplate svojih posmatrača), `KonkretanSubjekat` (čuva stanja koja interesuju konkretnog posmatrača i šalje obaveštenja kada se stanje menja), `Posmatrac` (definiše interfejs za ažuriranje kod objekata koji treba da budu obavešteni o promeni stanja subjekta) i `KonkretniPosmatrac` (čuva referencu na `KonkretniSubjekat`, čuva stanje subjekta kako bi očuvao konzistentnost i implemetira interfejs za ažuriranje Posmatrac).

Rešenje

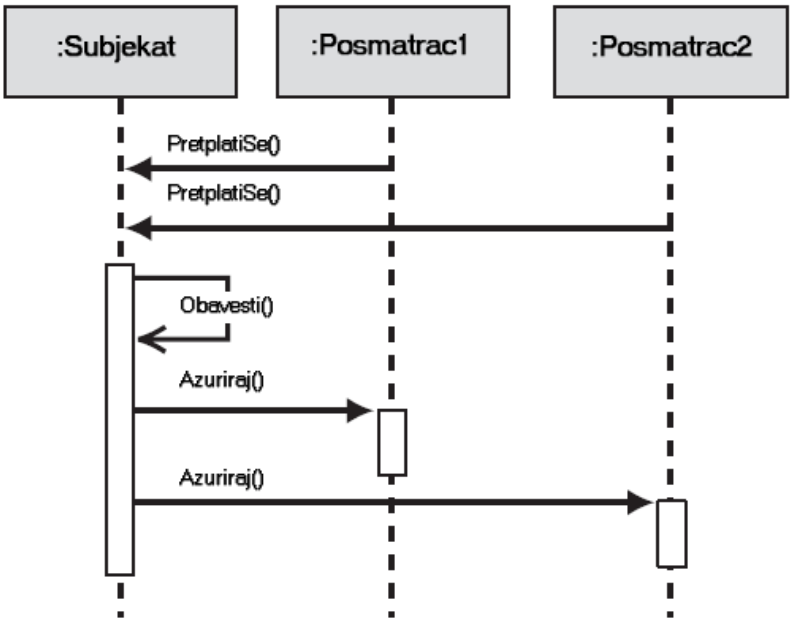
Koristiti obrazac posmatrač da bi se održavala lista zainteresovanih zavisnih objekata (posmatrača) u odvojenom objektu. Postaviti da svi posmatrači implementiraju zajednički interfejs `Posmatrac`, koji omogućuje eliminaciju direktnih zavisnosti između posmatranog objekta i posmatrača.



slika 4.5. Posmatrač – osnovna struktura

Kada dođe do promene stanja koja je značaja za sve zavisne objekte, konkretni subjekat poziva metod `Obavesti()`. Roditeljska klasa `Subjekat` čuva listu referenci na sve zainteresovane posmatrače, pa metod `Obavesti()` može u petlji da prođe kroz listu svih posmatrača i kod svakog registrovanog posmatrača pozove metod `Azuriraj()`. Posmatrači se registruju i otkazuju registraciju pozivom subjektovih metoda `PretplatiSe()` i `RaskiniPretplatu()`.

Kod ovakvog tipa komunikacije između subjekta i posmatrača, saradnja može da bude dinamička, a ne statička. Kako je logika obaveštavanja odvojena od logike sinhronizacije, to se novi posmatrači mogu dodati bez menjanja logike obaveštavanja, a logika obaveštavanja se može menjati bez uticanja na logiku sinhronizacije implementiranu u posmatračima. Kod je zato malo više razdvojen, pa je lakše njegovo održavanje i ponovno korišćenje.



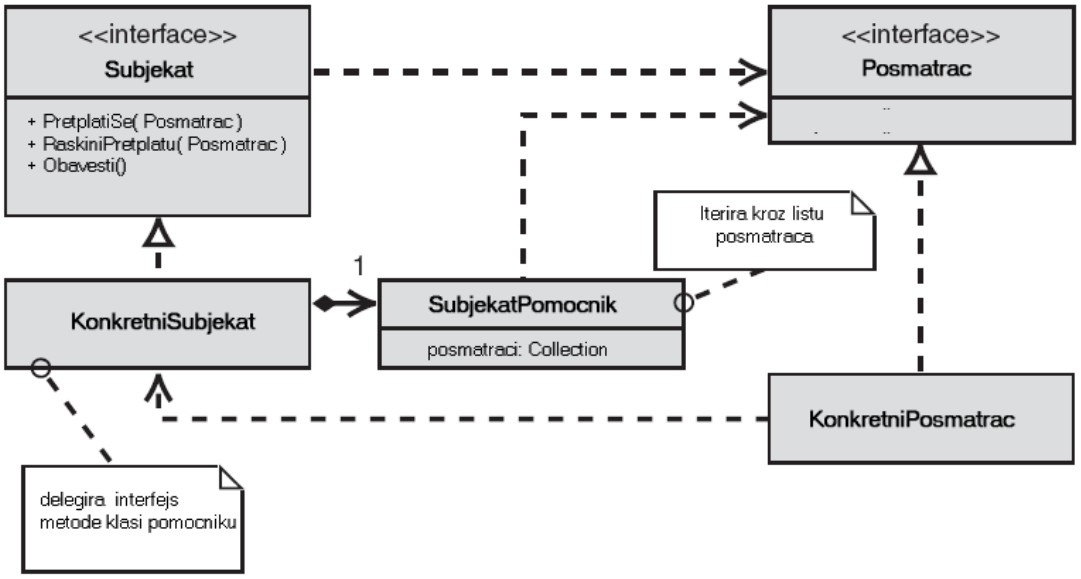
slika 4.6. Osnovna interakcija kod posmatrača

Obaveštavanje objekata o promenama bez uvođenja zavisnosti između klasa je tako čest zahtev da neke platforme obezbeđuju osobine jezika koje izvršavaju baš tu funkciju. Tako, na primer, Microsoft .NET okvir definiše delegate i događaje, koji su u stanju da izvrše zadatke posmatrača.

Implementacija

Klasni dijagram na slici 4.5 ukazuje da je klasa `KonkretniSubjekat` izvedena iz klase `Subjekat` (klase koja sadrži implementaciju metoda za dodavanje i uklanjanje posmatrača i za prolazak kroz listu posmatrača). Stoga, sve što `KonkretniSubjekat` treba da uradi je da bude izveden iz klase `Subjekat` i da pozove `Obavesti()` kada se njegovo stanje promeni. Međutim, ovakav dizajn, kod objektno orjentisanih jezika koji podržavaju jednostruko nasleđivanje (takvi su, na primer, Java i C#) dovodi do problema onemogućavanja nasleđivanja: izvođenje klase `KonkretniSubjekat` iz klase `Subjekat` onemogućuje da klasa `KonkretniSubjekat` bude izvedena iz ma koje druge klase, dok su, s druge strane, objekti klase `KonkretniSubjekat` domenski objekti i za njih je prirodno da budu izvedeni iz neke od klasa koja pripada domenu problema.

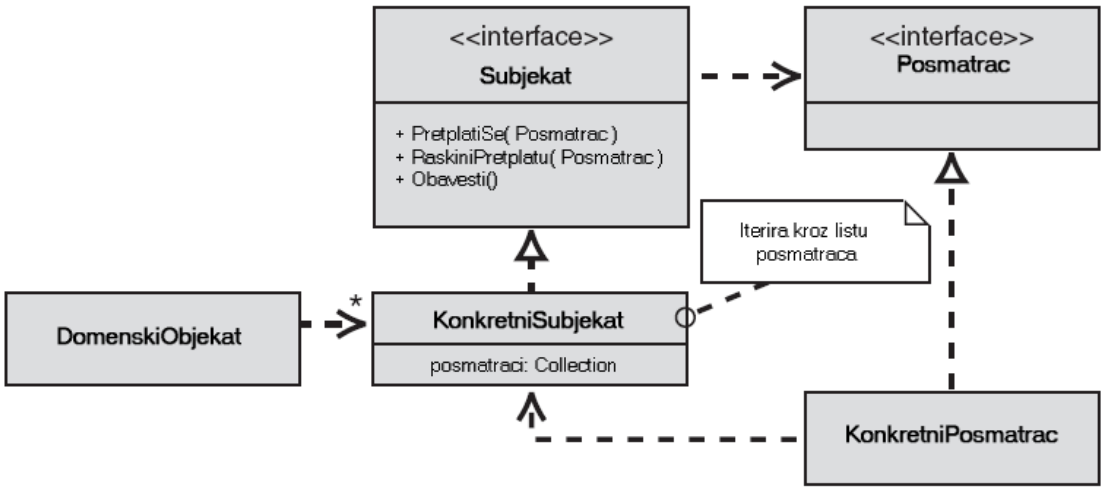
Stoga je u takvim jezicima primerenije rešenje opisano dijagramom na slici 4.7. Tu klasa `KonkretniSubjekat` može biti korišćena u hijerarhiji nasleđivanja, a ne mora biti izvedena iz klase `Subjekat`.



slika 4.7. Korišćenje pomoćne klase kod posmatrača da bi se izbeglo izvođenje iz Subjekta

Problem sa ovim pristupom je u tome što se mora dodati kod za svaku klasu koja implementira interfejs `Subjekat` – i to se mora ponavljati mnogo puta. Nadalje, kako `KonkretniSubjekat` nasleđuje domensku objekat, on ne može praviti razliku među tipovima promene stanja, koja mogu biti pridružena različitim subjektima. Dakle, u ovom scenariju posmatrači se preplaćuju na sve događaje konkretnog subjekta, čak i u slučaju kada bi trebalo da se vrši selekcija događaja.

Ovaj problem se može prevazići kompletnim odvajanjem subjekta od izvorne klase, kao što je prikazano dijagramom na slici 4.8.



slika 4.8. Razdvajanje subjekta i domenskog objekta

Na ovaj način, `KonkretniSubjekat` je sveden na implementaciju interfejsa `Subjekat` i nema drugih odgovornosti. Ovo dopušta da domenskom objektu bude pridruženo više od jednog konkretnog subjekta, pa se može napraviti razlika između različitih tipova unutar jednog domenskog objekta.

Događaji i delegati u :NET okviru su implementirani baš na pristupu koji je opisan prethodnim dijagramom (slika 4.8.) – i to kao konstrukti jezika, tako da autor ne mora ni dizajnirati klasu `KonkretniSubjekat`. U osnovi .NET događaji zamenjuju klasu `KonkretniSubjekat`, a delegati implementiraju zadatke interfejsa `Posmatrac`, pa se obrazac može realizovati uz mnogo manje kodiranja. U .NET implementaciji, događaj predstavlja apstrakciju (podržanu od .NET zajedničkog okruženja za izvršavanje i od kompajlera za razne

jezike) klase `SubjekatPomocnik`. Uloga posmatrača je tu nešto komplikovanija: umesto da implementira interfejs `Posmatrac` i da se pretplati kod subjekta, posmatrač mora kreirati konkretnu instancu delegata i registrovati tu instancu delegata uz subjektov događaj. Za registraciju se mora koristiti instanca delegata čiji je tip određen deklaracijom događaja kod subjekta – u suprotnom registracija neće proći. Da bi kreirao odgovarajuću instancu delegata, posmatrač obezbeđuje ime metoda (može biti i statički, ali ne mora) koji će biti obaveštavan od strane subjekta. Pošto se delegat priveže uz metod, on se može pretplatiti na događaj. Naravno, ako za to postoji potreba, delegat može i raskinuti pretplatu od događaja. Ovakva realizacija ima svojih prednosti (uklanjanje zavisnosti, lakoća proširenja tj. dodavanja novih tipova posmatrača, olakšano testiranje) i mana (porast broja delegata i događaja otežava debagiranje, smanjena efikasnost izvršavanja), pa pri donošenju odluke treba biti dobro upoznat i sa jednim i sa drugim aspektima modela koji se razmatra.
Pri implementaciji obrasca posmatrač, potrebno je povesti računa i još o jednom elementu – kako posmatrač može znati u kojem se stanju nalazi klijentski objekat. Tu se koriste dva pristupa, od kojih svaki ima svoje prednosti i mane:

- model guranja (eng. push model) – klijenti šalju sve relevantne informacije o promeni svog stanja subjektu, koji te informacije prosleđuje posmatraču. Ako su te informacije prosleđene u neutralnom formatu (npr. kao XML) ovaj model ne dopušta zavisnim posmatračima da pristupe klijentu. S druge strane, u ovom modelu subjekat mora da odluči koje od informacija (koje je poslao klijent) su relevantne za posmatrače. Preciznije, ako se doda novi posmatrač, možda će trebati da subjekat publikuje dodatne informacije koje zahteva taj novi posmatrač, a koje ranije nisu bile zahtevane – ovo dovodi do još jedne zavisnosti klijenta i subjekta od posmatrača. Zato se često u ovom modelu, pri svakom pozivu posmatrača, uključuje i referenca na subjekat, pa posmatrači mogu iskoristiti tu referencu da bi dobili informacije o stanju.
- model povlačenja (eng. pull model) – klijenti obaveštavaju subjekat o promeni stanja. Pošto posmatrači prime obaveštenje, oni pristupaju subjektu ili klijentu, kako bi dobili dodatne podatke. Ovaj model ne zahteva da subjekt prosleđuje bilo kakve informacije pri pozivu metoda `Azuriraj()`. Model povlačenja je obično malo manje efikasan od modela guranja.

Rezultati
Iako obrazac posmatrač razdvaja objekte i smanjuje zavisnosti, ne preporučuje se da se on primenjuje na svakom paru objekata među kojima postoje zavisnosti. Pre odluke da li i gde primeniti ovaj obrazac, potrebno je pažljivo proučiti prednosti i mane ovog obrasca dizajna. Prvo o prednostima:

- Slaba povezanost i smanjena zavisnost. Klijent više ne zavisi od posmatrača, jer je izolovan od njih.
- Promenljivi broj posmatrača. Posmatrači se mogu pretplaćivati i mogu raskidati pretplatu tokom izvršenja programa, jer subjekat ne pravi nikakve pretpostavke o broju posmatrača.

Naravno, ovaj obrazac ima i određene nedostatke:

- Oslabljene performanse. To što su zavisnosti među objektima apstrahovane ne znači da dodavanje posmatrača nema uticaj na vreme izvršenja aplikacije.
- Curenje memorije. Mehanizam povratnog poziva (eng. callback) koji se koristi kod posmatrača, a kojim se postiže da se objekat registruje za kasniji poziv, može dovesti do grešaka koje izazivaju curenje memorije, čak i kod jezika koji imaju ugrađen sakupljač otpadaka (eng. garbage collector) kao što su Java ili C#. Naime, ako posmatrač nestane iz opsega, ali se propusti da raskine pretplatu na događaje subjekta, tada subjekat još drži referencu na posmatrača, pa posmatrač ne bude recikliran pri sakupljanju otpadaka, već nastavi da zauzima prostor sve dok postoji subjekat. Ovim nastaju ozbiljna curenja memorije u slučajevima kada je životni vek posmatrača mnogo kraći od životnog veka subjekta – a obično je tako.
- Skrivene zavisnosti. Korišćenje ovog obrasca dizajna pretvara eksplicitne zavisnosti (iskazane pozivanjem metoda) u skrivene zavisnosti (preko posmatrača). Ako se posmatrači koriste izuzetno često u izvornom kodu, programeru postaje gotovo nemoguće da gledanjem izvornog koda otprati šta se dešava tokom izvršenja programa. Ovo dalje dovodi do otežanog shvatanja koje su sve implikacije promene koda, tj. jako otežava održavanje i nadogradnju aplikacije. Ovaj problem eksponencijalno raste sa nivoima širenja (novi nivo širenja je kada se neki subjekat istovremeno ponaša i kao posmatrač drugih subjekata). Stoga se primena obrasca posmatrač treba suziti samo na

dobro definisane interakcije. Svaka primena obrasca posmatrač između domenskih objekata predstavlja dodatni razlog za ponovni dizajn softverskog sistema.

- Teškoće pri testiranju i debagiranju. Što se više razdvoje objekti (što je sa arhitektonskog stanovišta dobro) time teže postaje shvatanje zavisnosti među tim objektima jednostavnim gledanjem izvornog koda ili dijagrama klasa. Samim tim testiranje i debagiranje se otežava.

Obrazac Singlton

Kontekst
U pojedinim situacijama, određeni tip podataka treba da bude na raspolaganju svim ostalim objektima aplikacije. U najvećem broju slučajeva, taj tip podataka je takođe jedinstven za sistem. Na primer, korisnički interfejs može imati samo jedan pokazivač miša kome pristupaju sve aplikacije. Slično tome, obimno i složeno (eng. enterprise) rešenje može da ima interakciju sa starim sistemom samo preko jedinstvenog objekta – fasade.

Problem
Kako obezbediti da instanca objekta bude globalno dostupna i garantovati ta će se kreirati samo jedna instanca te klase?

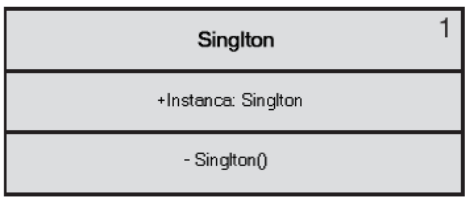
Posledice
Mnogi programski jezici (npr. C++, Microsoft Visual Basic 6) omogućuju kreiranje objekata sa globalnim opsegom vidljivosti (eng. scope). Ovakvi objekti se nalaze u korenu prostora imena i dostupni su svim objektima u aplikaciji. Ovaj pristup (kod tih jezika) obezbeđuje jednostavno rešenje globalnog pristupa, ali ne ispunjava zahtev jednostrukog instanciranja – ne sprečava druge objekte da kreiraju nove instance globalnog objekta. Nadalje, jezik C# (koji nas u ovoj diskusiji najviše interesuje) ne podržava direktno globalne promenljive.
Da bi se obezbedilo da može postojati samo jedna instanca klase, mora se kontrolisati proces instanciranja. dakle, treba sprečiti ostale objekte da kreiraju instancu klase korišćenjem mehanizma instanciranja koji je karakterističan za dati programski jezik (u C# je to korišćenje operatora `new`). Pored dostignuća tog zahteva, obrazac treba da obezbedi centralni mehanizam preko koga svi objekti mogu da dobiju referencu na tu jednu instancu.

Učesnici
Klasa kod ovog obrasca je Singlton klasa, koja je dizajnirana tako da omogućuje klijentima pristup samo jednoj svojoj instanci. Klasa je odgovorna za kreiranje i održavanje svoje jedine instance.

Rešenje
Obrazac Singlton obezbeđuje globalnu jednostruku instancu tako što:

- Čini da klasa kreira jednu instancu same sebe.
- Dopušta da drugi objekti pristupaju toj instanci kroz statički (ponegde se zove i klasni) metod koji vraća referencu na instancu. taj statički metod treba da bude globalno dostupan.
- Deklariše konstruktore klase kao privatne , tako da nijedan drugi objekat ne može da kreira novu instancu.

Implementacija
Slika 4.9 pokazuje statičku strukturu ovog obrasca. UML dijagram klasa je veoma jednostavan, zato što se singlton sastoji od proste klase koja sadrži referencu na jednu svoju instancu.



slika 4.9. Singlton

Prethodni dijagram pokazuje da klasa `Singlton` sadrži javnu statičku osobinu, koja vraća referencu na jednu instancu `Singlton` klase. Nadalje, broj `1` u gornjem desnom uglu ukazuje da tokom izvršavanja može biti samo jedna instanca te klase u sistemu. Kako je podrazumevani konstruktor privatan, svaki drugi objekat u sistemu mora da pristupi singltonu kroz osobinu Instanca.

Veoma često se Singlton klasifikuje kao idiom a ne kao obrazac, zato što rešenje zavisi od karakteristika programskog jezika koji se koristi.

S obzirom da se u razvoju EA softverskog sistema koristi .NET i C#, potrebno je u takvoj situaciji implementirati Singlton na način koji eksploatiše prednosti koje pruža izabrana platforma. Pored toga, (budući da je EA softverski sistem dizajniran kao višenitna aplikacija) od izuzetnog značaja je korišćenje implementacije singltona koje će dobro funkcionisati u takvoj situaciji - koja je sigurna za niti (eng. thread safe).

Iako je Singlton sasvim jednostavan obrazac, postoje različiti izbori i opcije u implementaciji.

Prva među implementacijama koja se razmatra je manje-više rešenje opisano u [Gamma95], s tim što je iskorišćena mogućnost kreiranja osobina koja karakteriše C#.

```
using System;
public class Singleton
{
  private static Singleton instance;
  private Singleton() {}
  public static Singleton Instance
  {
    get
    {
      if (instance == null)
      {
        instance = new Singleton();
      }
      return instance;
    }
  }
}
```

Ova implementacija ima dve glavne pozitivne osobine:

- kako se instanca kreira unutar `get` metoda osobine `Instance`, klasa može da dobije i dodatnu funkcionalnost;
- Instanciranje se ne izvršava sve dok objekat ne zatraži instancu i ovaj pristup se naziva lenjo instanciranje. Lenjo instanciranje izbegava instanciranje nepotrebnih singltona.

Glavni problem kod ove implementacije je to što ona nije sigurna za rad u višenitnim situacijama: svaka nit može ući u naredbu grananja i odlučiti da li da se kreira nova instanca.

Jedan način da se izbegne ovaj problem je statička inicijalizacija. Taj metod nije bio preporučen u [Gamma95] zato što C++ specifikacija ima nesaglasnosti oko redosleda kreiranja statičkih promenljivih. Međutim, kod C# su te nejasnoće uklonjene, pa se obrazac Singlton može implementirati na sledeći način:

```
public sealed class Singleton
{
  private static readonly Singleton instance = new Singleton();
  private Singleton(){}
  public static Singleton Instance
  {
    get
    {
      return instance;
    }
  }
}
```

U ovoj strategiji, instanca se kreira prvi put onda kada se referencira ma koji član klase. Za inicijalizaciju promenljive se u ovom slučaju brine zajednički jezički izvršni sistem CLR. Klasa je deklarisana kao `sealed` (zapečaćena, što je analogno sa `final` u Javi) i na taj način je blokirano nasleđivanje iz nje (jer bi izvedene klase mogle da prevaziđu ograničenje kod instanciranja). Pored toga, klasa je označena i sa `read only`, što znači da joj se može pristupiti radi promene samo kroz statičku inicijalizaciju ili preko konstruktora klase.

Ova implementacija se razlikuje od prethodne u tome što se oslanja na CLR pri inicijalizaciji promenljive. Već sa tim, ona daje odgovor na dva glavna problema koje obrazac Singlton pokušava da reši: globalni

pristup i kontrola instanciranja. Javna statička osobina obezbeđuje globalno vidljivu jedinstvenu tačku pristupa objektu. Nadalje, budući da je konstruktor privatan, klasa ne može biti instancirana iz spoljašnosti. pa promenljive referišu na jedinu instancu koja može postojati u softverskom sistemu. Jedini potencijalni problem kod ovog pristupa je to što programer ima manje kontrole nad mehanizmom instanciranja. Ovde, za razliku od prethodne implementacije, ne može da se koristi konstruktor različit od podrazumevanog koji bi, po potrebi, pre instanciranja obavio još i neke druge poslove. U najvećem broju slučajeva, ovakva implementacija obrasca Singlton pod .NET-om je sasvim zadovoljavajuća.

Ipak, kada višenitna aplikacija koja koristi singlton mora da pomeri instanciranje singltona za kasnije, ili kada mora da koristi konstruktor koji nije podrazumevan, ili kada treba izvršiti druge zadatke pre instanciranja, tada je potrebno osmisliti i realizovati drugačiju implementaciju. – tu se koristi mehanizam dvostrukog zaključavanja, koji sprečava da odvojene niti istovremeno kreiraju nove instance singltona.

Implementacija koja sledi dopušta da samo jedna nit pristupi kritičnoj oblasti (identifikovanoj sa `lock` blokom) tokom vremena kada nijedna instanca klase `Singlton` još nije kreirana.

```
using System;
public sealed class Singleton
{
  private static volatile Singleton instance;
  private static object syncRoot = new Object();
  private Singleton() {}
  public static Singleton Instance
  {
    get
    {
      if (instance == null)
      {
        lock (syncRoot)
        {
          if (instance == null)
            instance = new Singleton();
        }
      }
      return instance;
    }
  }
}
```

Prethodni pristup obezbeđuje da se, kada zatreba instanca, kreira tačno jedna instanca klase. Promenljiva `instance` je deklarisana kao `volatile`, čime je obezbeđeno da se dodela promenljivoj završi pre nego što se pristupi instanci. U ovoj implementaciji se za zaključavanje statički objekat `syncRoot`, a ne zaključava se sam tip i tako se izbegava uzajamno blokiranje. U ovom slučaju je razrešen i problem konkurentnih niti i obezbeđen nesmetan rad, uz istovremeno izbegavanje ekskluzivnog zaključavanja pri svakom pozivu osobine `Instance`. Naravno, i ovde je omogućena lenjo instanciranje – singlton se ne instancira sve dok ne zatreba.

Rezultati

Korišćenje obrasca Singlton dovodi do sledećih prednosti:

- Kontrola instanciranja. Singlton sprečava objekte da instanciraju njegove kopije, obezbeđujući da svi objekti pristupaju jednoj instanci.
- Fleksibilnost. Kako klasa kontroliše proces instanciranja, to klasa ima i fleksibilnost da promeni proces instanciranja.

Korišćenje obog obrasca skopčano je i sa određenim problemima:

- Mogućnost konfuzije pri pisanju programa. Kada se koristi singlton objekat (naročito ako je taj objekat definisan unutar biblioteke klasa) programeri koji ga koriste moraju da pamte da se instanciranje objekta ne vrši pomoću `new`. S obzirom da se može desiti da programeri nemaju pristup izvornom kodu biblioteke klasa, mogu se veoma iznenaditi kada uoče da ne mogu direktno da instanciraju klasu.
- Životni vek objekta. U nekim jezicima (kao što je C++) klase mogu ukloniti instancu singlton objekat i tada ostaje referenca koja "visi". Na sreću, to se ne može dogoditi kod jezika gde se automatski upravlja memorijom, (takvi su, na primer, .NET jezici i Java).

Obrazac Strategija

Kontekst
Mnogo puta se u razvoju softvera (a ista takva situacija je i u životu) mora donositi odluka o opštem pristupu za rešavanje problema. Većina je naučila da kretanje najlakšim putem pri donošenju kratkoročnih odluka može da dovede do ozbiljnih komplikacija pri dugoročnom razvoju.
Ipak, u pojedinim slučajevima ta lekcija nije savladana, pa su i dalje ne tako retki projekti u kojima se koncentriše samo na rešavanje trenutnih zahteva, a da se ne razmatra buduće održavanje i zahtevi. Razlozi za takav stav su obično sledeći:

- „Mi ne možemo znati kako će se menjati zahtevi i šta će biti novi zahtevi".
- „Ako bi pokušali da vidimo kako se zahtevi menjaju i šta mogu biti novi zahtevi, nikada nećemo izaći iz analize i krenuti u dizajn softvera".
- „Ako bi pokušali da pišemo kod tako da se lako može dodavati nova funkcionalnost, mi bi zauvek ostali u fazi dizajna softvera".
- „Nemamo ni vremena ni budžet za takav razvoj".

Ako se gleda tako crno-belo, izgleda da je pred članovima projektnog tima jedan od dva izbora:

- Prevelika analiza i/ili preveliki dizajn – u [Shallo03] autor duhovito primećuje da „previše analize dovodi do paralize".
- Kodiranje bez razmatranja dugoročnih elemenata i prelazak na novi projekat pre nego što se takvi problemi pojave u punom obimu, što se u [Shallo03] označava sa izrazom „napusti po isporuci".

Pristup koji se predlaže u ovom obrascu dizajna je (što je slučaj i kod drugih obrazaca) inspirisan opštim pristupom istaknutim u [Gamma95]:

- Programirati prema interfejsu, a ne prema implementaciji.
- Dati prednost kompoziciji u odnosu na nasleđivanje.
- Razmotriti šta je promenljivo u dizajnu. Dakle, umesto da se koncentriše na utvrđivanje šta sve može prouzrokovati promenu dizajna, razmotriti šta će se sve menjati a da se ne menja dizajn. Ovde je fokus na enkapsulaciju promenljivih pojmova.

Problem
Izbor algoritma koji treba biti primenjen zavisi od zahteva klijenta ili od podataka nad kojima se deluje.

Posledice
Obrazac Strategija definiše familiju algoritama. Na ovaj način, višestruka grananja i/ili uslovni skokovi mogu biti eliminisani. Uslov korišćenja je da se svi algoritmi pozivaju na isti način, tj. da svi imaju iste interfejse. Interakcija između konkretnih strategija i konteksta može zahtevati dodavanje u klasu kontekst metoda za dobijanje tipa strategije.

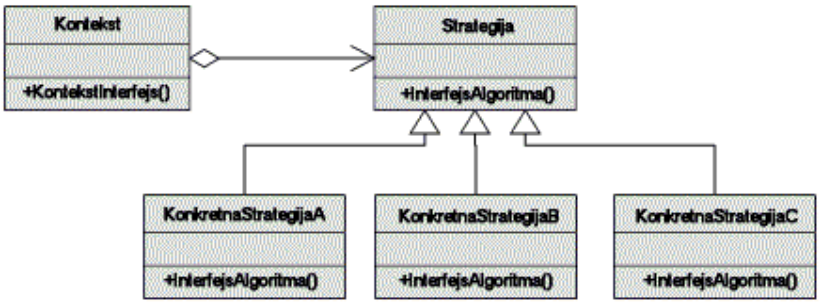
Učesnici
Klase i/ili objekti koji učestvuju u ovom obrascu su: `Strategija` (koja deklariše zajednički interfejs za sve podržane algoritme), `KonkretnaStrategija` (implementira algoritam koji koristi interfejs `Strategija`) i `Kontekst` (koji je proseljuje zahteve od klijenata prema strategiji, a koji je konfigurisan od strane instance `KonkretnaStrategija` i koji čuva referencu na objekat `Strategija`). Strategija i kontekst svojom interakcijom implementiraju izabrani algoritam pri čemu ponekad strategija mora da pita kontekst.

Rešenje
Prema GoF-u (videti [Gamma95]), obrazac strategija definiše familiju algoritama, enkapsulira svaki od njih i omogućuje njihovu međusobnu razmenu. Strategija dopušta da algoritam varira nezavisno od klijenta koji je koristi – koji će se algoritam koristiti zavisi od konteksta dešavanja. Na ovaj način je izbor algoritma razdvojen od njegove implementacije.
Obrazac strategija je baziran na nekoliko principa:

- Objekti imaju odgovornosti.
- Različite, specifične implementacije ovih odgovornosti se manifestuju kroz korišćenje polimorfizma.
- Postoji potreba za održavanjem nekoliko različitih implementacija nečega što je, konceptualno, isti algoritam.

- Potrebno je razdvojiti ponašanja koja se događaju u različitim domenima problema – na taj način je omogućeno da promena jedne klase (odgovorne za jedno ponašanje) ne zahteva promene u drugim klasama.

Implementacija
Klasa `Kontekst` koja koristi algoritam sadrži referencu na instancu apstaraktne klase `Strategija`, koja sadrži apstraktni metod `InterfejsAlgoritma` koji određuje kako će se pozvati algoritam.



slika 4.10. Strategija

Tehnički, obrazac Strategija enkapsulira algoritme. Međutim, u praksi se koristi i za enkapsuliranje ma koje vrste pravila koja treba realizovati u softverskom sistemu.

Rezultati
Obrazac strategija zahteva da algoritam koji je enkapsuliran bude izvan klase (klase `Kontekst` iz prethodnog dijagrama) koja ga koristi. Ovo znači da se informacije koje zahtevaju strategije moraju prosleđivati na isti način.
Jedini ozbiljni problem sa ovim obrascem je potreba za kreiranjem dodatnih klasa.

Obrazac Apstraktna fabrika

Kontekst
Prema GoF-u, apstraktna fabrika treba da obezbedi interfejs za kreiranje familije povezanih (tj. zavisnih) objekata bez specifikacije njihove konkretne klase [Gamma95].
Ponekad veći broj objekata treba da bude koordinisano instanciran i apstraktna fabrika obezbeđuje tu koordiniranost. Veoma često, postojanje višestrukog grananja ukazuje da ili postoji potreba za polimorfizmom, ili se radi o odgovornostima koje su pogrešno dodeljene, pa treba razmotriti opštije rešenje koje se vodi na apstrakciju i kreiranje hijerarhije nasleđivanja u kojoj će svaki od objekata imati adekvatnu odgovornost.
Iako produbljivanje hijerarhije nasleđivanja u jednostavnijim slučajevima daje dobre rezultate, ono ima toliko nedostataka u složenijim slučajevima, da pojedini autori (videti [Shallo03]) čak smatraju da je bolje da ostane višestruko grananje nego da se formira duboka hijerarhija nasleđivanja. Najvažniji problemi su:

- Kombinatorna eksplozija – za svaku od familija i za svaku novu familiju koja bude dodavana u budućnosti, mora da se naprave nove konkretne klase.
- Nejasno značenje – rezultujuće klase ne pomažu raščišćavanju šta se događa.
- Potreba favorizovanja kompozicije – preduboka hijerarhija nasleđivanja krši jedno od osnovnih pravila GoF-a „favorizuj kompoziciju u odnosu na nasleđivanje“.

Kod ovog obrasca objekti-fabrike instanciraju objekte iz klasa koje su kreirane.

Problem
Treba instancirati familije povezanih objekata.

Učesnici
Apstraktna fabrika definiše interfejs – kako kreirati kolaboraciju za svakog člana zahtevane familije objekata. Obično se svaka familija objekata kreira sa posebnom konkretnom fabrikom.
Klase i/ili objekti koji učestvuju u tom obrascu su: `ApstraktnaFabrika` (deklariše interfejs za operacije kojima se kreiraju apstraktni proizvodi), `KonkretnaFabrika` (implementira operacije radi kreiranja konkretnog objekta-proizvoda), `ApstraktniProizvod` (deklariše interfejs za tip objekta-proizvoda),

`Proizvod` (definiše objekat-proizvod koji se kreira odgovarajućom fabrikom i implemetira interfejs `ApstraktniProizvod`) i `Klijent` (koristi interfejse deklarisane sa klasama `ApstraktnaFabrika` i `ApstraktniProizvod`).

Posledice

Obrazac izoluje pravila o tome koji objekti da se koriste od logike koja definiše kako se objekti koriste.
Primena apstraktne fabrike dovodi do sledećih posledica:

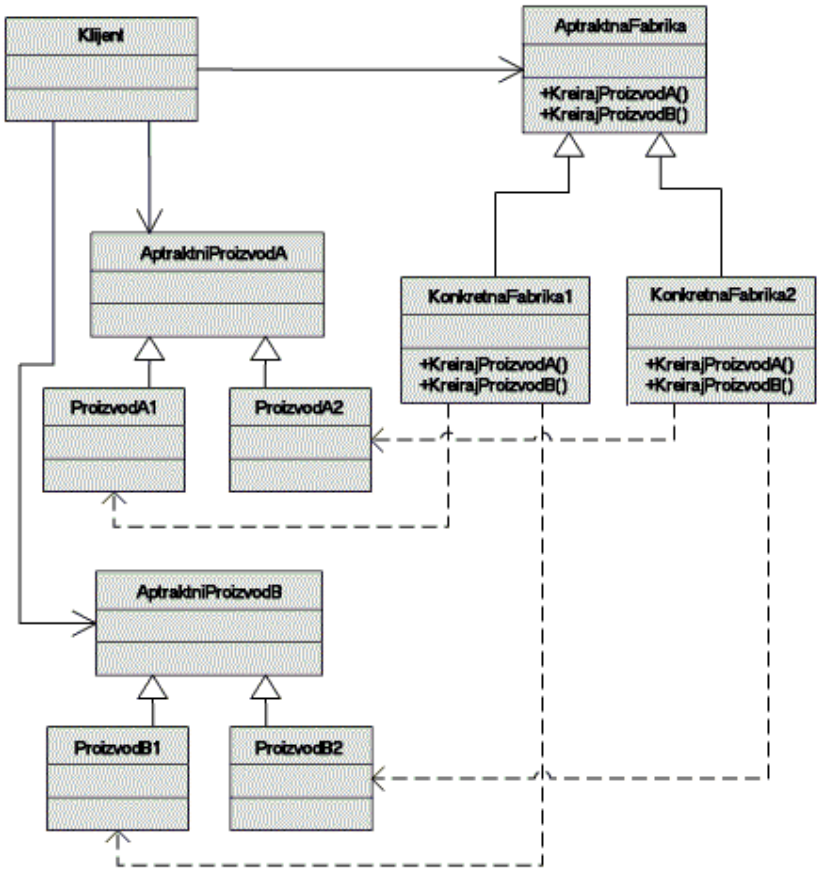
- Klijentski objekat sam zna koga da pita za to koji su mu objekti potrebni i kako da ih koriste.
- Apstraktna fabrika sadrži klasu koja određuje kakvi objekti mogu da se instanciraju tako što definiše metod za svaku od različitih vrsta objekata.
- Konkretne fabrike specificiraju koji objekti treba da budu instancirani.

Rešenje

Koordinira kreaciju familija objekata. Obezbeđuje način da se pravila o načinu izvršavanja instanciranja izbace iz klijentskog objekta koji koristi novokreirane objekte.

Implementacija

Implementacija se svodi na definisanje apstraktne klase koja određuje objekte koje treba napraviti, pa implementira jednu konkretnu klasu za svaku od familija. Za ostvarenje tog cilja se mogu koristiti tabele ili datoteke.



slika 4.11. Apstraktna fabrika

Slika 4.11 prikazuje klijenta (instancu klase `Klijent`) koji koristi objekte izvedene iz dve različite klase na serveru (`ApstraktniProizvodA` i `ApstraktniProizvodB`). Ovakav dizajn uprošćava, sakriva implementaciju i pravi sistem lakšim za održavanje.
Klijent ne zna koja će od konkretnih implementacija serverskih objekata biti korišćena, zato što objekat-fabrika ima odgovornost da ih kreira. nadalje, klijent čak ne zna, koja će konkretna fabrika biti korišćena, zato

što on samo zna za objekat `ApstraktnaFabrika` – on ima referencu na objekat `KonkretnaFabrika1` ili `KonkretnaFabrika2`, ali klijent ne zna tačno na koji od njih.
Obrazac apstraktna fabrika pruža novu vrstu dekompozicije – dekompozicija po odgovornosti. Korišćenjem ovog obrasca, problem se dekomponuje na dve celine: ko koristi konkretne objekte, ko određuje kojiće konkretni objekti biti korišćeni. Korišćenjem ovog obrasca se ukazuje da domen problema ima različite familije prisutnih objekata i svaka familija objekata se koristi pod različitim okolnostima.
Po identifikovanju familija objekata i članova svake od familija, mora se odlučiti kako treba implementirati svaki od slučajeva, tj. svaku od familija objekata.
Ponekad će postojati familije objekata, ali neće biti kontrole njihove instanciranosti – već se zahteva dinamičan pristup. Primeri za to je kada postoji konfiguraciona datoteka koja specificira koji će se objekti koristiti, pa informacija iz tog konfiguracionog fajla određuje da se instancira korektan objekat.
U jezicima koji podržavaju refleksiju, kao što su Java i C#, pristup sa konfiguracionom datotekom može da ode i još jedan korak dalje. Tu informacija (npr. string koji predstavlja ime klase) koja određuje koja se klasa instancira može biti iskorišćena da se u letu napravi instanca adekvatne klase. Ovakav pristup je, kao što ćemo dalje videti, primenjen i pri dizajnu EaWeb softverskog sistema.

4.2.7. Obrasci dizajna primenjeni na arhitekturu rešenja

Višeslojna aplikacija

Kontekst
Javlja se kada treba dizajnirati softverski sistem sastavljen od velikog broja komponenti koje se prostiru kroz mnogobrojne nivoe apstrakcije.

Problem
Kako struktuirati softverski sistem tako da on podržava operativne zahteve lakoće održavanja, ponovnog korišćenja, skalabilnosti, robusnosti i sigurnosti?

Posledice
Pri struktuiranju aplikacije, moraju se razmotriti sledeće posledice koje porizilaze iz konteksta okruženja:

- Lokalizacija promena na jedan deo rešenja tako da je minimiziran uticaj na ostale delove, dovodi do smanjenja posla na debagiranju, testiranju i ispravljanju koda, olakšava održavanje aplikacije i povećava ukupnu fleksibilnost sistema.
- Razdvajanje razmatranja na komponente (npr. razdvajanje korisničkog interfejsa od poslovne logike i razdvajanje poslovne logike od baze podataka) povećava fleksibilnost, skalabilnost i olakšava održavanje sistema.
- Komponente treba da budu ponovno korišćene od strane većeg broja aplikacija.
- Treba omogućiti da nezavisni timovi rade na delovima softverskog sistema, da zavisnost među timovima bude najmanja moguća, i da timovi razvijaju kod po dobro definisanim interfejsima.
- Individualne komponente treba da budu kohezivne.
- Nepovezane komponente treba da budu slabo spregnute.
- Različite komponente rešenja (tj. sistema) su nezavisno razvijane, održavane i ažurirane.
- Prelazak izvršavanja kroz previše granica komponenti ima negativan efekat na performanse sistema.
- Da bi aplikacije na web-u bile sigurne i pristupačne, potrebno ih je distribuirati na nekoliko fizičkih slojeva, pa se neki delovi sistema postavljaju iza „požarnog zida“, dok se druge komponente postavljaju tako da budu dostupne sa Interneta.
- Da bi se obezbedila pouzdanost i dobre performanse, komponente sistema treba da budu podložne softverskom testiranju.

Rešenje
Razdvojiti komponente rešenja u slojeve. Komponente u svakom sloju treba da budu kohezivne i da su, grubo gledano, na istom nivou apstrakcije. Svaki nivo treba da bude slabo spregnut u odnosu na slojeve koji se nalaze ispod njega. U radu [Buschm96] se proces raslojavanja opisuje na sledeći način: „Počinje se od najnižeg nivoa apstrakcije, nazvanog Sloj 1. Taj sloj je osnova sistema koji se gradi. Podižu se potom naviše

nivoi apstrakcije tako što se Sloj J postavlja na Sloj J-1 sve dok se ne dostigne najviša funkcionalnost, koja će butu označena sa Sloj N".

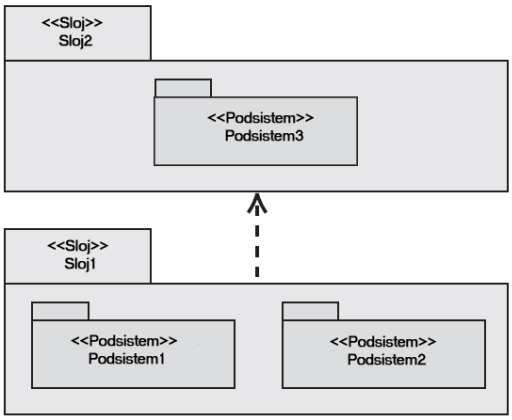
slika 4.12. Slojevi

Struktura

Ključ za višeslojne aplikacije je upravljanje zavisnostima. Komponente iz jednog nivoa mogu imati interakciju samo sa komponentama iz istog sloja ili sa komponentama iz nižih slojeva. Na ovaj način su smanjene zavisnosti među komponentama na različitim nivoima.

Postoje dva pristupa kod višeslojnih aplikacija: striktnost i relaksiranost. Striktna višeslojna aplikacija sadrži komponente koje imaju interakciju samo sa komponentama u tom sloju i komponentama u sloju koji se nalazi neposredno ispod njih. Ako je slika 4.12. predstavlja striktnu višeslojnu aplikaciju, tada Sloj J može imati interakciju samo sa Slojem J-1, Sloj J-1 samo sa Slojem J-2 itd. Relaksirana višeslojna aplikacija može povećati efikasnost, zato što sistem ne mora da prosleđuje proste pozive iz jednog sloja u drugi. Sa druge strane, relaksirana višeslojna aplikacija ne obezbeđuje isti nivo izolacije između slojeva, pa je teže izvršiti izmenu komponenti u nižim slojevima a da to ne utiče na više slojeve (videti [Bologn02]).

Za velike sisteme koji uključuju ogroman broj softverskih komponenti, obično postoji veliki broj komponenti u istom sloju koje nisu kohezivne. U takvim slučajevima, svaki sloj može biti dalje dekomponovan u kohezivnije podsisteme. Sledeća slika demonstrira moguću UML notaciju za predstavljanje slojeva koji sadrže više podsistema.



slika 4.13. UML reprezentacija slojeva sastavljenih od podsistema

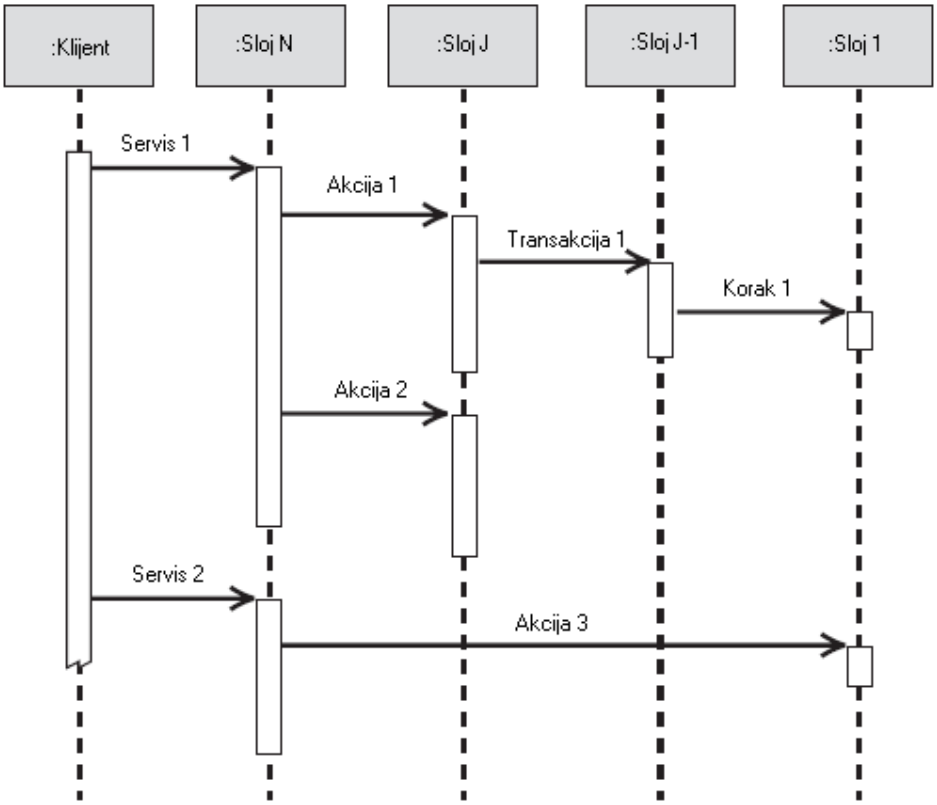
Klasična višeslojna aplikacija se često kombinuje sa sledećim tehnikama:

- Nadklasa sloja (videti [Fowler03]) – ako komponente u sloju dele skup zajedničkih ponašanja, tada se izvlači to ponašanje u zajedničku klasu iz koje se izvode sve te komponente. Ova tehnika ne samo što olakšava održavanje i promoviše ponovno korišćenje, već takođe smanjuje i zavisnosti između slojeva time što dopušta da zajedničko ponašanje bude pozvano tokom izvršavanja referisanjem na nadklasu a ne na konkretnu komponentu.
- Apstraktni interfejs – on se definiše za svaku komponentu u sloju koja biva pozvana od komponente iz višeg sloja. Viši sloj, dakle, pristupa komponentama nižeg sloja kroz apstraktni interfejs, bez direktnog poziva komponente.

- Fasada sloja – za veće sisteme je uobičajeno da se koristi obrazac Fasada kako bi se obezbedio jedinstven interfejs prema sloju, umesto da se razvija apstraktni interfejs za svaku izloženu komponentu (videti [Gamma95]). na ovaj način se postiže najniže grupisanje među slojevima.

Dinamika

U osnovi postoje dva modela interakcije unutar višeslojne aplikacije: odozgo-naniže (eng. top-down) i ododzdo-naviše (eng. bottom-up). U modu odozgo-naniže, spoljašni entitet interaguje sa najvišim slojem aplikacije. Najviši sloj koristi jedan ili više servisa nižih slojeva. U daljem razmatranju se pretpostavlja da je spoljašnji entitet klijentska aplikacija, koja ima interakciju sa najvišim slojem serverski zasnovane višeslojne aplikacije, koja je svoju funkcionalnost izložila kao skup servisa (upravo je takav slučaj sa razvijenim EA softverskim sistemom). Ta situacija je opisana UML sekvencnim dijagramom na slici 4.14.

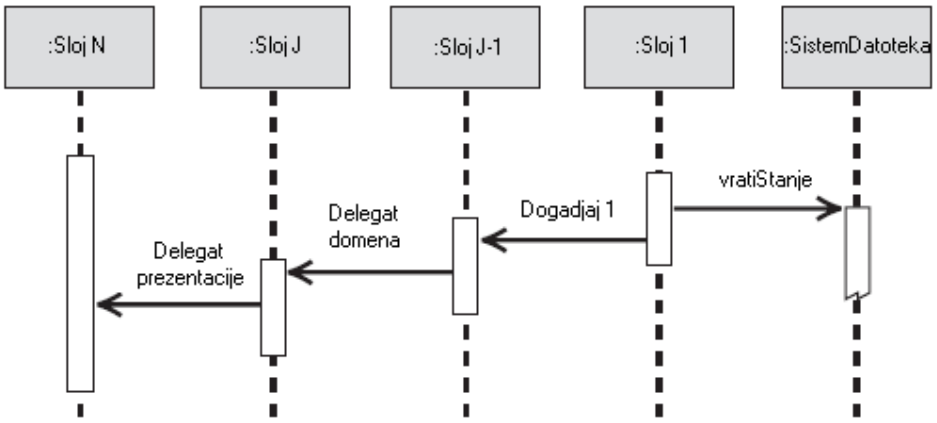


slika 4.14. Sekvencni dijagram za scenario odozgo-naniže

U ovom scenariju, klijentska aplikacija koristi skup servisa ponuđenih od strane serverski zasnovane aplikacije. Servisi koji se mogu koristiti su izloženi u najvišem sloju te višeslojne aplikacije. Dakle, klijent može da ima interakciju samo sa najvišim nivoom i nema direktno znanje o nižim nivoima.

Situacija opisana prethodnim dijagramom (da jedan dolazeći poziv u Sloju N kao rezultat daje više odlazećih poziva) se često događa kada servis visokog nivoa agregira više servisa nižeg nivoa ili koordinira izvršavanje većeg broja nižih servisa koji se moraju izvršavati u datom redosledu. Uočava se da dijagram sa slike 4.14. predstavlja relaksiranu višeslojnu aplikaciju: Servis 2 je implementiran tako da zaobilazi sve međuslojeve i direktno poziva Sloj 1. Na kraju, uočava se da poziv najvišeg sloja ne mora nužno da dovede do pozivanja svih slojeva. Tako, na dijagramu, sekvenca poziva Servis1 - Akcija 2 ne poziva niže nivoe. Takva situacija se često javlja kada treba podržati keširanje u višeslojnoj aplikaciji.

U modu odozdo-naviše, Sloj 1 detektuje situaciju koja je pogodila više slojeve. Scenario koji sledi podrazumeva da Sloj 1 nadgleda stanje nekog spoljašnjeg entiteta, kao što je sistem datoteka na serveru gde se izvršava aplikacija. Kada Sloj 1 detektuje promenu, on „ispali“ događaj koji je izložen komponentama na sloju J-1. Komponenta u Sloju J-1poziva povratni delegat ka Sloju J, pri čemu se ažurira stanje domenskog sloja. Potom domenska komponenta korišćenjem delegata, obaveštava Sloj N da je bila ažurirana.



slika 4.15. Sekvencni dijagram za scenario odozdo-naviše

Uočava se da u prethodna dva slučaja (odozgo-naniže i odozdo-naviše) slojevi interaguju na drugačiji način. U scenariju odozgo-naniže, viši nivoi direktno pozivaju niže nivoe i na taj način zavise od njih. U scenariju odozdo-naniže, niži nivoi komuniciraju sa višim nivoima kroz događaje, povratne pozive (eng. callback) i delegate. Taj nivo indirekcije, slično kao u obrascu Posmatrač, čini da niži nivoi nisu zavisni od viših.

Implementacija

Postoje dva pristupa u implementaciji obrasca višeslojne aplikacije:

- Kreiranje sopstvene sheme slojeva – u radu [Buschm96] se iscrpno razmatra ovakav pristup. Struktura procesa određivanja slojeva treba da bude:
    - Korišćenjem dobro definisanog skupa kriterijuma grupisati funkcionalnost rešenja u skup slojeva i definisati servise koje obezbeđuje svaki od slojeva. Ovo je iterativan proces u kome se pokušava sa više kombinacija kriterijuma, različitim brojevima nivoa, različitim dekompozicijama funkcionalnosti i raznovrsnim dodelama odgovornosti za servise. UML sekvencni dijagrami koji opisuju interakciju između slojeva i komponenti rešenja su idealni alat za shvatanje posledica kompromisa koji se nužno moraju postići pri definisanju ma koje sheme slojeva.
    - Definisati interfejs između svakog sloja i protokole koji su neophodni radi komunikacije tog sloja sa ostalim. Da bi se izbegla zavisnost nižih nivoa od viših, za komunikaciju koja putuje kroz stek se koriste prethodno pomenute tehnike asinhronog procesiranja, povratnih poziva i događaja. I ovde UML dijagrami imaju važno mesto u obezbeđivanju da je kreiran kompletan i konzistentan skup interfejsa. U ovom koraku naročito treba voditi računa o tome koliko puta se tokom izvršavanja datog scenarija prelazi granica između slojeva i treba biti spreman za refaktorizaciju dizajna u slučaju kada su povezane komponente iz različitih slojeva (npr. kada komponente iz Sloja J direktno koriste komponente Sloja J-1). Pri refaktorizaciji je od velike pomoći i često se koristi prethodno opisan obrazac Fasada, kao i drugi obrasci dizajna koji dovode do razdvajanja komponenti.
    - Dizajnirati implementaciju slojeva. U realizaciji ovog zadatka se obično koriste tradicionalne tehnike objektno – orjentisanog dizajna. U najvećem broju slučajeva se ovde koristi neki od obrazaca dizajna koji omogućuje prebacivanje između višestrukih implementacija prethodno definisanog interfejsa sloja (to može biti prethodno opisana Strategija, ili Adapter, ili Most). Značaj takvog dizajna najviše dolazi do izražaja pri testiranju interfejsa i implementacija sloja. Još jedna veoma važna odluka je odluka o načinu tretiranja grešaka – konzistentna strategija obrade grešaka mora biti definisana u svim nivoima, tj. slojevima. Ta strategija bi trebalo da obrađuje grešku na najnižem mogućem nivou (što bliže njenom izvoru), da se izbegava izlaganje apstrakcija nižih

nivoa višim slojevima kroz izuzetke i da (ako izuzetak baš mora da eskalira) izuzetci nižeg nivoa budu konvertovani nešto što ima smisla na višem nivou.

- Preuzimanje (tj. ponovno korišćenje) postojeće sheme slojeva. Kanonska troslojna aplikacija, koja se sastoji od sloja prezentacije, domenskog sloja i sloja podataka je često dobar izbor za postavljanje slojeva kod dizajna višeslojne aplikacije.

Rezultat

Višeslojna aplikacija ima sledeće prednosti:

- Održavanje i nadogradnja rešenja su lakše, zato što su slojevi više razdeljeni, zato što je veća kohezija između slojeva i zbog mogućnosti lake promene implementacije za interfejs sloja.
- Novonapravljena rešenja mogu da ponovo iskoriste funkcionalnost koju su izložili raznovrsni slojevi, naročito onda kada su interfejsi slojeva dizajnirani tako da se razmišljalo o njihovom ponovnom korišćenju.
- Olakšan je distribuisani razvoj i izvršavanje ako se aplikacija može distribuirati poštujući granice slojeva.
- Distribuisanje slojeva na više fizičkih lokacija poboljšava skalabilnost, otpornost na greške i performanse.
- Prednosti pri testiranju zbog postojanja dobro definisanih interfejsa za slojeve, kao i zbog mogućnosti lakog prelaska sa jedne implementacije interfejsa sloja na drugu.

Naravno, postoje i određeni nedostaci višeslojne aplikacije:

- Dodatne aktivnosti koje se dešavaju pri prolasku kroz slojeve, umesto direktnog poziva komponente mogu negativno da utiču na performanse sistema. Da bi se performanse održale, često se koristi relaksirani pristup, u kom viši sloj može pozivati ma koji od nižih slojeva.
- Razvoj aplikacija koje korisnik intenzivno koristi može potrajati nešto duže nego što je to slučaj kod monolitnih aplikacija (kod kojih se ne izdvajaju slojevi).
- Korišćenje slojeva pomaže kontroli i enkapsulira složenost kod velikih aplikacija, ali dodaje kompleksnost manjim, prostim aplikacijama.
- Promene kod interfejsa u nižim slojevima imaju tendenciju da se prošire i na više slojeve, pogotovo u slučaju kada se koristi relaksirani pristup.

Pojmovi kolaboracije

Klase, objekti, komponente i interfejsi su osnovni gradivni blokovi modernog softvera. Neki od tih elemenata enkapsuliraju domen problema, a drugi obezbeđuju sistemsku infrastrukturu i tehničku arhitekturu. Svaki gradivni blok obezbeđuje neku korisnu funkciju, ali njihova prava moć leži u kompoziciji pojedinih elemenata u kolaborativno rešenje. Da bi se obezbedio potreban nivo kolaboracije, softverski elementi moraju da se uklapaju u principe organizacije i moraju jedan drugom izložiti standardizovane interfejse. Čak i kada komponente nisu slične, jedan elemenat se mora prilagoditi drugom, ili se oba moraju prilagoditi standardu. Ove teme se detaljno razmatraju u radovima [Jezier02] i [Olive03].

Kolaboracija zasnovana na servisima

Kolaboracija bazirana na servisima funkcioniše dobro tamo gde aplikacija koja koristi servis nema kontrolu nad udaljenim servisima ili kada mora zajednički funkcionisati sistem izgrađen na različitim programskim jezicima i/ili različitim platformama.

Interfejsi zasnovani na servisima izlažu jednu instancu interfejsa, koja obezbeđuje usluge potencijalnim mušterijama. U kontekstu web servisa (koji će biti detaljno opisani u sledećem poglavlju) Microsoft softverski servis definiše kao: „diskretnu celinu aplikativne logike koja izlaže interfejs baziran na porukama, kome je pogodno pristupati kroz mrežu".

Servis ne zavisi od procesa koji ga je pozvao, on je samodovoljan i nezavistan od konteksta. Ovo dopušta da ma koji potencijalni klijent na mreži pristupi servisu. Servisi su dobro definisani kao ugovor koji specificira format zahteva servisu i format pridruženog odgovora.

Koncept logički povezanih servisa (mada ne nužno baziranih na porukama) je bio korišćen pri razvoju aplikacija čak i pre postanka distribuisanih aplikacija (takav je npr. slučaj sa Microsoft Windows GDI

bibliotekom koja obezbeđuje grafički servis i sa ODBC (eng. Open DataBase Connectivity – otvorena veza sa bazom podataka) API-jem koji izlaže servise za pristup bazi podataka.

Servis-orjentisana arhitektura

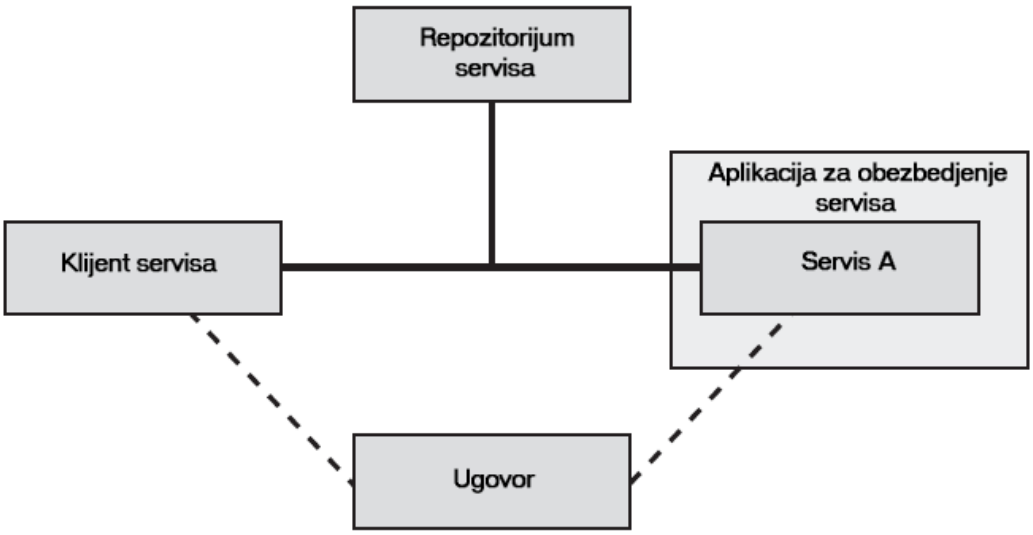
Servis-orjentisana arhitektura (skraćeno SOA) primenjuje koncept servisa na distribuisane složene aplikacije (videti [Wilson02]). U SOA, svaka od aplikacija izlaže poslovne funkcije visokog nivoa kao servise koje mogu konzumirati druge aplikacije. Zbog povećanog opsega i povećane složenosti servis-orjentisanih rešenja, servis-orjentisana arhitektura mora obezbediti dodatne funkcije koje prevazilaze sposobnost pozivanja udaljenog servisa. Najvažnije od ovih funkcija su:

- Određivanje lokacije tokom vremena izvršavanja. Za samostalnu, monolitnu aplikaciju je lako da lokalizuje servis GDI, jer je on implementiran u dinamički vezanoj biblioteci `gdi32.dll`. Međutim, „eneterprise" servisi mogu biti distribuirani na veći broj računara, mreža ili uređaja, gde neki od servisa mogu promeniti svoje lokacije. Stoga, lociranje servisa u distribuisanoj servis-orjentisanoj arhitekturi može biti dosta složen zadatak.
- Određivanje saglasnosti o zajedničkom formatu kod servisa i klijenta. Kada se pronađe korektan servis, aplikacija koja ga koristi mora da bude sposobna da dinamički odredi koji protokoli se koriste za pristup servisu, kako se formatira zahtev i koji tipovi odgovora se očekuju. S obzirom da servisi mogu biti implementirani na raznovrsnim jezicima i platformama, cilj da se servis i klijent saglase oko zajedničkog formata može biti izazovan zadatak.

Ugovori servisa

Kada metod poziva drugi metod unutar aplikacije, potpis metoda definiše „razumevanje" između metoda i pozivaoca, npr. broj i tipove parametara prosleđenih u metod i tip vraćen pri završetku rada metoda. Pozivalac i metod kreiraju određeni broj implicitnih pretpostavki (npr. da se oboje izvršavaju unutar istog procesa i da dele isti memorijski prostor, da oba metoda koriste isit programski jezik itd.). U svetu distribuisanih SOA, mnoge od ovih pretpostavki nisu više validne i potrebno je da se sve pretpostavke eksplicitno iskažu u ugovoru o servisu.

Ugovor o servisu mora da specificira implementaciju komunikacionog kanala koji povezuje klijente servisa i aplikaciju koja obezbeđuje servis, npr. mrežni protokol. Ugovor o servisu mora takođe specificirati koje vrste poruka servis može konzumirati ili proizvoditi i za svaku vrstu poruka koja je uključena u interakciju se određuje detaljna shema.



slika 4.16. Poziv servisa kod SOA

Jedan servis ponekad treba da podrži veći broj ugovora.

Za pozivanje udaljenog servisa potrebni su sledeći koraci:

1. Otkrivanje. Klijent servisa (ma koja aplikacija koja pristupa servisu) pita repozitorijum servisa, koji obezbeđuje lokaciju željenog servisa.

2. Pregovaranje. Klijent servisa i aplikacija koja obezbeđuju servis se dogovaraju o formatu komunikacije koji su specificirani ugovorom.
3. Pozivanje. Klijent servisa poziva servis.

4.2.8. Web Servisi

Web servisi obezbeđuju SOA implementaciju baziranu na standardima. Web servisi definišu sklop tehnologija i protokola koji u velikoj meri uprošćavaju kreiranje rešenja koja su bazirana na skupu aplikacija koje međusobno sarađuju. Među svim principima i tehnologijama web servisa, ključni značaj imaju:

- ugovor o komunikaciji između klijenta i servisa;
- interoperabilnost.

Ugovor o komunikaciji

Kao metafora za komunikaciju između sistema, obično se koristi stek protokola, gde je uobičajen primer slojevi OSI modela. Stek protokola opisuje komunikaciju kao skup slojeva servisa na obe strane komunikacije, gde slojevi višeg nivoa koriste slojeve nižeg nivoa. Na primer, aplikativni protokoli kao što su FTP ili HTTP mogu da koriste TCP/IP kao komunikacioni protokol.
Ugovor o komunikaciji detaljno definiše sve slojeve stek protokola koji se koriste u komunikaciji. Ugovor web servisa razrešava dva primarna aspekta komunikacije:

- zajednički komunikacioni kanal;
- predstavljanje podataka i sheme poruka.

Zajednički komunikacioni kanal

Da bi aplikacije mogle da komuniciraju, one moraju da koriste kompatibilne protokole. Protokol TCP/IP je postaao podrazumevani stek protokola za komunikaciju na nižem nivou. Najveći broj modernih operativnih sistema obezbeđuje ugrađenu TCP/IP funkcionalnost. Aplikacija koja koristi korektno konfigurisan TCP/IP stek može komunicirati sa ostalim aplikacijama koje koriste TCP/IP stek u istoj lokalnoj mreži. Sledeća slika prikazuje komunikacioni kanal između dve aplikacije sa kompatibilnim stekovima protokola.

slika 4.17. Komunikacioni kanal i stek protokola za TCP/IP

Na primer, Aplikacija A može napraviti zahtev za web servis koji je izložila Aplikacija B. Stek protokola kao Aplikacije A razlaže zahtev sa aplikativnog nivoa na veći broj paketa sa podacima nižeg nivoa koji se prenose kroz mrežu. Stek protokola Aplikacije B prevodi te pakete nazad (na aplikativni nivo) u poziv web servisu. Web servis odgovara čime se pokreće proces iste vrste, ali sada u suprotnom smeru. Široka primenjivost TCP/IP protokola čini ga idealnom osnovom za interoperabilnu komunikaciju. Međutim, s obzirom da je TCP/IP protokol niskog nivoa, on se ponaša slično telefonskoj liniji kod telefoniranja – obezbeđuje kanal za komunikaciju, ali ne specificira zajednički jezik na kome će se odvijati ta komunikacija.
HTTP je protokol koji se nalazi u sloju iznad TCP/IP i on obezbeđuje najosnovije konvencije za zahtevanje odgovora od spoljašnjosti. Jednostavnost ovog protokola je pomogla da HTTP dobije široku podršku kao protokol za prenos informacija kroz Internet i kroz korporacijske „požarne zidove“ (eng. firewall). Univerzalnost njegovog korišćenja je učinila da on postane idealan za preusmeravanje poruka, ali njegova sposobnost prolaska kroz najveći broj požarnih zidova predstavlja uzrok stalne zabrinutosti administratora za sigurnost u IT sektorima kompanija.
Slika koja sledi predstavlja rezultujući stek protokola kada se doda HTTP.

slika 4.17. Komunikacioni kanal i stek protokola uz dodatak HTTP

Predstavljanje podataka i sheme poruka

Drugi deo ugovora o komunikaciji određuje šta se to prosleđuje kroz komunikacioni kanal. Ovaj deo ugovora treba da definiše tri stvari:

- Format za reprezentaciju podataka – da bi aplikacije uspešno međusobno komunicirale, sve one moraju da se saglase o zajedničkom skupu definicija za podatke koji prolaze kroz komunikacioni kanal. Stek protokola za web servise već sadrži podatke koji idu u jednu ili drugu stranu, a koji se nalaze u tekstualnom formatu. Ali šta ti podaci predstavljaju? da li se radi o serijalizovanom objektu, o nizu celih brojeva, ili o XML dokumentu, ili o nečem drugačijem od svega pobrojanog? Postoje dva načina da se odgovori na prethodno pitanje tj. da se obezbedi informacija šta podaci predstavljaju – to su obezbeđivanje spoljašnjeg opisa ili korišćenje samoopisujućih podataka.
  - Spoljašnji opis definiše shemu za podatke u nekom spoljašnjem obliku. Jezik za definiciju interfejsa IDL koji se koristi u Microsoft-ovom DCOM-u i distribuisanom RPC-u je jedan primer ovakvog pristupa. Drugi primer takvog pristupa je CORBA. Spoljašnji pristup opisu podataka ograničava interoperabilnost, zato što svi računari koji sarađuju moraju imati pristup tim spoljašnjim opisima.
  - Samoopisujući podaci sadrže opis podataka umetnut u same podatke. Korišćenje ovog pristupa pojačava interoperabilnost, jer se podaci mogu prosleđivati a da se ne konsultuje spoljašnji opis podataka.

  Web servisi koriste XML kao format za reprezentaciju podataka. Korišćenje XML-a obezbeđuje sledeće prednosti:
  - XML je baziran na tekstu, pa je stoga kompatibilan sa komunikacionim kanalom.
  - XML je industrijski standard sa širokom podrškom industrije i korisnika.
  - XML je samoopisujući.
  - XML je interoperabilan.
  - XML parseri postoje na skoro svim platformama i postoji mnogo alata za razvoj koji olakšavaju razvoj XML aplikacije.
- Shema poruka – poruka je osnova komunikacije kod web servisa, pa je imperativ u komunikaciji da sve strane koje komuniciraju jasno shvate sadržaj poruke. Ugovor o komunikaciji mora da specificira sve poruke sa zahtevima koji se šalju servisima i odgovarajuće poruke-odgovori koje se vraćaju kao rezultat. Takođe se mora specificirati i sadržaj svake poruke. Ovaj zadatak obično uključuje identifikaciju elemenata-podataka kod svake poruke, specificiranje tipa podatka za elemente i specificiranje ma kog uslova koji je vezan za tip ili za odnos među tipovima. SOAP (eng. Simple Object Access Protocol) deli poruke na dve sekcije: opciono zaglavlje i obavezno telo poruke. Zaglavlje sadrži informacije koje se odnose na infrastrukturu komunikacije i servisa, a telo sadrži poslovno-orjentisan sadržaj poruke. Zbog svog značaja, SOAP će biti detaljnije opisan u posebnom poglavlju.
- Povezivanje poruka sa servisima – po definisanju svih vrsta poruka, poruke se moraju pridružiti komunikacionom kanalu. SOAP ne mora da radi samo sa HTTP-om, već može raditi sa ma kojim drugim tekst-baziranim komunikacionim protokolom, kao što je SMTP. Dakle, servis može podržavati više od jednog komunikacionog protokola. Za opis sheme poruka i povezivanja poruka sa servisima, koristi se Jezik za opis web servisa WSDL (eng.Web Service Description Language), o kome će biti više reči kasnije.

Interoperabilnost

U prethodnoj diskusiji o komunikacionim kanalima i opisivanjima poruka, jako je podvučen značaj interoperabilnosti, a tu su web servisi u prednosti u odnosu na alternative.

Naime, jedan od ključnih nedostataka kod tradicionalnog pristupa distribuisanom izračunavanju je da aplikacije zavise od tehnologija za komunikaciju, protokola i/ili formata koji su vlasništvo konkretne kompanije – nisu slobodno dostupni. Web servisi su u potpunosti bazirani na skupu široko podržanih otvorenih standarda koji su nezavisni od platforme. Zbog toga skoro svaka platforma ima jednu ili više implementacija za stek protokola kod web servisa. Ovo značajno smanjuje napore i troškove u razvoju kolaborativnih aplikacija.

SOAP i WSDL

Web servisi koriste XML-bazirane poruke kao fundamentalni način za prenos podataka, kako bi se premostili jazovi između sistema koji koriste neuklapajuće modele komponenti, operativne sisteme i programske jezike ([Turner03]). Preciznije rečeno, web servisi su izgrađeni na standardima kao što su protokol za poziv udaljenih metoda SOAP i jezik za opis Web servisa WSDL .

SOAP tj. prosti protokol za pristup objektima je industrijski protokol koji je dizajniran za prenos XML poruka kroz veći broj protokola (u [W3C]). Specifikacija SOAP-a koristi HTTP protokol kao primer koji demonstrira vezivanje protokola. Protokol SOAP oformljuje temeljni sloj za stek protokola web servisa, obezbeđujući osnovni okvir za poruke na koju se može nadograditi više apstraktnih nivoa. SOAP koristi obrazac servis-orjentisanog arhitektonskog obrasca SOA, o kojima je bilo reči ranije.

U SOAP-u postoji nekoliko različitih tipova obrazaca za poruke i do sada se najčešće koristi obrazac udaljenog poziva procedura (eng. Remote Procedure Call - RPC), gde jedan čvor mreže (klijent) šalje poruku sa zahtevom drugom čvoru (serveru) i da server odmah šalje odgovor klijentu.

Kao što je već istaknuto, HTTP je izabran kao primarni protokol aplikativnog nivoa za SOAP, zato što je to jako pogodno za današnju Internet infrastrukturu. SOAP naročito dobro funkcioniše kod mrežnih „požarnih zidova“ i to mu je velika prednost u odnosu na druge distribuisane protokole kao što su GIOP/IIOP ili DCOM.

XML je izabran za standardni format poruka, zato što je široko prihvaćen od velikih kompanija i od proizvođača koji kreiraju otvoren izvorni kod. Ponekad je opširna XML sintaksa prednost, a ponekad je ograničenje. Format poruka je jasno struktuiran i lak za ljudsku manipulaciju, ali može usporiti vreme procesiranja.

S obzirom na opširnost XML formata, SOAP je osetno sporiji od drugih tehnologija za srednji sloj (kao što je CORBA) koje transportuju podatke u binarnom obliku.

Da bi se uklonilo to usko grlo, mogu se koristiti hardverski uređaji koji i za red veličina ubrzavaju procesiranje XML poruka.

SOAP specificira standardni format, čiji deo predstavljaju podaci koji su enkodirani u XML dokument. Taj XML dokument se sastoji od taga-korena `Envelope`, koji dalje mora sadržati `Body` elemenat i eventualno `Header` elemenat. SOAP poruke koriste XML prostore imena.

Neobavezno zaglavlje, tj. elemenat `Header`, sadrži relevantne informacije o poruci (kao što su datum slanja poruke, informacija potrebna za autentifikaciju itd.).

Primer. Ovako bi klijent mogao formatirati SOAP poruku kojom zahteva informacije o proizvodu od web servisa nekog stovarišta (pretpostavimo da traži informacije o proizvodu čiji je kod 123456):

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>123456</productID>
     </getProductDetails>
   </soap:Body>
 </soap:Envelope>
```

Web servis skladišta bi, kao odgovor na primljeni zahtev od klijenta, mogao poslati poruku sledećeg oblika:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Komplet od tri kofera </productName>
         <productID>123456</productID>
         <description>Komplet od tri kofera za putovanja. Crna koža.</description>
         <price>96.50</price>
         <inStock>true</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```

WSDL tj.jezik za opisivanje web servisa se, kako mu i samo ime kaže, koristi za opisivanje strukture svih poruka koje podržava servis. I WSDL je XML – on koristi XSD (eng. XML Schema Definition) standard za dokumentovanje unutrašnje strukture svake poruke i svakog ograničenja koje se odnosi na neki od elemenata poruke (videti [W3C]).

WSDL grupiše poruke u operacije. Operacija je logička jedinica interakcije sa servisom, koja je definisana kao poruka-zahtev i sve pridružene poruke-odgovori. Na kraju, WSDL povezuje operacije sa jednim ili više protokola, i grupiše te povezane operacije u servis. Pored specificiranja operacija koje se povezane sa servisom, WSDL specifikacija i dokumentuje komunikacioni kanal koji konkretno adresira servis. Tako, na primer, WSDL specifikacija dokumentuje URL da bi identifikovala servis koji je izložen preko protokola SOAP nad HTTP-om.

Otkrivanje servisa

Kao što je već istaknuto na početku rada, web servisi su distribuisane serverske komponente koje liče na web stranice. Oni predstavljaju moćan metod za pružanje servisa kojima se (preko Interneta) može pristupiti programskim putem. Ove komponente, za razliku od web prezentacija, nemaju UI, već su dizajnirane tako da budu korišćene od strane drugih aplikacija, kao što su tradicionalne klijentske aplikacije, web-bazirane aplikacije, ili čak drugi web servisi (videti [Chung03], [Peltz03]).

Web servisi mogu biti korišćeni interno (od strane jedne aplikacije) ili izloženi eksterno preko Interneta za korišćenje od strane većeg broja aplikacija. S obzirom na to da su dostupni preko standardnih interfejsa, web servisi dopuštaju da heterogeni sistemi rade zajedno kao jedna mreža izračunavanja.

Krajnji deo web servis slagalice je otkrivanje servisa. Kako aplikacija koja koristi servise pronalazi te servise sa kojima treba da sarađuje? Odgovor leži u razdeljenom skladištu koje sadrži opise servisa i pridružuje te opise različitim metapodacima koji su korisni za identifikaciju konkretnog servisa. Na primer, skladište servisa može da vrati pokazivače na servise na osnovu više različitih kriterijuma kao što su: organizacija koja je razvila servis, organizacija kod koje je postavljen servis, vrsta industrije, poslovni proces koji servis podržava itd.

Korišćenje skladišta servisa značajno umanjuje povezanost između subjekta koja obezbeđuje servis i klijenta tog servisa. Do umanjenja dolazi zato što klijent samo treba da referiše na servis, a ne da postavlja u kodu sve detalje koji su potrebni da bi se pristupilo servisu. Ovo dalje dopušta da subjekt koji obezbeđuje servis promeni veći broj delova u komunikacionom ugovoru a da kod klijenta servisa ništa ne mora da se menja – jedino što se, po promeni servisa, dešava je ažuriranje skladišta podataka o servisima. Klijenti koji koriste podatke iz skladišta će pri sledećem pokušaju pristupa automatski očitati nove podatke.

Specifikacija UDDI (eng. Universal Discovery, Description and Integration) rešava probleme otkrivanja servisa za web servise. S obzirom da je interoperabilnost jedan od primarnih ciljeva UDDI, nije čudo što UDDI koristi mnoge tehnologije koje su prethodno već razmatrane. UDDI je prosto skladište koje sadrži veze prema WSDL opisima servisa. UDDI definiše i nekoliko XML opisa raznovrsnih metapodataka koji mogu biti pridruženi servisu (opisi obuhvataju informacije o organizaciji koja je razvila servis, o organizaciji kod koje je postavljen servis, o vrsti industrije, o poslovnom procesu koji servis podržava itd.). Na kraju, UDDI izlaže svoje funkcionalnosti kao skup SOAP servisa.

Apstrakcija i asinhroni rad servisa

Prethodno je istaknuta jedna od osnovnih karakteristika web servisa - visok nivo apstrakcije između implementacije servisa i korišćenja servisa. Korišćenjem XML-zasnovanih poruka kao mehanizma za

kreiranje i pristup servisu, klijent web servisa i sam web servis su oslobođeni potrebe da znaju bilo šta o onom drugom, osim ulaza, izlaza i lokacije (adrese).
Web servisi podržavaju i sinhronu i asinhronu komunikaciju između klijenta i servera koji je domaćin web servisu. Kod sinhrone komunikacije, klijent pošalje zahtev za servis i čeka na odgovor. Kod asinhrone komunikacije klijent po slanju zahteva nastavlja sa radom tj. obradom, a rezultate koje kasnije dobije od servisa obrađuje tek onda kada mu budu na raspolaganju. Korišćenje asinhrone komunikacije poboljšava iskorišćenost sistema i izbegava klijentovo kašnjenje zbog čekanja na odgovor web servisa. Uspešno korišćenje asinhrone komunikacije je čvrsto povezano sa višenitnim programiranjem.

4.2.9. Arhitektura sistema

Web servis za Evolutivni algoritam, nazvan `EaWebService` je razvijen na programskom jeziku C#, u Microsoft .NET okviru. On demonstrira prednosti distribuisanog evolutivnog programiranja na Internet-u. Nadalje, ovakav pristup demonstrira klijent-server obradu podataka, kao i web servise, tj. korišćenje XML-a i SOAP-a u Internet izračunavanjima.

Funkcionalnost EA sistema

Sistem je dizajniran i implementiran tako da istovremeno podržava različite tipove EA (sa potpuno različitim vrstama reprezentacije, ukrštanja, selekcije, mutacije, politike zamene jedinki, različitim mrežnim topologijama, itd.). U ovom pristupu je korišćena objektno-orjentisana paradigma, osobine, interfejsi i indekseri, pa uvođenje i implementacija novih modifikacija EA zahteva minimalnu promenu koda (opisano u [Filipo03a]). Ovakvo rešenje dopušta unificiran pogled na performanse i ponašanje različitih EA tehnika i, u isto vreme, postiže efikasno izvršavanje EA.
EA sistem može biti korišćen u svim scenarijima koji uključuju optimizaciju pomoću EA. U isto vreme, servis može biti korišćen i na način kao kod Evolutivnih strategija ili Genetskog programiranja.
EaWebService obezbeđuje:

- višenitnu i višeprocesnu evoluciju;
- stacionarno i generacijsko izvršavanje EA (uz mogućnost kreiranja alterantivne sheme zamene jedinki u populaciji);
- rad sa više potpopulacija i razmenu jedinki u među potpopulacijama;
- automatski definisane funkcije i makroe;
- logovanje (videti [Keogh03]) i čitanje populacije iz datoteke;
- veoma podesive jedinke koje mogu imati raznovrsnu strukturnu reprezentaciju;
- nekoliko problemskih domena koji su već pripremljeni za eksperimente (optimizacija De Jongovih funkcija, optimizacija proizvoljne funkcije, rešavanje obmanjivačke „zamka“ funkcije, rešavanje sistema 2X2 linearnih jednačina);
- apstrakciju za mnogobrojne EA pristupe, a ne samo za GA.

Celokupni EA sistem je dizajniran kao višeslojna aplikacija sa SOA arhitekturom.
U ovom EA sistemu, istraživač (čak i kad nije ekspert za EA) može eksperimentisati (podešavanjem vrednosti XML elemenata) radi dobijanja najboljih parametara za rešavanje konkretnog problema, ili lako preći sa jednog problemskog domena na drugi. Aplikacija koja koristi servis samo treba da oformi XML niske koje su parametri EA i da prihvati i prikaže rezultat (koji je takođe XML niska).

EaWebService i C#

Ceo sistem, pa i servis EaWebService je dizajniran kako bi se omogućilo da mnoge raznovrsne i složene EA eksperimente kroz lako proširiv dizajn i bogat skup karakteristika. Najpopularniji EA softver je ili mali i orjentisan prema specijalnim namenama, ili ga je teško proširivati.
Zbog svojih prethodno opisanih karakteristika, kao i zbog prenosivosti (preciznije upravljivosti) izvršnog koda i zbog dobre podrške višenitnom i distribuisanom programiranju C# (slično kao Java) postaje sve popularniji u oblasti EA.
Primarni kriterijum dizajna za EaWebService je proširivost, prenosivost i bogat skupa karakteristika, ali zato (opet slično kao što je slučaj u Javi) treba uložiti dodatni napor pri razvoju da bi se očuvala brzina algoritma.

Korišćenjem refleksije kod C# i mogućnosti da se odgovarajuća klasa učita tokom izvršavanja (što je i opisano pri opisu obrasca Strategija), ovaj softverski EA sistem tokom izvršavanja određuje skoro sve (koje vrste objekata će biti napravljene, sa kojim parametrima kreiranja itd.) na osnovu datoteka sa parametrima koje je obezbedio korisnik. Datoteka sa parametrima su, naravno, u XML formatu i oni govore o tome koji se problem rešava, koja se varijacija EA koristi za rešavanje, kako se izveštava o rezultatima itd. S obzirom da se klase instanciraju u vremenu izvršavanja, postoji veoma malo zavisnosti mađu objektima, što je (kako je i opisano u poglavlju o obrascima dizajna) cilj kome se generalno teži prilikom dizajniranja softverskog sistema – na taj način jedna od klasa može biti zamenjena nezavisno od drugih. Samim tim, EaWebService se može lako održavanti i proširivati (videti [Filipo03a]).
C# je, kako je ranije istaknuto, standardizovan i dizajniran tako da bude prenosiv na raznovrsne platforme. Pored toga, C# je standardizovan (standardom su propisani tipovi podataka, operacije itd.) što garantuje da će rezultati izvršavanja biti identični bez obzira na platformu, osim za greške u Java implementaciji. Za naučne eksperimente je izuzetno značajno da se garantuje ponovljivost, a to EA softverski sistemi napisani u jezicima C, C++ i Lisp ne mogu da garantuju.
S obzirom da su upravljani i da se izvršavaju u CLR-u, programi u jeziku C# imaju tendenciju da budu spori. Da bi se blokirala ta tendencija, EaWebService u potpunosti koristi prednosti višenitnog okruženja izbegavajući situacije koje zahtevaju korišćenje C# `Mutex` klase za sinhronizaciju, retko koristi pomoćne objekte koji su čvrsto sinhronizovani. nadalje, EA sistem, kad god je moguće, izbegava korišćenje sporog mehanizma za izuzetke kod C#.
Jezik C#, kao što smo videli, koristi sistem za prikupljanje otpadaka i obezbeđuje automatsko uklanjanje nekorišćenih objekata i proveru dužine niza u vreme izvršavanja. EA tokom izvršavanja generiše veliki broj kratkotrajnih objekata i nizova i nije lako voditi računa o pokazivačima, dealociranju, izlasku indeksa van opsega itd. Korisnik EA sistema, s obzirom da je sistem pisan u jeziku C#, ne mora da brine o tim stvarima.

EaWebService i XML

Pri izvršavanju EA, aplikacija tj. klijent šalje Web servisu XML nisku sa parametrima EA koji će biti izvršen. Neki od ovih parametara (kao što su zamena populacije, selekcija, kriterijum završetka, generator slučajnih brojeva, izveštaji) ne zavise od reprezentacije jedinke dok neki drugi (npr. tip populacije, inicijalizacija, instanca, ukrštanje i mutacija) zavise. Svaki parametar je XML čvor sa elementima koji određuju vrednost parametara. `EaWebService` podržava tri tipa reprezentacije jedinki (kao binarna niska, kao broj u pokretnom zarezu i drvoidna reprezentacija). U okviru parametara EA zadaje se i problem (u XML reprezentaciji) koji se rešava (npr. zapis funkcije je predstavljen u XML formatu, pa isti kod može optimizovati različite funkcije), inicijalni podaci (kada je to potrebno), te veze među Web servisima iste vrste koji zajednički rade (paralelno) kako bi se odredilo rešenje.
Web servis vraća (u XML formatu) nađeno rešenje i neophodne dodatne podatke. Web servis dopušta aplikaciji da prekine tekuće izvršavanje i vrati dotadašnji rezultat u ma kom momentu izvršavanja. Ovaj servis takođe (korišćenjem refleksije) dopušta aplikaciji da dobije informacije o klasama koje predstavljaju pojedine delove EA i da, na taj način, omogući lakše i brže formiranje EA.
Validacija XML-ova koji se koriste tokom rada servisa `EaWebService` je realizovana korišćenjem XML shema, tj. XSD-a

EaWebService objekti, slojevi i obrasci dizajna

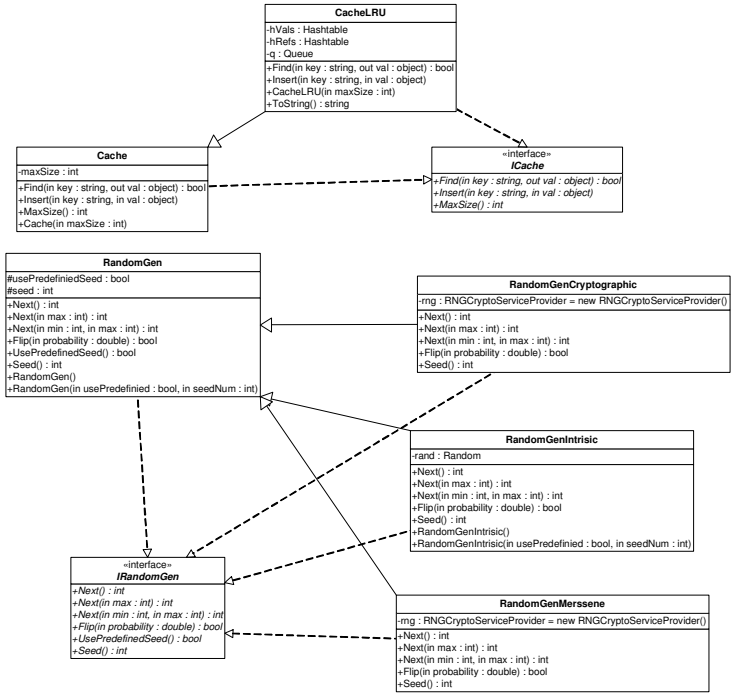
Sam servis EaWebService sadrži 189 klasa i interfejsa: u u prostoru imena `VladoFilipovic.Misc` 15, u prostoru imena `VladoFilipovic.EA.XML` 2, u prostoru imena `VladoFilipovic.EA.Common` 9, u prostoru imena `VladoFilipovic.EA.Deme` 109 i u prostoru imena `VladoFilipovic.EA.Dispatch` 54.
Dizajn servisa EaWebService i aplikacija koje ga koriste, intenzivno koristi obrasce dizajna. Tako su, unutar prostora imena `VladoFilipovic.EA.Deme` korišćeni sledeći obrasci dizajna:

- Obrazac Fasada je korišćen kod klasa `EaDemeFacade` i `DispachFacade`.
- Obrazac Strategija je korišćena pri dizajnu `EaDeme` objekta (trebalo je postići različita izvršavanja algoritma), pri dizajnu `CrossoverExecution` objekta kao i ogromnog broja drugih objekata sa sličnom funkcijom koji su služili za enkapsuliranje mutacije, selekcije, određivanja prilagođenosti itd.

- Obrazac Posmatrač je korišćen za koordinaciju operacija (npr. zahtev za prekidom koji je stigao treba da prekine izvršavanje evolutivnog algoritma). Kao što već ranije istaknuto, u .NET okviru se obrazac Posmatrač realizuje preko događaja i delegata. Događaji koji su kreirani u okviru EaWebService sistema su:
  - `StartExecutionOnDeme` – događaj koji predstavlja početak izvršenja EA na jednom procesorskom elementu. Klasa `StartExecutionOnDeme` izlaže tip događaja `StartExecutionOnDemeEvent` i obezbeđuje inicijalnu obradu ovog događaja. Ona takođe obezbeđuje mehanizam kojim se druge klase pretplaćuju na izloženi događaj.
  - `StartExecution` – događaj koji predstavlja početak izvršenja EA na jednom procesorskom elementu i svim susednim povezanim procesorskim elementima.
  - `StopExecutionOnDeme` – događaj koji signalizira završetak izvršavanja EA na jednom procesorskom elementu. On može nastupiti bilo postavljanjem signala iz spoljašnosti, bilo kada se uslov konvergencije ispuni.
  - `StopExecution` – događaj koji signalizira završetak izvršavanja EA na jednom procesorskom elementu i svim njegovim neposrednim susedima.
  - `TerminateExecutionOnDeme` – događaj koji se ispaljuje kada je okončano izvršavanje na jednom procesorskom elementu i kada više niko ne referiše na objekte koji su korišćeni tokom izvršavanja.
  - `TerminateExecution` – ispaljuje se kada je okončano izvršavanje na procesorskom elementu i svim njegovim neposrednim susedima.
  - `GetStatusOnDeme` – dešava se kada je procesorskom elementu stigao zahtev da se očita status EA koji se izvršavao (a možda se i dalje izvršava) na njemu.
  - `GetStatus` – biva ispaljen po prispeću zahteva za očitavanje statusa konkretnog EA na čvoru primaocu zahteva i na svim čvorovima koji se nalaze u njegovom neposrednom susedstvu.
  - `GetResultsOnDeme` – dešava se kada je procesorskom elementu stigao zahtev da se pročitaju trenutni rezultati rada EA koji se izvršavao (a možda se i dalje izvršava) na njemu.
  - `GetResults` – podiže sa kada prispe zahtev za očitavanje rezultata na čvoru-primaocu zahteva i na svim njegovim neposrednim susedima.
- Obrazac Filter za presretanje je korišćen pri dizajnu objekata koji proveravaju da li su ispunjeni uslovi za završetak izvršavanja algoritma (koji mogu biti različite kombinacije jednostavnijih uslova), kao i pri generisanju izveštaja o izvršavanju EA, bilo na jednom procesorskom elementu, bilo u celoj populaciji.
- Obrazac Apstraktna fabrika je u skoro svim slučajevima korišćen u kombinaciji sa obrascem Strategija – svi objekti potrebni za izvršavanje EA su kreirani korišćenjem ovog obrasca.

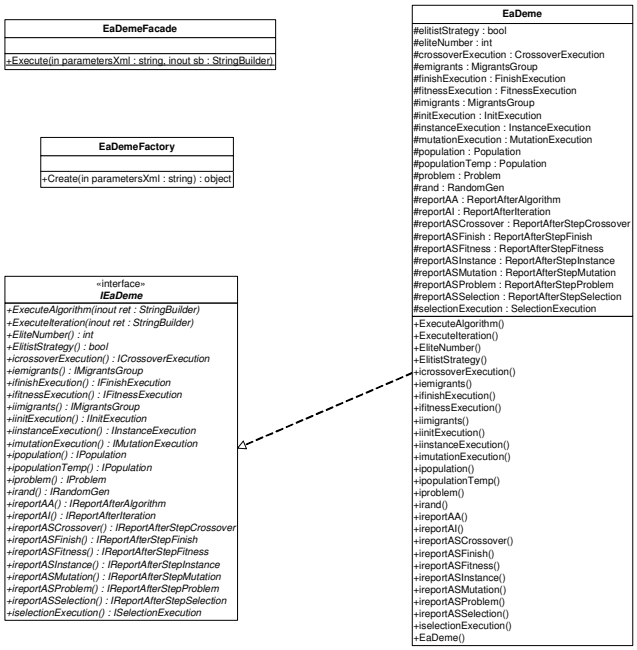
Lakoća nadogradnje Web servisa sledi iz činjenice da je servis čvrsto oslonjen na interfejse. Tako, kad god se pristupa metodu ili osobini neke klase van te klase, to se realizuje uzimanjem interfejsa i pristupom metodu odnosno osobini interfejsa (a ne direktno klase).

Prostor imena `VladoFilipovic.EA.Common` sadrži, kao što mu i ime kaže, metode koje su zajedničke u različitim segmentima EA – konkretno metode za generisanja pseudoslučajnih brojeva na različite načine i metode za keširanje.



slika 4.18. Najvažnije klase u prostoru imena VladoFilipovic.EA.Common

Prostor imena u `VladoFilipovic.EA.Deme` sadrži klase pomoću kojih se realizuje evolutivni algoritam na jednoj potpopulaciji.



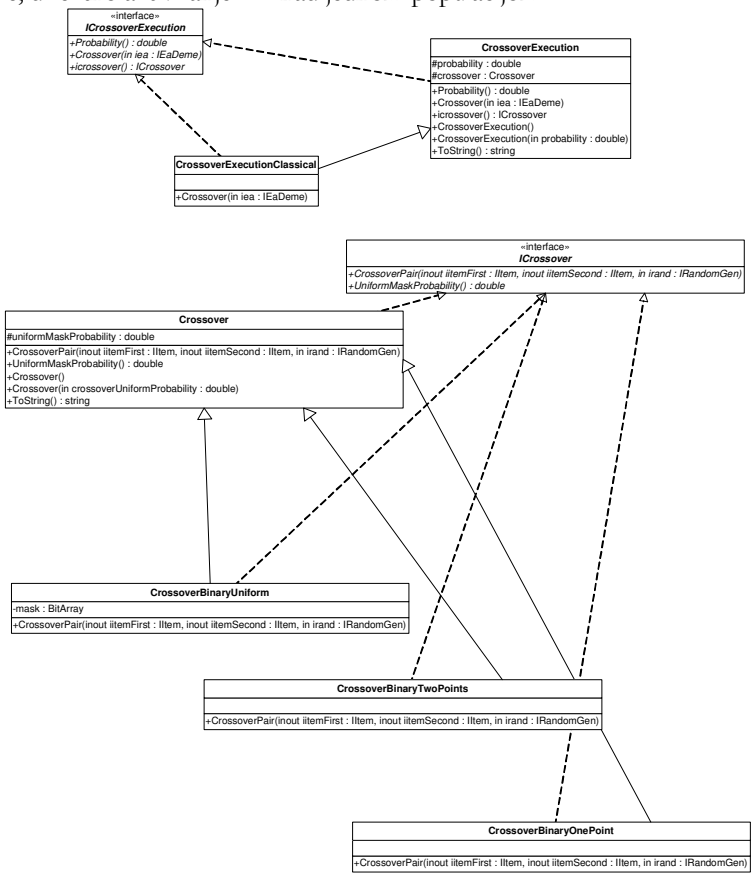
slika 4.19. Centralne klase u prostoru imena VladoFilipovic.EA.Deme

Centralne klase (i interfejsi) u realizaciji EA kod jedne populacije obezbeđuju metode za:

- raznovrsna ukrštanja, mutacije i selekcije;
- razne vrste populacija i tipove jedinki;
- različite vrste inicijalizacija jedinki (koje zavise od tipa jedinke);

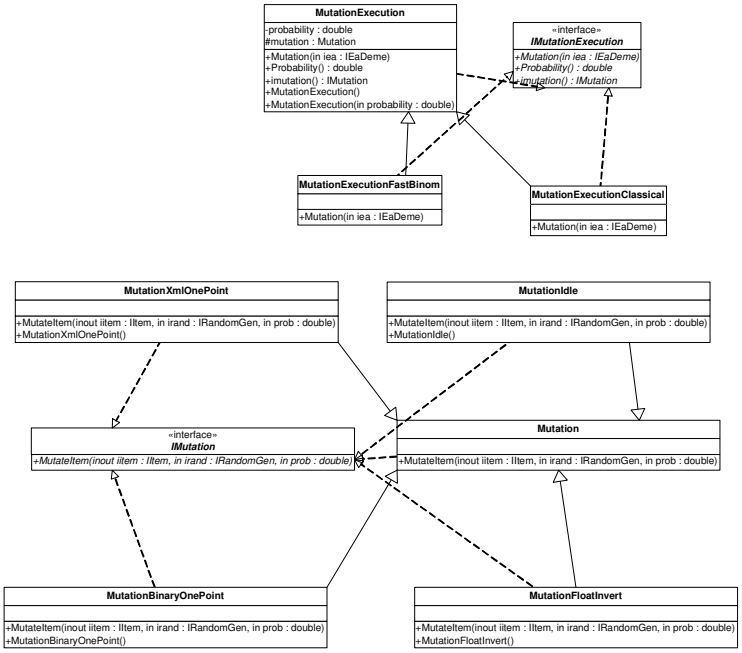
- različite tipove problema i instance problema (koje zavise od tipa problema);
- različite metode odrešivanja prilagođenosti (po određivanju funkcije objekcije);
- različite kombinacije kriterijuma za završetak algoritma;
- raznovrsne izveštaje u raznim fazama izvršavanja EA, a postoji i mogućnost zapisa u log.

Naravno, različite implementacije interfejsa IEaDeme omogućuju (prevazilaženjem metoda `ExecuteIteration` i `ExecuteAlgorithm`) implementaciju raznovrsnih politika zamene jedinki u populaciji i ostale, potencijalno veoma egzotične, modifikacije.
Klasa `EaDemeFacade` se koristi kao fasada, kao i za direktan pristup evolutivnom algoritmu, gde se zaobilazi web servis i za brže, direktno aktiviranje EA nad jednom populacijom.

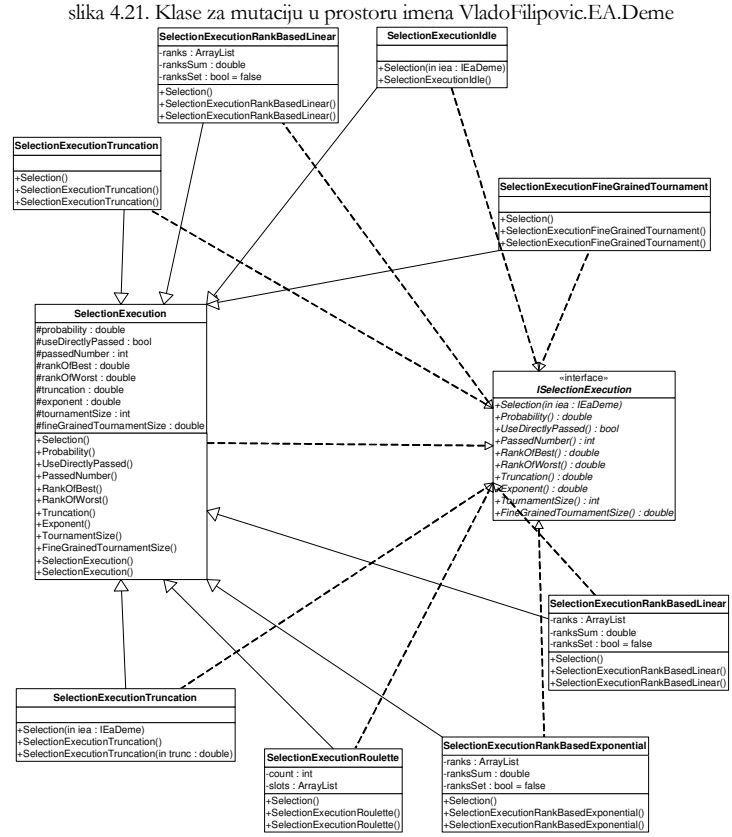


slika 4.20. Neke od klasa za ukrštanje u prostoru imena VladoFilipovic.EA.Deme

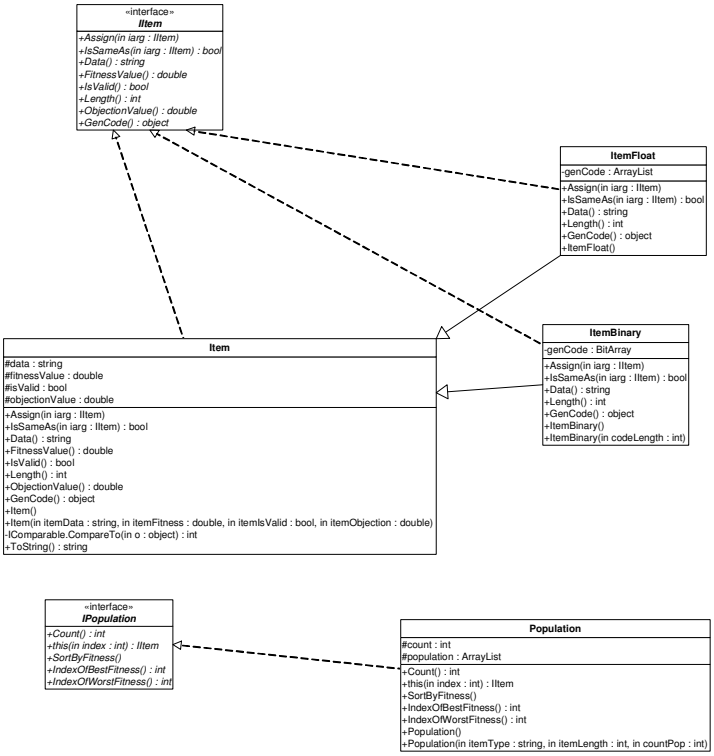
Pri dizajnu klasa za ukrštanje korišćeni su dizajn obrasca Strategija i Apstraktna fabrika, tako da klijent servisa odlučuje koje će se od ponuđenih ukrštanja izvršavati. Sama politika izvršavanja faze ukrštanja, koja je enkapsulirana u klasi `CrossoverExecution` i klasama koje su izvedene iz nje, ne zavisi od vrste reprezentacije jedinke. S druge strane, postupak ukrštanja dve jedinke, realizovan u klasama izvedenim iz klase `Crossover`, nužno zavisi od izabrane reprezentacije. Tako je, da bi se održala fleksibilnost sistema, samo ukrštanje realizovano kao kolaboracija klase koja određuje politiku ukrštanja i koja proziva drugu klasu – a ta druga klasa određuje kako će biti realizovano ukrštanje dve konkretne jedinke.
Analogna logika je primenjena i na dizajn klasa za mutaciju i za selekciju.



slika 4.21. Klase za mutaciju u prostoru imena VladoFilipovic.EA.Deme



slika 4.22. Klase za selekciju prostoru imena VladoFilipovic.EA.Deme

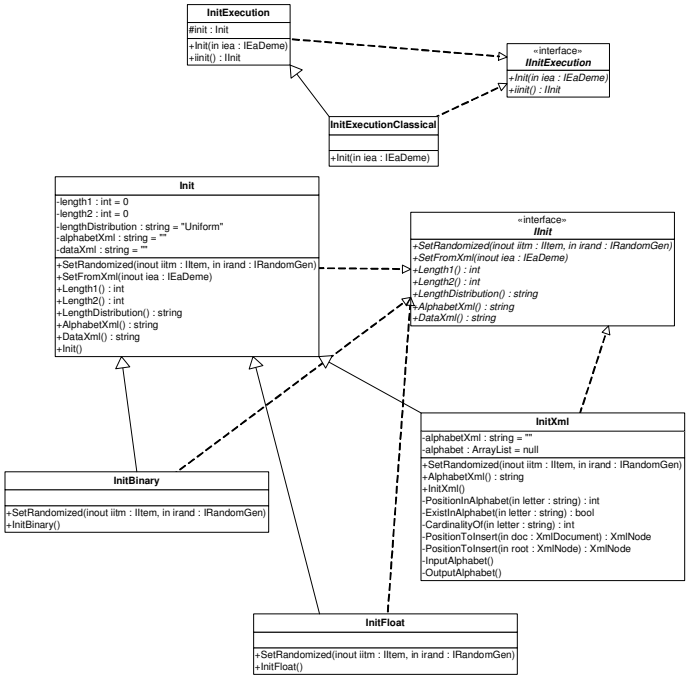


slika 4.23. Klase za jedinke i za populaciju u prostoru imena VladoFilipovic.EA.Deme

U ovoj inkarnaciji EA softverski sistem podržava tri vrste predstavljanja jedinki:

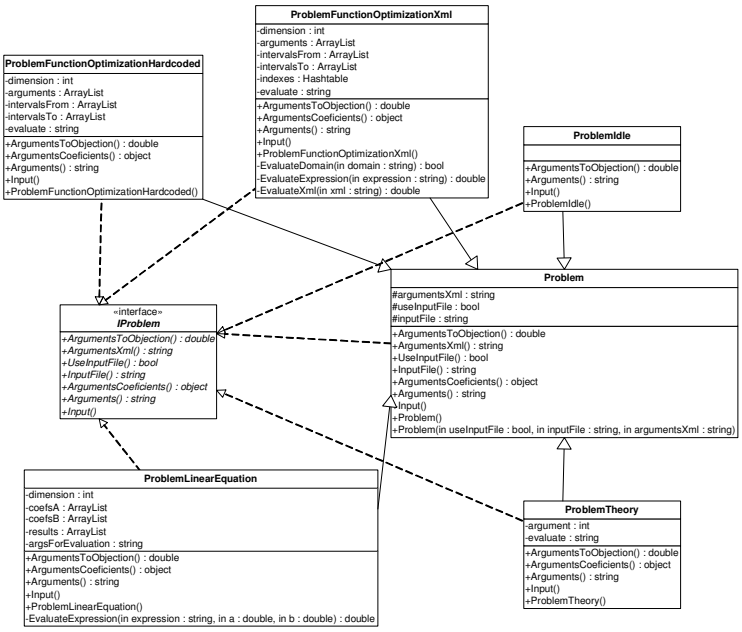
- u obliku bitovne niske (kojoj se dužina prosleđuje kao parametar pri kreiranju jedinke);
- u obliku broja (ili niza brojeva) u pokretnom zarezu;
- u obliku drvoidne strukture (koja je predstavljena XML dokumentom), a koja je veoma pogodna za GP izračunavanja.

Jedinke populacije implementiraju interfejs IComparable, što omogućava njihovo direktno poređenje, a u populaciji je implementiran indekser, što dopušta da se pojedinim objektima-jedinkama u okviru objekta-populacije pristupa kao članovima niza - korišćenjem notacije sa uglastim zagradama [ ] koja je uobičajene u programskim jezicima Pascal, C, C++, Java i C#.

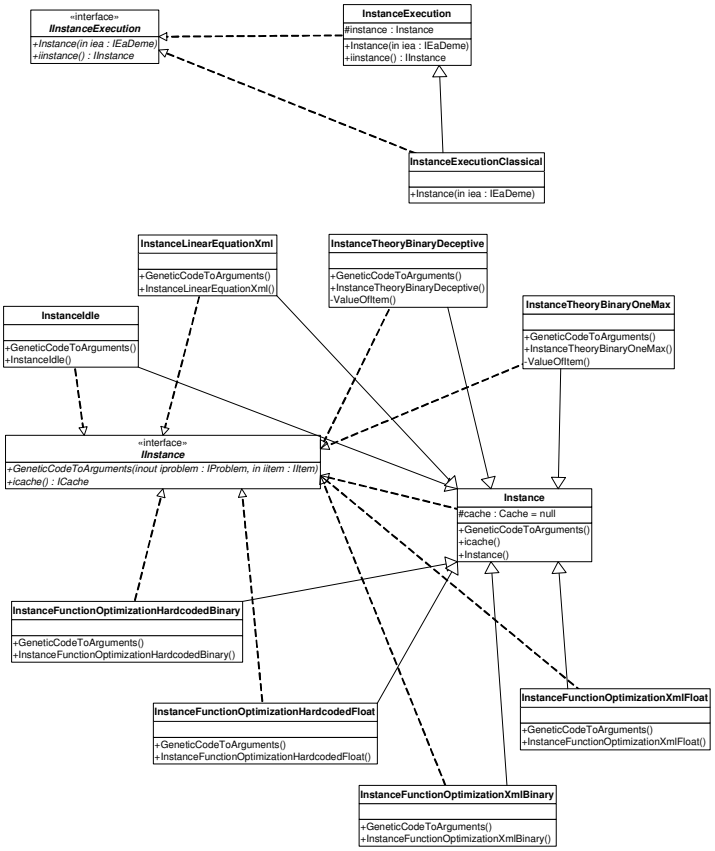


slika 4.24. Klase za inicijalizaciju u prostoru imena VladoFilipovic.EA.Deme

Kod klasa sa prethodnog dijagrama, naravno, postoji zavisnost – sam proces inicijalizacije zavisi tipa jedinke koja se inicijalizuje, a pri tome jedan tip jedinke može biti inicijalizovan na više načina (u ovom trenutku su podržane binarna reprezentacija, reprezentacija sa brojevima u pokretnom zarezu i drvoidna XML reprezentacija..

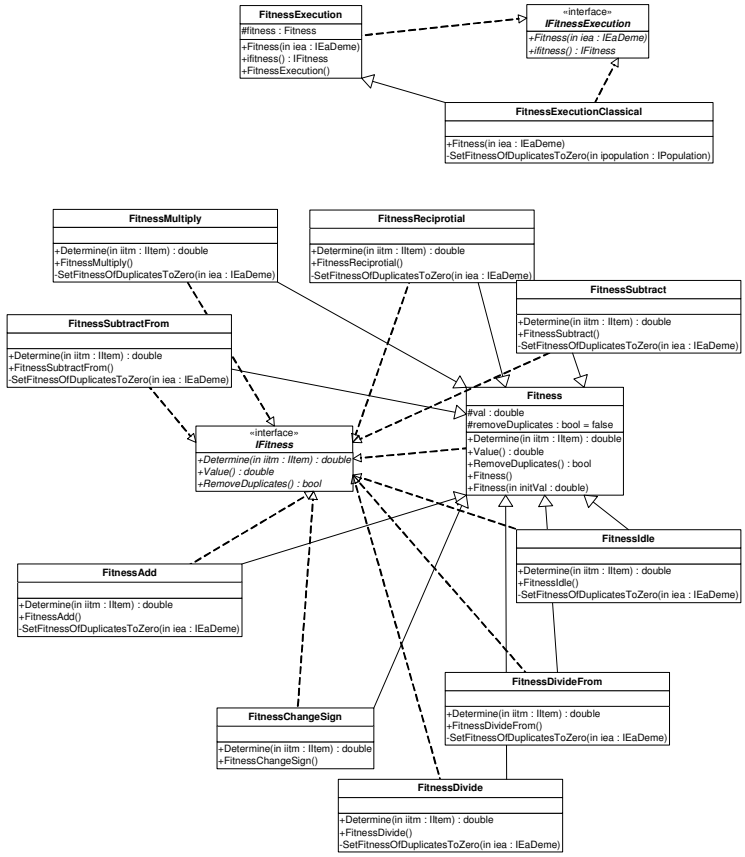


slika 4.25. Klase sa domenski-zavisnim problemima u prostoru imena VladoFilipovic.EA.Deme

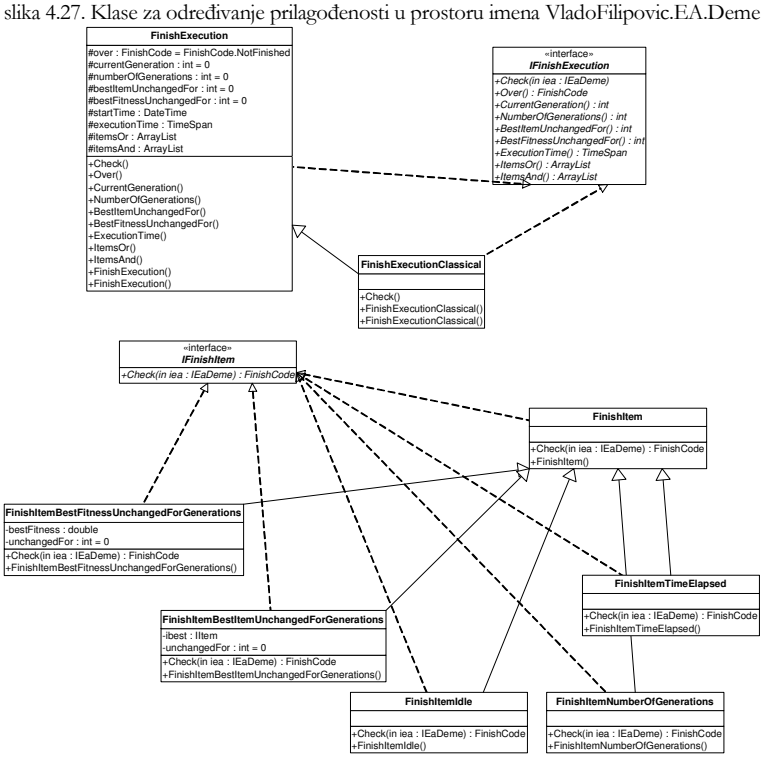


slika 4.26. Klase sa instancama problema u prostoru imena VladoFilipovic.EA.Deme

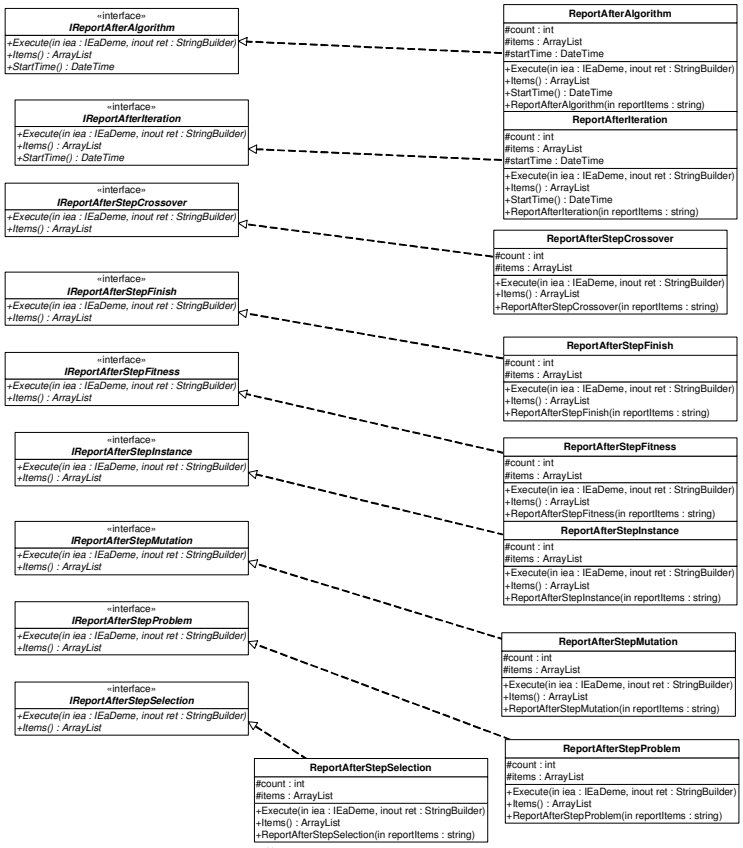
Analizirajući prethodni dijagram može se uočiti da objekti-instance (tj. njihova implementacija) zavisi i od problema koji se rešava i od tipa jedinke.



slika 4.27. Klase za određivanje prilagođenosti u prostoru imena VladoFilipovic.EA.Deme

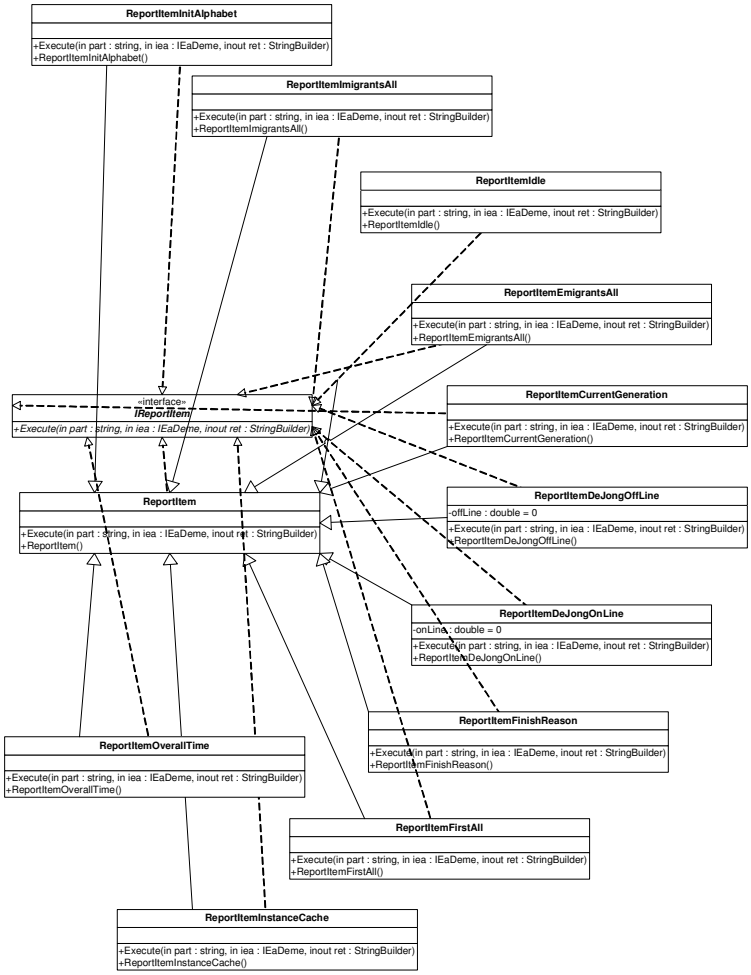


slika 4.28. Klase za proveru završetka izvršavanja algoritma u prostoru imena VladoFilipovic.EA.Deme



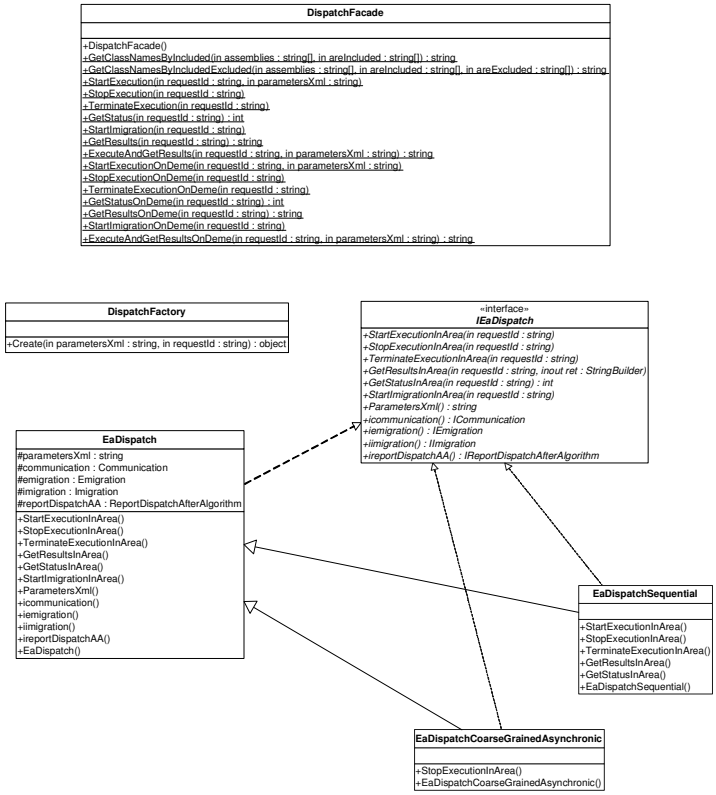
slika 4.29. Klase za izveštaje u prostoru imena VladoFilipovic.EA.Deme

Kod izveštaja su korišćeni obrasci Strategija, Filter za presretanje i apstraktna fabrika. Naime, svaki od izveštaja



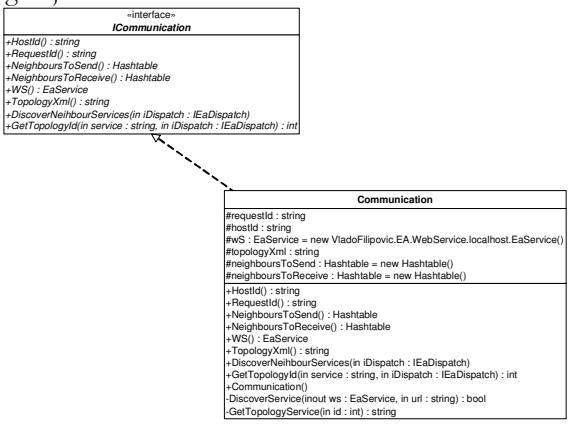
slika 4.30. Klase za elemente izveštaja u prostoru imena VladoFilipovic.EA.Deme

U prostoru imena `VladoFilipovic.EA.Dispatch` se nalaze klase koje su esencijalne za uspešan rad web servisa i kolaboraciju i paralelan web servisa nad potpopulacijama. Klase ovog porostora imena opisane sa dijagramima sa slika 4.31. – 4.33.

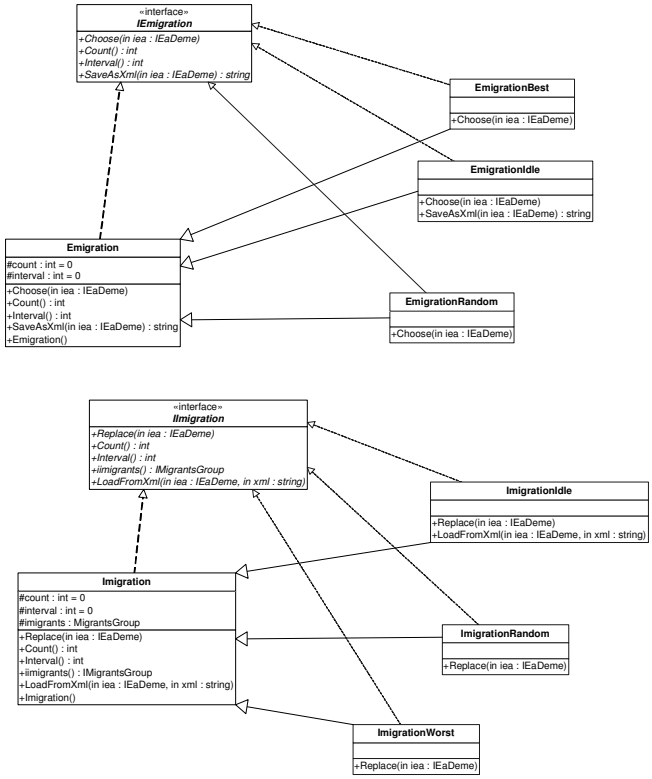


slika 4.31. Centralne klase za paralelni rad u prostoru imena VladoFilipovic.EA.Dispatch

Mehanizam paralelnog izvršavanja EA nad potpopulacijama se suštinski poklapa sa izvršavanjem algoritma PGANP koji je opisan u poglavlju 4.1.2.



slika 4.32. Klase za komunikaciju u prostoru imena VladoFilipovic.EA.Dispatch



slika 4.33. Klase za imigraciju i emigraciju u prostoru imena VladoFilipovic.EA.Dispatch

5. EMPIRIJSKO POREĐENJE EVOLUTIVNIH ALGORITAMA

EA se razvijaju i mnogi autori su predlagali razna poboljšanja, o čemu je detaljnije bilo reči i u poglavlju 1.5. Neke od predloženih izmena su bile skoncentrisane i efikasno primenljive samo na problem kojim se autor-predlagač bavio, dok su drugi predlozi bili opšte primenjivi.
Postoji veliki broj radova u kojima se porede različite modifikacije, tj različite varijante EA. U nekim od njih su upoređivani operatori selekcije kod EA (videti [Goldbe91a], [DeLaMa93], [Hancoc94], [Bäck95], [Filipo96]), u drugim su upoređivani operatori mutacije i nivoi operatora mutacije (videti [Fogart89], [Bäck93], [Hinter95]), u trećim se porede reprezentacije jedinke (radovi [Janiko91], [Schrau92], [Filipo96a]), itd.
U nekim radovima se porede klase EA sa istim genetskim operatorima, ali sa različitim veličinama kontrolnih parametara koji utvrđuju način primene ovih operatora. Cilj takvih radova je utvrđivanje optimalne vrednosti (ili optimalne politike dodele vrednosti) za kontrolne parametre (videti [Grefen86], [Grefen89], [Schaff89], [Sriniv94], [Park95], [Filipo97a]).
Da bi se precizno moglo eksperimentisati sa raznovrsnim modifikacijama i utvrđivati njihov kvalitet, neophodno je da se te modifikacije primene na neke već poznate test-probleme. U tu svrhu se koriste problemi nalaženja ekstremne vrednosti funkcije u zadatoj oblasti. Pogodnost kod takvog izbora je laka mogućnost praćenja toka izvršavanja algoritma, kao i lakoća ispitivanja ponašanja EA pod raznovrsnim uslovima, tj. lakoća ispitivanja robusnosti algoritma. Nadalje, za ovaj problem se relativno lako mere i upoređuju performanse raznovrsnih tipova paralelnih EA.
Među istraživačima iz oblasti evolutivnog programiranja je praksa da se kvalitet neke modifikacije algoritma procenjuje na osnovu rezultata koje tako modifikovani algoritam postiže u rešavanju problema optimizacije funkcije.
Ovakvi empirijski i eksperimentalni prilazi poređenju algoritama imaju mnoge nedostatke, naročito kada se algoritmi dizajniraju tako da budu robusna oruđa za optimizaciju i pretragu. Jedna očigledna opasnost pri empirijskom evaluiranju algoritama pretraživanja je da zaključak više zavisi od problema koji su uzeti za testiranje nego od algoritama koji se porede. Zbog ovoga se može dogoditi da algoritmi daju bolje rezultate na određenom, konkretnom skupu test-funkcija, a da se za druge funkcije ne ponašaju ni izbliza tako kvalitetno.
Dakle, neophodno je razviti metodologiju za empirijsko poređenje raznih EA kojima se rešava određeni problem, kao i za empirijsku proveru da li i gde predloženo poboljšanje zaista daje bolji rezultat.
Realizacija EA koji rešava nalaženje ekstremuma je izvedena tako što je za prostor niski uzet skup binarnih niski fiksirane dužine. Preslikavanje realnog broja iz nekog intervala u nisku bitova (dužine n) se obično obavlja na sledeći način: interval se podeli na ekvidistantne tačke (broj tačaka je 2^n), i broju se ili dodeli binarni kod najbliže donje tačke podele, ili mu se dodeli Grejov kod prethodno određenog binarnog celog broja. Funkcija prilagođenosti se računa na osnovu vrednosti funkcije čiji se ekstremum traži - ako se traži maksimum, tada se funkcija prilagođenosti poklapa sa funkcijom objekcije (inače se vrši skaliranje).
Treba napomenuti da se u razvoju metodologije za empirijsko poređenje optimizacionih algoritama ne uzimaju u obzir problemi kombinatorne optimizacije. Za takve probleme obično od pre postoje dobro poznati test-primeri.

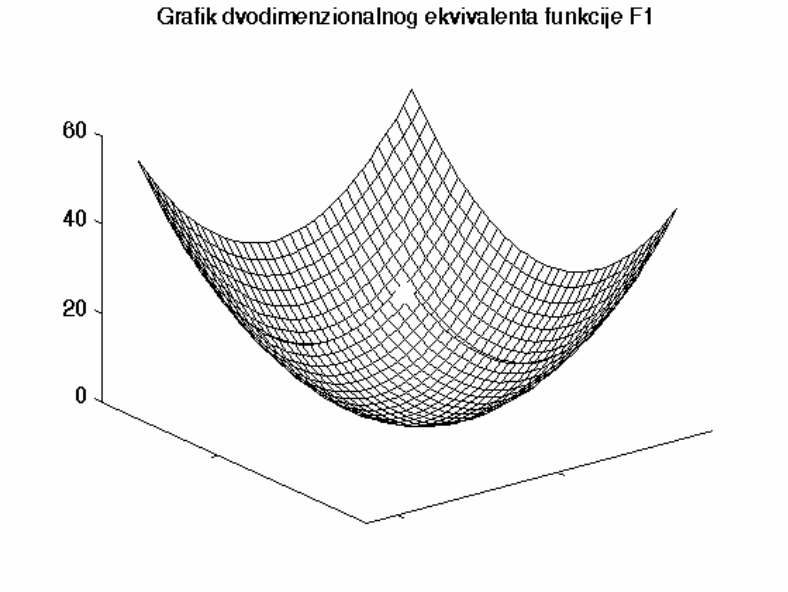
5.1. De Jongove test-funkcije

Već je naglašena potreba da se spreči specijalizacija, tj. prilagođavanje algoritma konkretnom skupu test-funkcija. To se postiže izborom skupa test-funkcija (pored naziva test-funkcija koristiće se i naziv test-problem) tako da on bude istovremeno izazovan i raznovrstan. Ideja je da skup test-problema uključuje funkcije sa sledećim karakteristikama: neprekidne i prekidne; konveksne i nekonveksne; jednomodalne i višemodalne; determinističke i stohastičke; malodimenzionalne i mnogodimenzionalne.
De Jongovo test okruženje (videti [DeJong75]) sadrži sledećih pet funkcija:

1) Sferna funkcija

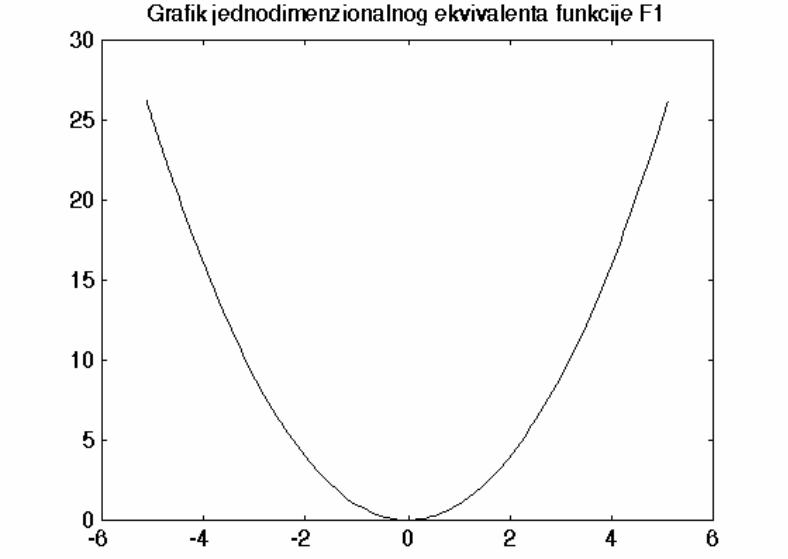
Analitički oblik funkcije i njen domen je dat sa:

$$F1(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2 \quad \text{(pri čemu je } -5.12 \le x_i \le 5.12\text{)}.$$



slika 5.1. - Dvodimenzionalna restrikcija sferne funkcije

Ova funkcija ima karakteristike neprekidnosti, konveksnosti, jednomodalnosti, kvadrata, malodimenzionalnosti i determinističnosti. Dominantna karakteristika ove funkcije je jednomodalnost.



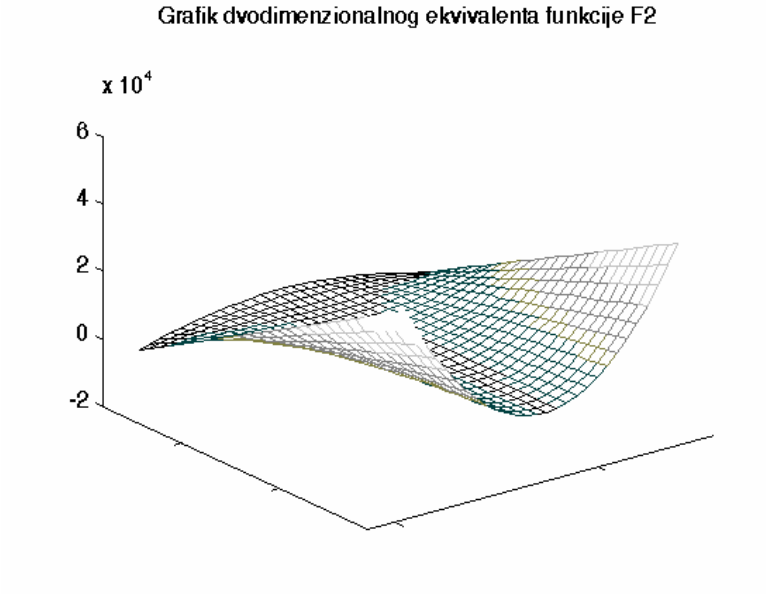
slika 5.2. - Jednodimenzionalna restrikcija sferne funkcije

Sferna funkcija se označava sa F1.

2) Rozenbrockova funkcija:

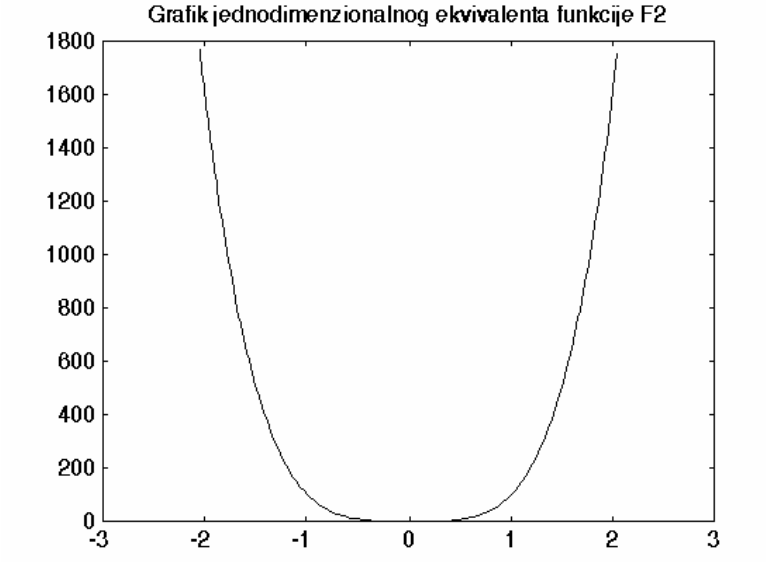
Analitički oblik Rozenbrockove funkcije i njen domen su dati sledećom formulom:

$$F2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \quad \text{(pri čemu je } -2.048 \le x_i \le 2.048\text{)}$$



slika 5.3. – Dvodimenzionalna restrikcija Rozenbrokove funkcije

S obzirom na De Jongove karakteristike, ova funkcija je neprekidna, nekonveksna, višemodalna, nekvadratna, malodimenzionalna i deterministična. Dominantna karakteristika Rozenbrokove funkcije je nelinearnost.



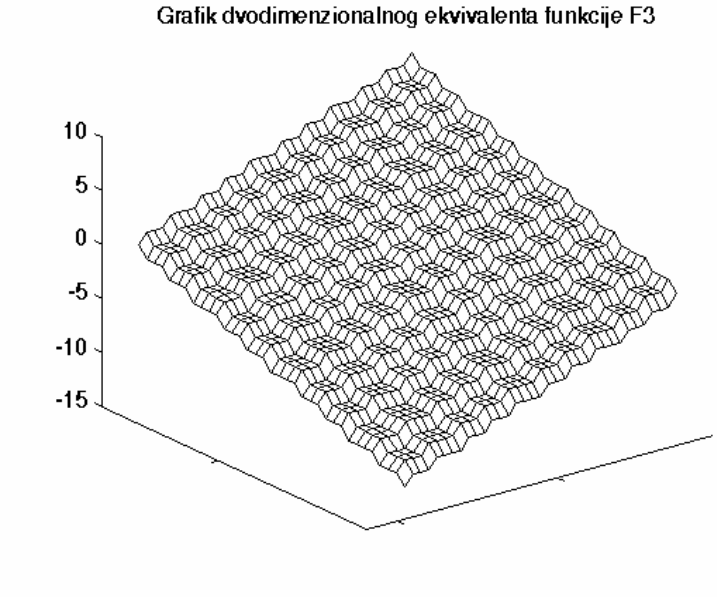
slika 5.4. – Jednodimenzionalna restrikcija Rozenbrokove funkcije

Oznaka ove funkcije je F2.

3) Stepeničkа funkcija

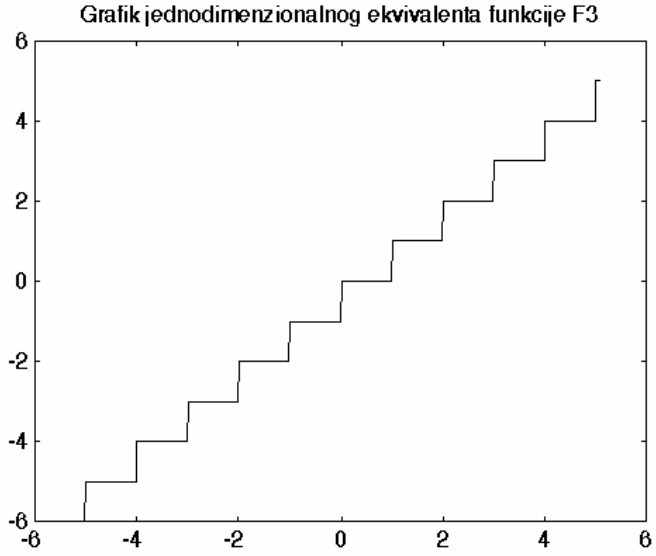
Analitički oblik i domen Stepeničke funkcije je dat sledećom formulom:

$$F3(x_1,\ldots,x_5)=\sum_{i=1}^{5}\lfloor x_i \rfloor$$ (pri čemu je $-5.12 \le x_i \le 5.12$)



slika 5.5.- – Dvodimenzionalna restrikcija Stepeničke funkcije

Dominantna karakteristika, (karakteristika zbog koje je ova funkcija uključena u skup test-funkcija) je njena prekidnost.



slika 5.6. – Jednodimenzionalna restrikcija Stepeničke funkcije

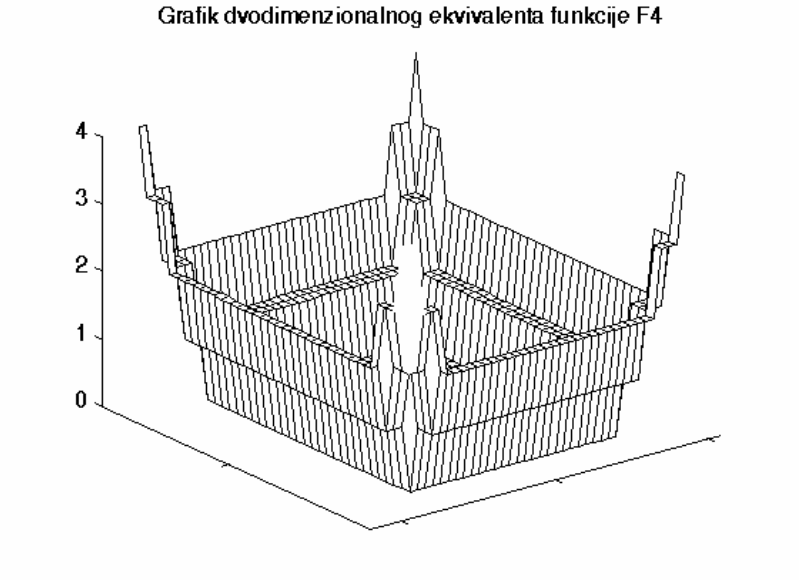
Oznaka ove funkcije je F3.

4) Četvorna funkcija sa šumom

Analitički oblik i domen Četvorne funkcije sa šumom je dat sledećom formulom:

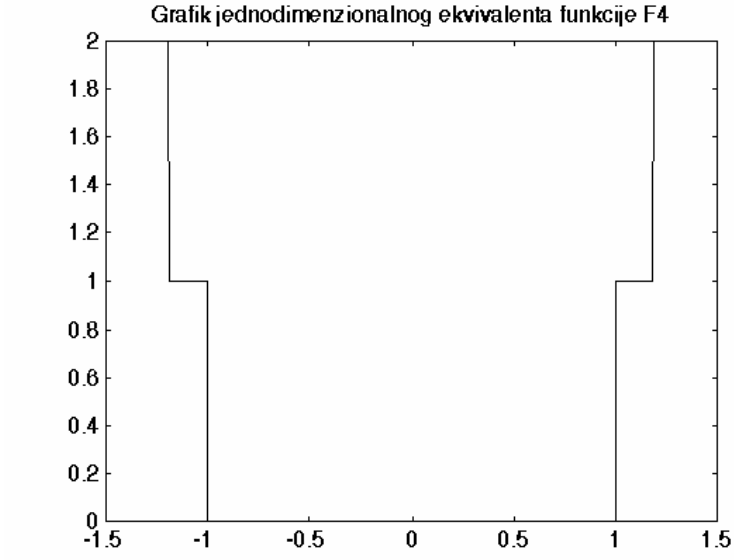
$$F4(x_1,...,x_{30}) = \sum_{i=1}^{30} i x_i^{\,4} + N(0,1)$$ (pri čemu je $-1.28 \le x_i \le 1.28$)

Grafik dvodimenzionalnog ekvivalenta ove funkcije prikazan je na slici 5.7, a jednodimenzionalnog na slici 5.8.



slika 5.7. Četvorna funkcija sa šumom (dvodimenzionalni ekvivalent)

Kao što joj samo ime kaže, jedna od najvažnijih karakteristika ove funkcije je šum. Veličina šuma je slučajna promenljiva koja ima normalnu raspodelu.



slika 5.8. Četvorna funkcija sa šumom (jednodimenzionalni ekvivalent)

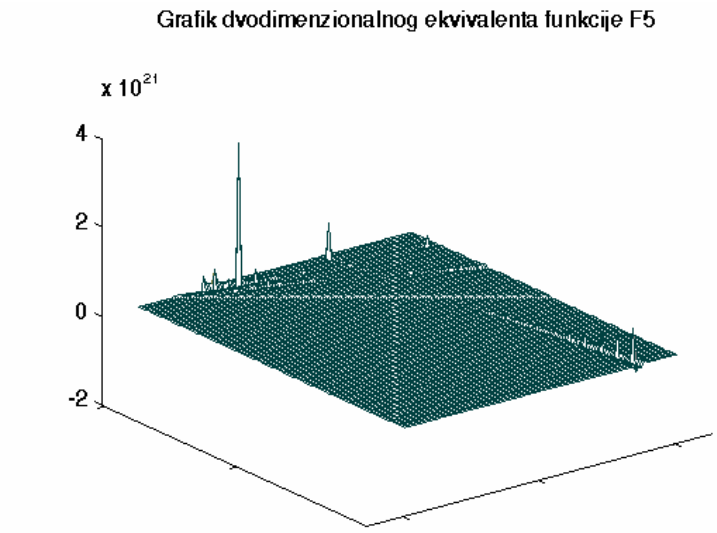
Ova funkcija se označava sa F4.

5) Shekelova funkcija:

Analitički oblik i domen Shekelove funkcije (još se zove i Shekelove lisičije rupe) je dat sledećom formulom:

$$F5(x_1, x_2) = 0.0002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$ (pri čemu je $-65.536 \le x_i \le 65.536$)

Grafik ove funkcije je prikazan na slici 5.9



slika 5.9. Četvorna funkcija sa šumom (dvodimenzionalni ekvivalent)

Ova funkcija je uključena u skup test-funkcija zbog svoje višemodalnosti i činjenice da postoji veći broj lokalnih optimuma. U literaturi se Shekelova funkcija označava sa F5.

5.2. De Jongovi kriterijumi

Za vrednovanje rezultata različitih EA, koriste se tri mere: jedna za merenje konvergencije, druga za merenje performansi toka, a treća za merenje širine pretrage. Prva je nazvana on-line performansa, a druga off-line performansa. Ove mere su dobile naziv po analogiji sa on-line i off-line aplikacijama. Treća mera je dobila naziv „broj izgubljenih gena". Novouvedene performanse se definišu na sledeći način:

- On-line performansa strategije *s* u okruženju *E*: $x_E^*(s) = \frac{1}{T}\sum_{g=1}^{T} f_E^*(g)$.
- On-line performansa se još naziva i konvergencija. Sa $f_E^*(g)$ je označena maksimalna vrednost funkcije prilagođenosti tokom probe *g*, tj. maksimalna vrednost funkcije prilagođenosti u generaciji *g*. On-line performansa je prosek dosad najboljih performansi u raznim generacijama. Predložena je i opštija verzija ovog kriterijuma, koja dopušta neravnomernu raspodelu za težine raznim pokušajima.
- Off-line performansa (tok) strategije *s* u okruženju *E*: $x_E(s) = \frac{1}{T}\sum_{g=1}^{T} f_E(g)$. Sa $f_E(g)$ je označena srednja vrednost funkcije prilagođenosti za okruženje *E* tokom probe *g*, tj. prosečna vrednost funkcije prilagođenosti u generaciji *g*. Off-line performansa je prosek svih funkcijskih evaluacija zaključno sa tekućim pokušajem. I za ovu meru je bila predložena i neuniformna varijanta, ali je češće korišćena uniformna.
- Broj izgubljenih gena tokom probe *t*, u oznaci $lo_E(s)$: razlika 2^l i broja različitih jedinica i nula na svim pozicijama za sve jedinke u populaciji. Performansa nazvana broj izgubljenih gena ukazuje na širinu pretrage kroz pretraživački prostor.

Ispitivane su razne varijacije genetskih algoritama. Reproduktivni plan je familija strategija koje se razlikuju po vrednosti stohastičkih parametara. U svojim radovima ([Grefen86], [Grefen89], [Schaff89]) autori razmatraju odnos nekoliko reproduktivnih planova i eksperimentalno određuju vrednosti stohastičkih parametara za razne varijacije EA. Tokom vremena, produbljivanjem dotadašnjih znanja, ponekad je dolazilo i do promene ovih vrednosti, koje su odslikavale nova saznanja.

5.3. Novije metodologije empirijskih poređenja

Test-problemi se koriste radi poređenja performansi raznih evolutivnih algoritama. Međutim, često se dešava, da test-problemi nisu reprezentativni, odnosno da ne predstavljaju sve važne karakteristike datog problema. Kao što je već istaknuto, postoji opasnost da zbog takve prakse algoritmi mogu biti prilagođeni samo malom skupu test-problema. Ta opasnost dobija na značaju ukoliko test-problemi ne predstavljaju tipove problema za koje se evolutivni algoritmi najviše koriste. To kod ovih test-funkcija i jeste najčešće slučaj - postoje drugi tipovi algoritama koji bi za prethodno pobrojane funkcije brže doveli do rešenja.
Dakle, do uključivanja ovih funkcija u skup test funkcija došlo je zbog toga što funkcije F1-F5 ne odslikavaju u potpunosti tipične probleme koji se rešavaju pomoću genetskih algoritama. Naime, funkcije F1-F5 su uključene u skup test-funkcija da bi se istakla širina klase problema na koje se mogu primeniti EA. Ubrzo se ispostavilo da su ove test-funkcije ili suviše male dimenzije ili suviše jednostavne, pa neki drugi, jednostavniji, algoritmi (npr. metoda gradijentnog spusta, metoda enumerativne pretrage, metoda slučajne pretrage) daju bolje rezultate.
Važno je napomenuti da, pored proširenja skupa De Jongovih test funkcija, (videti [Davis91a], [DeFalc96])u kojima se test funkcije analiziraju sa nešto drugačijeg stanovišta (penjanje po bitovima, predrasude kod reprezentacije, itd.).

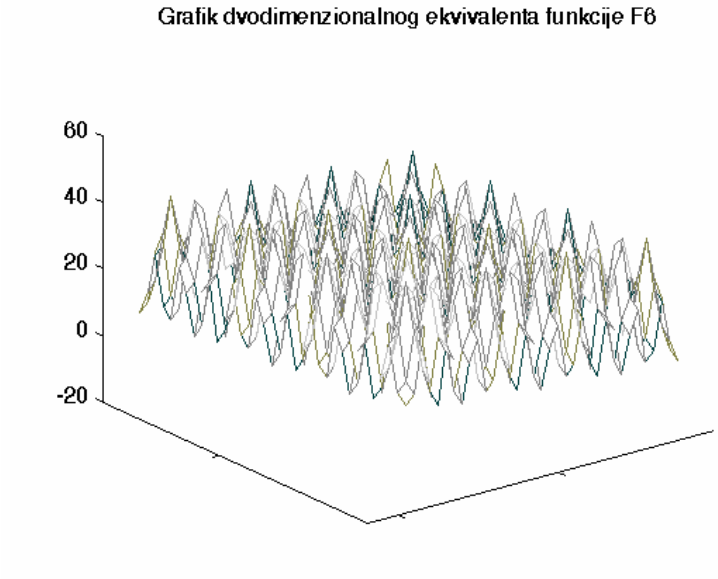
5.3.1. Proširenja De Jongovih funkcija

Stoga su, tokom godina, pored De Jongovih pet funkcija, u skup test-funkcija uključene još neke funkcije (videti [Davis91], [Hancoc94], [Janiko91], [Schaff89]).

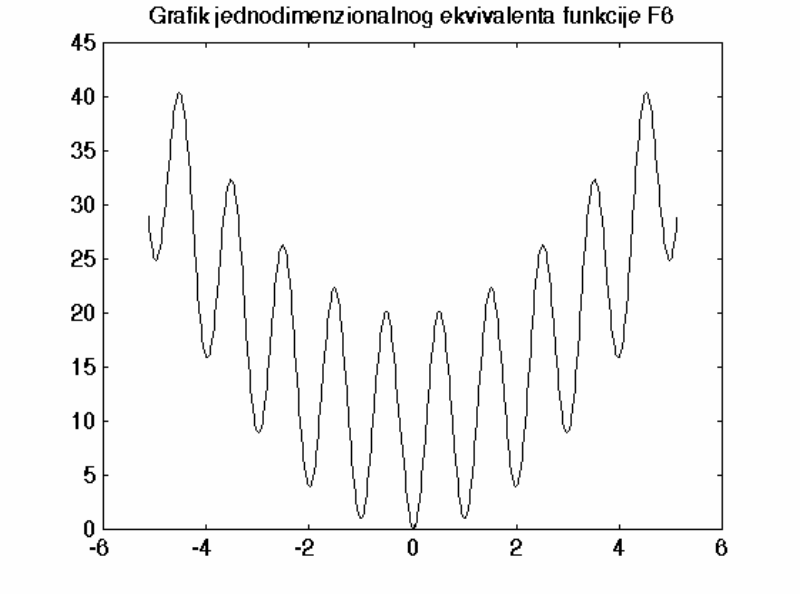
1) Rastriginova funkcija:
Analitički oblik Rastriginove funkcije je dat sledećom formulom:

$$F6(x_1,...,x_N) = 10N + \sum_{i=1}^{N}(x_i^2 - 10\cos 2\pi x_i) \quad \text{(pri čemu je } -5.12 \le x_i \le 5.11\text{)}$$

Grafici dvodimenzionalne i jednodimenzionalne restrikcije ove funkcije prikazani su na slikama 5.10 i 5.11.



slika 5.10. Rastriginova funkcija (dvodimenzionalni ekvivalent)

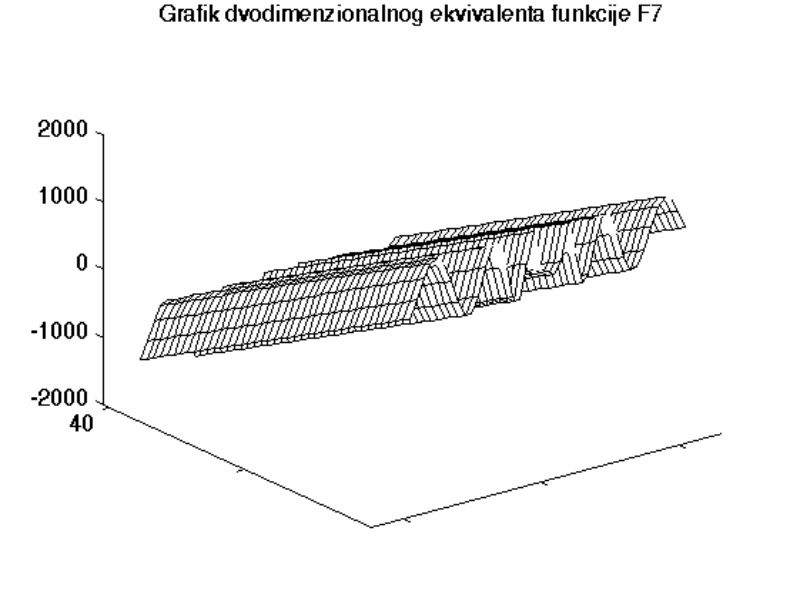


slika 5.11. Rastriginova funkcija (jednodimenzionalni ekvivalent)

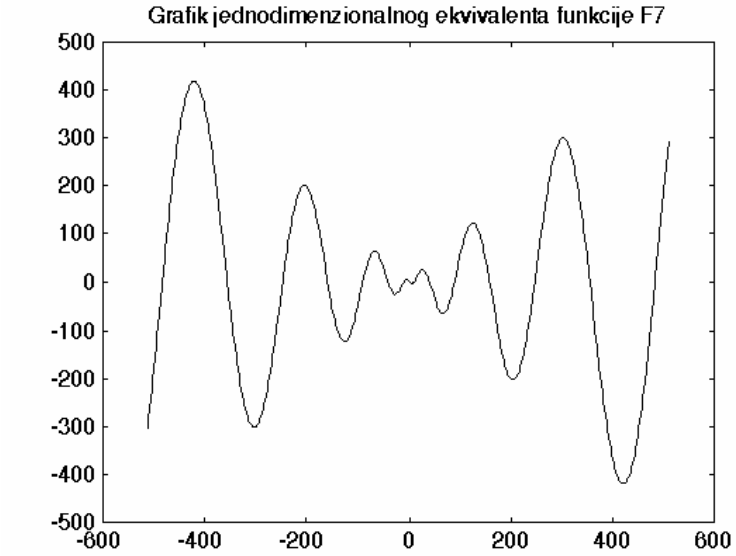
2) Schwefelova funkcija:
Analitički oblik Schwefelove funkcije je dat sledećom formulom:

$$F7(x_1,...,x_N) = \sum_{i=1}^{N} -x_i \sin\sqrt{|x_i|} \quad \text{(pri čemu je } -5.12 \le x_i \le 5.11\text{)}$$

Grafici dvodimenzionalne i jednodimenzionalne restrikcije ove funkcije prikazani su na slikama 5.12 i 5.13.



slika 5.12. Schwefelova funkcija (dvodimenzionalni ekvivalent)



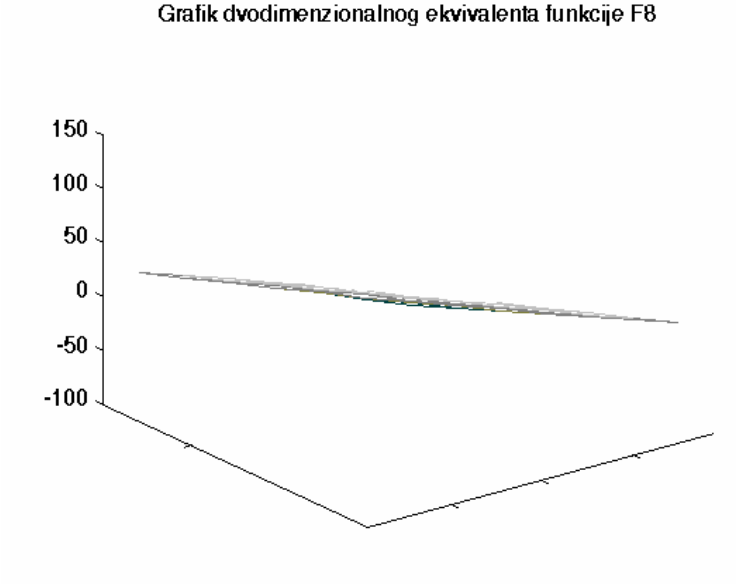
slika 5.13. Schwefelova funkcija (jednodimenzionalni ekvivalent)

3) Griewangkova funkcija:

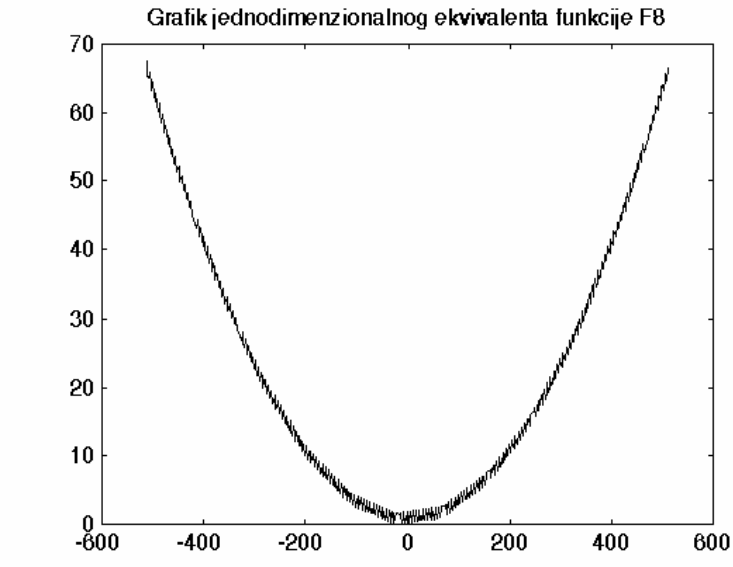
Analitički oblik Griewangkove funkcije je dat sledećom formulom:

$$F8(x_1,...,x_N) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$ (pri čemu je $-5.12 \le x_i \le 5.11$)

Grafici dvodimenzionalne i jednodimenzionalne restrikcije ove funkcije prikazani su na slikama 5.14. i 5.15.



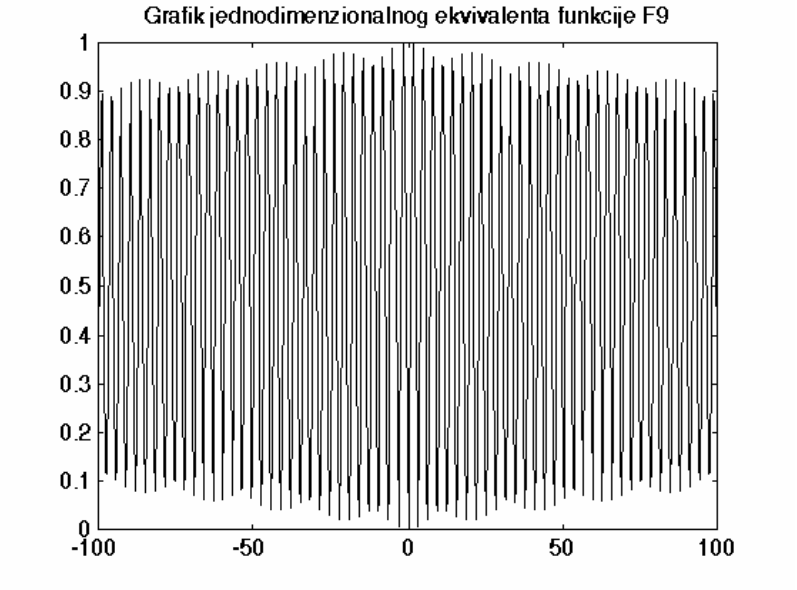
slika 5.14. Griewangkova funkcija (dvodimenzionalni ekvivalent)



slika 5.15. Griewangkova funkcija (jednodimenzionalni ekvivalent)

4) Sinusna ovojnica sinusni talas funkcija:
Grafik ove funkcije dat je slikom 5.16, a njen analitički oblik sledećom formulom:

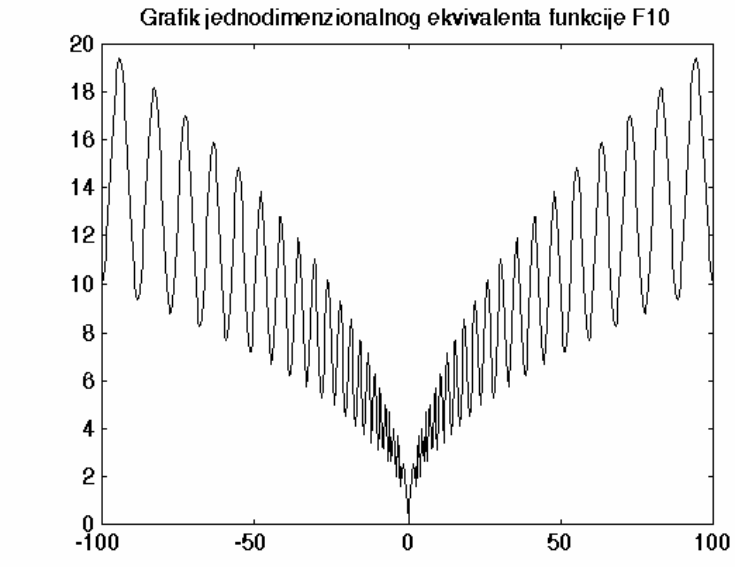
$$F9(x_1,x_2)=0.5+\frac{\sin^2\sqrt{x_1^2+x_2^2}-0.5}{\left(1+0.001\left(x_1^2+x_2^2\right)\right)^2}$$ (pri čemu je $-100 \le x_i \le 100$)



slika 5.16.

5) Skupljeno V sinusni talas funkcija:
Grafik ove funkcije dat je slikom 5.17, a njen analitički oblik sledećom formulom:

$$F10(x_1,x_2)=\sqrt[4]{x_1^2+x_2^2}\left(\sin^2\left(50\sqrt[10]{x_1^2+x_2^2}\right)+1\right)$$ (pri čemu je $-100 \le x_i \le 100$)



slika 5.17. Skupljeno V sinusni talas funkcija (jednodimenzionalni ekvivalent)

Učinjen je pokušaj da se, umesto nasumičnog dodavanja novih test-funkcija koje bi ispravile postojeću neravnotežu, definišu povoljne i nepovoljne osobine koje neka test funkcija treba da ima. Tako se zahteva (videti [Whitley95] i [Whitley96]) da skup test-funkcija obrazovan od starih i novih test-funkcija treba da zadovoljava sledeće osobine:

- Skup test-funkcija treba da sadrži probleme koji su otporni na metodu penjanja uz brdo. Problemi koji nisu otporni na metodu penjanja uz brdo, tj. problemi koji se mogu lako rešiti metodama zasnovanim isključivo na eksploataciji, su problemi sa malom nelinearnošću (malom epistazom). Kada je strategija penjanja uz brdo uspešna, tada je ona obično brža od evolutivnog algoritma. Ovo ne znači da treba isključiti iz skupa test-problema one probleme koji se mogu rešiti pomoću penjanja uz brdo, ali se pri analizi rezultata za takve probleme mora obratiti pažnja i na prethodno istaknutu činjenicu.
- Skup test-funkcija treba da sadrži probleme koji su nelinearni, nerazdvojivi i nesimetrični. Nelinearne (eng. nonlinear) funkcije sadrže nelinearnu interakciju između argumenata funkcije. Pojam epistaze, opisan u poglavlju 2, je u uskoj vezi sa nelinearnošću funkcije. Problem se naziva nerazdvojivim ukoliko nije razdvojiv. Problem je razdvojiv (eng. separable) ako nema nelinearne interakcije između promenljivih. Kod razdvojivih funkcija (videti [Thomp96]) se optimalna vrednost za svaki parametar može izračunati nezavisno od svih ostalih parametara. Funkcija je simetrična ukoliko se zamenom dve promenljive analitički izraz za funkciju ne menja. Za funkciju od N promenljivih može postojati $N!$ ovakvih veza. Eksploatisanjem eventualne simetričnosti funkcije može bitno da se umanji veličina prostora pretraživanja. Tada će se, iako izgleda da je broj čvorova u prostoru pretrage isuviše veliki, čisto enumerativna tehnika koja eksploatiše razdvojivost i simetričnost problema ponašati bolje od ma koje druge tehnike.
- Funkcije iz skupa test-funkcija treba da budu skalabilne. Primeri skalabilnih funkcija su funkcije *F7* i *F8*. Svojstvo skalabilnosti omogućava testiranja sa progresivno većim dimenzijama. Treba napomenuti da i nelinearna interakcija u test-funkcijama takođe treba biti osetljiva na skaliranje .
- Skupovi test-funkcija treba da sadrže probleme sa skalabilnom cenom evaluacije. Na primer, neki od problema seizmičke interpretacije podataka zbog promene jednog parametra moraju izvršiti $O(N^2)$ evaluacija, pri čemu je N dimenzija problema. Test funkcija treba da odslika i takve slučajeve.
- Test-problemi treba da imaju kanonsku formu. Naime, kada se za testiranje koriste funkcije *F1-F10,* često se načini kodiranja razlikuju (npr. klasično binarno, Grejovo binarno, realno). Pri testiranju ne samo što treba navesti koje se funkcije koriste, već i kakvo se kodiranje koristi.

Ako se pokuša ispitati da li funkcije F1-F10 zadovoljavaju gornje osobine, lako se može uočiti da:

- *F1*, *F3* i *F5* su razdvojive funkcije;

- *F4* je takođe razdvojiva, iako dodati šum (tj. vrednost slučajne promenljive sa normalizovanom normalnom raspodelom) može sprečiti algoritam da locira ekstremum;
- *F6* i *F7* su razdvojive funkcije;
- *F2*, *F9* i *F10* su nerazdvojivi, nelinearni problemi - ali su relativno male dimenzije i nisu skalabilni;
- Samo je funkcija *F8* skalabilna, nerazdvojiva i linearna; kao što se i na slici 5.15. vidi, grafik ove funkcije ima parabolični oblik kod kojeg kosinusna funkcija stvara "talase" preko paraboličke površi; međutim, sa porastom dimenzije problema uticaj kosinusne funkcije postaje manji, pa veličine lokalnih ekstremuma (tj. veličine "talasa") postaju sve manje; dakle, sa porastom dimenzije problema, grafik funkcije postaje sve više "ispeglan".

Predložena je metodologija za kreiranje novih test-funkcija, koje će imati prethodno pobrojanih pet svojstava. Nove test-funkcije treba da se prave od postojećih.

1) Proširivanje funkcije

Jedan način formiranja novih test-funkcija je proširivanje postojećih funkcija. Proširivanjem se dobija funkcija čija je dimenzija za jedan veća od dimenzije primitivne funkcije. Postupak proširivanja je sledeći:

Definicija 5.1.1. Neka je početna prosta funkcija označena sa *F(x,y)*. Tada se proširenje funkcije *F*, u oznaci *E-F(x,y,z)*, određuje na sledeći način:

$$E\text{-}F(x,y,z) = F(x,y) + F(y,z) + F(x,z).$$

Napomena. Lako se uočava da ovako formirana proširena funkcija nije razdvojiva i da uključuje nelinearnu interakciju između više promenljivih. □

Primer. Proširivanjem funkcije *F10* dobija se realna funkcija od tri promenljive *E-F10*, koja čuva dobre, a popravlja loše osobine funkcije *F10* prilikom testiranja. □

Pristup sugerisan definicijom 5.1.1. može i da se uopšti, kako bi se obezbedili različiti stepeni nelinearnosti.

	q	***x***	***y***	***z***
q		*qx*	*qy*	*qz*
x	*xq*		*xy*	*xz*
y	*yq*	*yx*		*yz*
z	*zq*	*zx*	*zy*	

tabela 5.1

Ilustracije radi, u tabeli 5.1. se u zaglavlju vrsta i kolona nalaze redom q, x, y, z. Njihove kombinacije, svako sa svakim, se nalaze unutar tabele. Od toga koji će među ponuđenim parovima iz tabele graditi proširenu funkciju četiri promenljive (nastalu proširivanjem primitivne funkcije dve promenljive), zavisi vrsta i stepen nelinearne interakcije među argumentima. Strategija punog matričnog skaliranja uzima sve moguće uređene parove različitih promenljivih. Tu postoje nelinearne interakcije ma koje promenljive sa bilo kojom drugom promenljivom. U tom slučaju složenost proširene funkcije je $O(n^2)$, pri čemu je n dimenzija problema kod proste funkcije.

2) Kompozicija funkcija

Drugi put za formiranje novih test-funkcija je kompozicija, tj. slaganje funkcija. Taj metod se kombinuje sa prethodno opisanim proširivanjem funkcija, kako bi se dobile složene nelinearne test-funkcije, tj. složeni nelinearni pejzaži prilagođenosti. Obično je unutrašnja prosta funkcija u oformljenoj kompoziciji, koju označavamo sa *T(x,y)*, realna funkcija dve promenljive, takva da joj je kodomen podskup domena spoljašnje proste funkcije $F(x_1,x_2,...,x_n)$. Kompozicija ove dve funkcije se dobija tako što se svaki od argumenata xi spoljašnje proste funkcije, zameni sa $T(x_i,x_{i+1})$, a argumenat x_n se zameni sa $T(x_n,x_1)$.

Napomena. Ukoliko je spoljašnja funkcija $F(x_1,x_2,...,x_n)$ razdvojiva (obično se i uzima takva spoljašnja funkcija), tada je:

$$F(x_1,x_2,...,x_n) = \sum_{i=1}^{n} S(x_i).$$

U tom slučaju, kompozicijom se dobija nova test-funkcija $E\text{-}F(T)(x_1,x_2,...,x_n)$, sledećeg oblika:

$$E - F(T)(x_1,x_2,...,x_n) = S(T(x_n,x_1)) + \sum_{i=1}^{n-1} S(T(x_i,x_{i+1})) \ \square$$

Primer. Kompozicijom funkcije *F8* i funkcije *AVG* (računa aritmetičku sredinu dva broja) dobija se funkcija *E-F8(AVG)*:

$$E-F8(AVG)(x_1,...,x_N)=1+\sum_{i=1}^{N-1}\frac{\left(\frac{x_i+x_{i+1}}{2}\right)^2}{4000}+\frac{\left(\frac{x_n+x_1}{2}\right)^2}{4000}-\prod_{i=1}^{N-1}\cos\left(\frac{\frac{x_i+x_{i+1}}{2}}{\sqrt{i}}\right)\cos\left(\frac{\frac{x_n+x_1}{2}}{\sqrt{n}}\right)$$

Ukoliko se kao unutrašnja funkcija koristi funkcija dve promenljive *F2*, dobija se *E-F8(F2)*:

$$E-F8(F2)(x_1,...,x_N)=1+\sum_{i=1}^{N-1}\frac{(F2(x_i+x_{i+1}))^2}{4000}+\frac{(F2(x_n+x_1))^2}{4000}-$$
$$-\prod_{i=1}^{N-1}\cos\left(\frac{F2(x_i+x_{i+1})}{\sqrt{i}}\right)\cos\left(\frac{F2(x_n+x_1)}{\sqrt{n}}\right)\square$$

5.4. Izvršavanje EA web servisa kod različitih domena problema

Ovo poglavlje je posvećeno opisu korišćenja EA web servisa u različitim domenima problema. Ranije je istaknuto da je jedna od osnovnih vodilja pri dizajnu EA sistema bila potreba za fleksibilnošću i mogućnost lakog uklapanja servisa u „enterprise" sisteme procesiranja. Da bi se dobio taj stepen fleksibilnosti, u nekoj meri je žrtvovana brzina izvršavanja, tako da se EA na ovom sistemu sporije izvršava nego što je to slučaj kod prethodno razvijenog GANP – naročito kod klasičnih GA. Važno je istaći (videti [Chong98], [Chong99], [Luke00]) da su i autori drugih EA softverskih sistema uočili slabosti u performansama svojih implementacija kreiranih u modernim jezicima/razvojnim okruženjima, kao što su Java i .NET, pri čemu su takođe uočili da je razlika u performansama manja kod GP nego što je to slučaj sa ostalim oblastima EA. Ipak, kvalitet nekog sosftverskog sistema, odnosno softverske komponente, nije određen samo brzinom izvršavanja koda, već i lakoćom korišćenja, programiranja i nadogradnje, fleksibilnošću dizajna i lakoćom uklapanja u složena izračunavanja – a prednosti EA web servisa u tim domenima, po mišljenu autora, bitno prevazilaze problem koji donose oslabljene performanse..

5.4.1. EA web servis i De Jongove funkcije

Optimizacija De Jongovih funkcija, kao i drugih funkcije je kod opisanog prethodno opisanog web servisa realizovana tako što se, pri pozivu, prosleđuje XML koji sadrži zapis te funkcije, pa servis tokom određivanja vrednosne funkcije (tj. funkcije objekcije) interpretira prosleđeni XML zapis. Tako je, na primer, EA web servisu za De Jongovu sfernu funkciju F1, prosleđen XML tekst sledećeg oblika:

```
<?xml version='1.0'?>
<!-- Parameters for invocation of EaWebService -->
<EaWebParams xmlns='EaWebParametersSchema.xsd'>
  <Communication>
    <Type>EaSequential</Type>
    <TopologyFile>Topologies/EaTopology01.xml</TopologyFile>
    <WebService>
      http://www.dev1.eraserver.net/VLADOFILIPOVIC/EaWebService/Main.asmx
    </WebService>
  </Communication>
  <Population>
    <ItemType>ItemBinary</ItemType>
    <ItemLenght>48</ItemLenght>
    <Count>116</Count>
  </Population>
  <Problem>
    <Type>ProblemFunctionOptimizationXml</Type>
     <InputFile>Problems/F01_Spheric.xml</InputFile>
  <Problem>
  <RandomGenerator>
    <Type>RandomGenIntrisic</Type>
    <UsePredefiniedSeed>true</UsePredefiniedSeed>
    <Seed>17</Seed>
  </RandomGenerator>
  <Crossover>
```

```
    <Type>CrossoverBinaryOnePoint</Type>
    <Probability>0.95</Probability>
  </Crossover>
  <Emigration>
    <Type>EmigrationIdle</Type>
  </Emigration>
  <Finish>
    <ItemsOr> FinishItemExecutionTimeElapsed FinshItemNumberOfGenerations</ItemsOr>
    <ItemsAnd> FinishItemIdle</ItemsAnd>
    <NumberOfGenerations>100</NumberOfGenerations>
    <ExecutionTimeDays>0</ExecutionTimeDays>
    <ExecutionTimeHours>0</ExecutionTimeHours>
    <ExecutionTimeMinutes>0</ExecutionTimeMinutes>
    <ExecutionTimeSeconds>30</ExecutionTimeSeconds>
    <ExecutionTimeMiliseconds>0</ExecutionTimeMiliseconds>
  </Finish>
  <Fitness>
    <Type>FitnessSubtractFrom</Type>
    <Value>3000</Value>
    <RemoveDuplicates>false</RemoveDuplicates>
  </Fitness>
  <Imigration>
    <Type>ImigrationIdle</Type>
  </Imigration>
  <Init>
    <Type>InitBinary</Type>
    <FromXmlFile>false</FromXmlFile>
  </Init>
  <Instance>
    <Type>InstanceFunctionOptimizationXmlBinary</Type>
  </Instance>
  <Mutation>
    <Type>MutationBinaryOnePoint</Type>
    <Probability>0.005</Probability>
  </Mutation>
  <NewGeneration>
    <UseElitistStrategy>true</UseElitistStrategy>
    <EliteNumber>1</EliteNumber>
  </NewGeneration>
  <ReportItems>
    <AA>
ReportItemBestAll ReportItemCommunication ReportItemCurrentGeneration
ReportItemEaParameters ReportItemFinishReason ReportItemProblemInput
ReportItemTimeInAlgorithm
    </AA>
    <AI>
ReportItemBestAll ReportItemCurrentGeneration ReportItemDeJongOffLine
ReportItemDeJongOnLine ReportItemTimeInIteration
    </AI>
    <ASCrossover> ReportItemIdle</ASCrossover>
    <ASFinish> ReportItemIdle</ASFinish>
    <ASFitness> ReportItemIdle</ASFitness>
    <ASInstance> ReportItemIdle</ASInstance>
    <ASMutation> ReportItemIdle</ASMutation>
    <ASProblem> ReportItemIdle</ASProblem>
    <ASSelection> ReportItemIdle</ASSelection>
    <Url>Reports/F01_Spheric.xml</Url>
  </ReportItems>
  <Selection>
    <Type>SelectionTournament</Type>
    <Probability>0.97</Probability>
    <UseDirectlyPassed>false</UseDirectlyPassed>
    <TournamentSize>3</TournamentSize>
  </Selection>
</EaWebParams>
```

pri čemu je funkcija koja se optimizuje $F1(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2$ opisana sledećim XML tekstom:

```
<EaProblemParameters xmlns:EA="urn:vlado1/EaWebService/Parameters">
```

```
  <Problem>
    <Description>F1 - Spheric function</Description>
    <Dimension>3</Dimension>
    <Interval>
      <Id>0</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>1</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>2</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Evaluate>
      <Condition>
        <Domain>
          <ConstantBool>
            true
          </ConstantBool>
        </Domain>
        <Expression>
          <BinaryPlus>
            <BinaryPlus>
              <Square>
                <Argument>
                  <Id>0</Id>
                </Argument>
              </Square>
              <Square>
                <Argument>
                  <Id>1</Id>
                </Argument>
              </Square>
            </BinaryPlus>
            <Square>
              <Argument>
                <Id>2</Id>
              </Argument>
            </Square>
          </BinaryPlus>
        </Expression>
      </Condition>
    </Evaluate>
    <KnownSolution>
      <Argument>
        <Id>0</Id>
        <Value>0</Value>
      </Argument>
      <Argument>
        <Id>1</Id>
        <Value>0</Value>
      </Argument>
      <Argument>
        <Id>2</Id>
        <Value>0</Value>
      </Argument>
    </KnownSolution>
  </Problem>
</EaProblemParameters>
```

Na sličan način, ako želimo da rešavamo De Jongovu funkciju F08 (tzv. Griewangkovu funkciju, čija je formula $F8(x_1,...,x_N) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right)$), XML će biti sledećeg oblika:

```
<EaProblemParameters xmlns:EA="urn:vlado1/EaWebService/Parameters">
  <Problem>
    <Description>F8 - Griewangk function</Description>
    <Dimension>15</Dimension>
    <Interval>
      <Id>0</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>1</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>2</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>3</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>4</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>5</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>6</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>7</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>8</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>9</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>10</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
    <Interval>
      <Id>11</Id>
      <From>-5.12</From>
      <To>5.12</To>
    </Interval>
```

```
<Interval>
  <Id>12</Id>
  <From>-5.12</From>
  <To>5.12</To>
</Interval>
<Interval>
  <Id>13</Id>
  <From>-5.12</From>
  <To>5.12</To>
</Interval>
<Interval>
  <Id>14</Id>
  <From>-5.12</From>
  <To>5.12</To>
</Interval>
<Evaluate>
  <Condition>
    <Domain>
      <ConstantBool>
        true
      </ConstantBool>
    </Domain>
    <Expression>
      <BinaryMinus>
        <BinaryPlus>
          <Constant>
            1
          </Constant>
          <Sum>
            <LowLimit>
              <Constant>
                0
              </Constant>
            </LowLimit>
            <HiLimit>
              <Constant>
                14
              </Constant>
            </HiLimit>
            <CurIndex>
               i
            </CurIndex>
            <Item>
              <BinaryDivide>
                <Square>
                  <Argument>
                    <Index>
                       i
                    </Index>
                  </Argument>
                </Square>
                <Constant>
                  4000
                </Constant>
              </BinaryDivide>
            </Item>
          </Sum>
        </BinaryPlus>
        <Product>
          <LowLimit>
            <Constant>
               0
            </Constant>
          </LowLimit>
          <HiLimit>
            <Constant>
              14
            </Constant>
          </HiLimit>
```

```
            <CurIndex>
              i
            </CurIndex>
            <Item>
              <Cosinus>
                <BinaryDivide>
                  <Argument>
                    <Index>
                      i
                    </Index>
                  </Argument>
                  <SquareRoot>
                    <IndexValue>
                      i
                    </IndexValue>
                  </SquareRoot>
                </BinaryDivide>
              </Cosinus>
            </Item>
          </Product>
        </BinaryMinus>
      </Expression>
    </Condition>
  </Evaluate>
 </Problem>
</EaProblemParameters>
```

Ako se želi iskoristiti drugo kodiranje, to se izuzetno lako realizuje. Ako bi klijent web servisa hteo da optimizuje funkciju F1, a da pri tome koristi reprezentaciju preko brojeva u pokretnom zarezu, tada bi XML sa parametrima bio npr. sledećeg oblika:

```
<?xml version='1.0'?>
<!-- Parameters for invocation of EaWebService -->
<EaWebParams xmlns='EaWebParametersSchema.xsd'>
  <Communication>
    <Type>EaSequential</Type>
    <TopologyFile>Topologies/EaTopology01.xml</TopologyFile>
    <WebService>
      http://www.dev1.eraserver.net/VLADOFILIPOVIC/EaWebService/Main.asmx
    </WebService>
  </Communication>
  <Population>
    <ItemType>ItemFloat</ItemType>
    <ItemLenght>3</ItemLenght>
    <Count>16</Count>
  </Population>
  <Problem>
    <Type>ProblemFunctionOptimizationXml</Type>
    <InputFile>Problems/F01_Spheric.xml</InputFile>
  </Problem>
  <RandomGenerator>
    <Type>RandomGenIntrisic</Type>
    <UsePredefiniedSeed>true</UsePredefiniedSeed>
    <Seed>17</Seed>
  </RandomGenerator>
  <Crossover>
    <Type>CrossoverFloatArithmeticGeometric</Type>
    <Probability>0.95</Probability>
  </Crossover>
  <Finish>
    <ItemsOr> FinishItemExecutionTimeElapsed FinshItemNumberOfGenerations</ItemsOr>
    <ItemsAnd> FinishItemIdle</ItemsAnd>
    <NumberOfGenerations>100</NumberOfGenerations>
    <ExecutionTimeDays>0</ExecutionTimeDays>
    <ExecutionTimeHours>0</ExecutionTimeHours>
    <ExecutionTimeMinutes>0</ExecutionTimeMinutes>
    <ExecutionTimeSeconds>30</ExecutionTimeSeconds>
    <ExecutionTimeMiliseconds>0</ExecutionTimeMiliseconds>
```

```
  </Finish>
  <Fitness>
    <Type>FitnessSubtractFrom</Type>
    <Value>3000</Value>
    <RemoveDuplicates>false</RemoveDuplicates>
  </Fitness>
  <Init>
    <Type>InitFloat</Type>
  </Init>
  <Instance>
    <Type>InstanceFunctionOptimizationXmlFloat</Type>
  </Instance>
  <Mutation>
    <Type>MutationFloatInvert</Type>
    <Probability>0.005</Probability>
  </Mutation>
  <NewGeneration>
    <UseElitistStrategy>true</UseElitistStrategy>
    <EliteNumber>1</EliteNumber>
  </NewGeneration>
  <ReportItems>
    <AA>
ReportItemBestAll ReportItemCommunication ReportItemCurrentGeneration
ReportItemEaParameters ReportItemFinishReason ReportItemProblemInput
ReportItemTimeInAlgorithm
    </AA>
    <AI>
ReportItemBestAll ReportItemCurrentGeneration ReportItemDeJongOffLine
ReportItemDeJongOnLine ReportItemTimeInIteration
    </AI>
    <ASCrossover> ReportItemIdle</ASCrossover>
    <ASFinish> ReportItemIdle</ASFinish>
    <ASFitness> ReportItemIdle</ASFitness>
    <ASInstance> ReportItemIdle</ASInstance>
    <ASMutation> ReportItemIdle</ASMutation>
    <ASProblem> ReportItemIdle</ASProblem>
    <ASSelection> ReportItemIdle</ASSelection>
    <Url>Reports/F01_Spheric.xml</Url>
  </ReportItems>
  <Selection>
    <Type>SelectionTournament</Type>
    <Probability>0.97</Probability>
    <UseDirectlyPassed>false</UseDirectlyPassed>
    <TournamentSize>3</TournamentSize>
  </Selection>
</EaWebParams>
```

Dakle, samo su promenjene XML sekcije `Item` unutar `Population`, `Instance`, `Init`, `Crossover` i `Mutation`. Već je ranije istaknuto da je web servis dizajniran tako da se objekti potrebni za izvršavanje EA kreiraju (korišćenjem refleksije koja je imanentna jeziku C#) pri prispeću zahteva od klijenta.
Ovakav pristup (sa interpretiranjem XML zapisa funkcija) je manje efikasan, ali mnogo fleksibilniji od uobičajenog. Kao što je u opisu softverskog sistema već istaknuto, EA web servis rezultat tj. izveštaj takođe šalje u obliku XML-a. Segment takvog izveštaja, za De Jongovu funkcije F1 gde je korišćeno binarno kodiranje ima sledeći oblik:

```
<?xml version='1.0'?>
<!--Report: Timestamp 2006 01 31 09:23:22:957 -->
<EaWebReport xmlns='EaWebReportsSchema.xsd'>
  <AfterAlgorithm>
    <BestAll>
      <Data>
        1000 0000 0000 0000 1000 0000 0000 0000 0111 1111 1111 1111 11
      </Data>
      <IsValid>True</IsValid>
      <ObjectionValue>2.44140625E-08</ObjectionValue>
      <FitnessValue>2999.99999997559</FitnessValue>
      <ProblemArguments> 0 0 -0.00015625</ProblemArguments>
    </BestAll>
```

```
    <CurrentGeneration>101</CurrentGeneration>
    <FinishReason>Number of generations is reached!</FinishReason>
    <TimeInAlgorithm>
      <Days>0</Days>
      <Hours>0</Hours>
      <Minutes>0</Minutes>
      <Seconds>25</Seconds>
      <Miliseconds>15</Miliseconds>
      <Summary>00:00:25.0159712</Summary>
    </TimeInAlgorithm>
  </AfterAlgorithm>
</EaWebReport>
```

Sa druge strane, segment XML-a sa dobijenim rezultatom u slučaju kada se izvršava EA sa reprezentacijom u pokretnom zarezu je sledećeg oblika:

```
<EaWebReport>
  <AfterAlgorithm>
    <BestAll>
      <Data>
        0.338895187405355 0.485176505281206 0.327682401206196
      </Data>
     <IsValid>True</IsValid>
      <ObjectionValue>5.85816882449178</ObjectionValue>
      <FitnessValue>194.141831175508</FitnessValue>
      <ProblemArguments>
        -1.64971328096917 -0.151792585920446 -1.76453221164855
      </ProblemArguments>
    </BestAll>
    <FinishReason>Number of generations is reached!</FinishReason>
    <TimeInAlgorithm>
      <Days>0</Days>
      <Hours>0</Hours>
      <Minutes>0</Minutes>
      <Seconds>1</Seconds>
      <Miliseconds>832</Miliseconds>
      <Summary>00:00:01.8326352</Summary>
    </TimeInAlgorithm>
  </AfterAlgorithm>
</EaWebReport>
```

Naravno, sadržaj i format dobijenog rezultata zavisi od toga kako su postavljeni parametri za izveštaje kada je poslet zahtev web servisu.

5.4.2. Korišćenje EA web servisa kod GP izračunavanja

U cilju korišćenja novodizajniranog EA web servisa kao alata za genetsko programiranje, potrebno je osmisliti i implementirati odgovarajuću reprezentaciju za jedinke GP, tj. za drvoidnu reprezentaciju. To je urađeno korišćenjem XML-a. GP se obično koristi za rešavanje problema kao što su: odrešivanje analitičkog oblika formule kada su poznati eksperimentalni podaci koji je zadovoljavaju, dizajniranje programa autonomnog robota koji se kreće kroz lavirint, kao i drugi problemi koji obično spadaju u domen veštačke inteligencije. U ovoj sekciji biće opisano kako se EA web servis može koristiti za dobijanje formule koja zadovoljava eksperimentalne podatke, pri čemu struktura formule nije unapred određena. U suštini, biće ponovljen (samo u malo uprošćenijoj formulaciji) znameniti eksperiment koji je izveo John Koza i na taj način će biti prikazano kako se EA web servis može koristiti i u ovom segmentu EA.
Neka je problem koji se rešava opisan sledećim XMLtekstom:

```
<!--Solving linear equation ax=b -->
<LinearEquation>
  <Dimension> 9  </Dimension>
  <Element>
    <Id>0</Id>
    <Coeficients>
      <a>3</a>
      <b>5</b>
    </Coeficients>
    <Results>
      <x>1.6666666666666666666666666666667</x>
```

```
  </Results>
</Element>
<Element>
  <Id>1</Id>
  <Coeficients>
    <a>2</a>
    <b>1</b>
  </Coeficients>
  <Results>
    <x>0.5</x>
  </Results>
</Element>
<Element>
  <Id>2</Id>
  <Coeficients>
    <a>3</a>
    <b>2</b>
  </Coeficients>
  <Results>
    <x>0.66666666666666666666666666666667</x>
  </Results>
</Element>
<Element>
  <Id>3</Id>
  <Coeficients>
    <a>1</a>
    <b>2</b>
  </Coeficients>
  <Results>
    <x>2</x>
  </Results>
</Element>
<Element>
  <Id>4</Id>
  <Coeficients>
    <a>1</a>
    <b>7</b>
  </Coeficients>
  <Results>
    <x>7</x>
  </Results>
</Element>
<Element>
  <Id>5</Id>
  <Coeficients>
    <a>3</a>
    <b>7</b>
  </Coeficients>
  <Results>
    <x>2.33333333333333333333333333333333</x>
  </Results>
</Element>
<Element>
  <Id>6</Id>
  <Coeficients>
    <a>8</a>
    <b>7</b>
  </Coeficients>
  <Results>
    <x>0.875</x>
  </Results>
</Element>
<Element>
  <Id>7</Id>
  <Coeficients>
    <a>8</a>
    <b>-7</b>
  </Coeficients>
  <Results>
```

```
      <x>-0.875</x>
    </Results>
  </Element>
  <Element>
    <Id>8</Id>
    <Coeficients>
      <a>-8</a>
      <b>7</b>
    </Coeficients>
    <Results>
      <x>-0.875</x>
    </Results>
  </Element>
</LinearEquation>
```

Dakle, dato je devet trojki (*a,b,x*), pri čemu *x* predstavlja rešenje a *a* i *b* su koeficijenti. EA (preciznije GP) traži koja se formula, izgrađena od osnovnih operacija, najbolje uklapa u podatke koji su dati.
Pri izvršavanju EA, parametri koji se prosleđuju EA web servisu su istog formata kao u prethodnim slučajevima (postavi se drugi tip problema, instance, jedinke, inicijalizacije, ukrštanja i mutacije):

```
<?xml version='1.0'?>
<!-- Parameters for invocation of EaWebService -->
<EaWebParams xmlns='EaWebParametersSchema.xsd'>
  <Communication>
    <Type>EaWebService.EaSequential</Type>
    <TopologyFile>Topologies/EaTopology01.xml</TopologyFile>
    <WebService>
      http://www.dev1.eraserver.net/VLADOFILIPOVIC/EaWebService/Main.asmx
    </WebService>
  </Communication>
  <Population>
    <ItemType>ItemXml</ItemType>
  <Count>76</Count>
  </Population>
  <Problem>
    <Type>ProblemLinearEquation</Type>
    <InputFile>Problems/LinearEquation01.xml</InputFile>
  </Problem>
  <RandomGenerator>
    <Type>RandomGenIntrisic</Type>
    <UsePredefiniedSeed>False</UsePredefiniedSeed>
  </RandomGenerator>
  <Crossover>
    <Type>CrossoverXmlOnePoint</Type>
    <Probability>0.999</Probability>
  </Crossover>
  <Finish>
    <ItemsOr> FinishItemNumberOfGenerations</ItemsOr>
    <ItemsAnd> FinishItemIdle</ItemsAnd>
    <NumberOfGenerations>22</NumberOfGenerations>
  </Finish>
  <Fitness>
    <Type>FitnessChangeSign</Type>
    <RemoveDuplicates>False</RemoveDuplicates>
  </Fitness>
  <Init>
    <Type>InitXml</Type>
    <InitData>
      <Length1>
        3
      </Length1>
      <Length2>
        6
      </Length2>
      <LengthDistribution>
        Uniform
      </LengthDistribution>
      <Alphabet>
```

```
      <Dimension>
        9
      </Dimension>
      <Item>
       <Id>0</Id>
       <Letter>a</Letter>
       <Cardinality>0</Cardinality>
      </Item>
      <Item>
       <Id>1</Id>
       <Letter>b</Letter>
       <Cardinality>0</Cardinality>
      </Item>
      <Item>
       <Id>2</Id>
       <Letter>ChangeSign</Letter>
       <Cardinality>1</Cardinality>
      </Item>
      <Item>
       <Id>3</Id>
       <Letter>AbsoluteValue</Letter>
       <Cardinality>1</Cardinality>
      </Item>
      <Item>
       <Id>4</Id>
       <Letter>Square</Letter>
       <Cardinality>1</Cardinality>
      </Item>
      <Item>
       <Id>5</Id>
       <Letter>Plus</Letter>
       <Cardinality>2</Cardinality>
      </Item>
      <Item>
       <Id>6</Id>
       <Letter>Minus</Letter>
       <Cardinality>2</Cardinality>
      </Item>
      <Item>
       <Id>7</Id>
       <Letter>Multiply</Letter>
       <Cardinality>2</Cardinality>
      </Item>
      <Item>
       <Id>8</Id>
       <Letter>Divide</Letter>
       <Cardinality>2</Cardinality>
      </Item>
    </Alphabet>
  </InitData>
 </Init>
 <Instance>
   <Type>InstanceLinearEquationXml</Type>
 </Instance>
 <Mutation>
   <Type>MutationXmlOnePoint</Type>
   <Probability>0.005</Probability>
 </Mutation>
 <NewGeneration>
   <UseElitistStrategy>True</UseElitistStrategy>
   <EliteNumber>1</EliteNumber>
 </NewGeneration>
 <ReportItems>
   <AA>
ReportItemBestAll ReportItemCommunication ReportItemCurrentGeneration
ReportItemDeJongOffLine ReportItemDeJongOnLine ReportItemEaParameters
ReportItemFinishReason ReportItemFirstAll ReportItemHostId ReportItemIdle
ReportItemOverallTime ReportItemProblemInput ReportItemRequestId
ReportItemTimeInAlgorithm ReportItemTimeInIteration ReportItemInitAlphabet
```

```
    </AA>
    <AI>
ReportItemBestAll ReportItemCurrentGeneration ReportItemIdle ReportItemPopulationAll
    </AI>
    <ASCrossover> ReportItemIdle ReportItemPopulationAll</ASCrossover>
    <ASFinish> ReportItemIdle</ASFinish>
    <ASFitness> ReportItemIdle</ASFitness>
    <ASInstance> ReportItemIdle</ASInstance>
    <ASMutation> ReportItemIdle</ASMutation>
    <ASProblem> ReportItemIdle</ASProblem>
    <ASSelection> ReportItemIdle</ASSelection>
    <Url>Reports/EaWeb98.xml</Url>
  </ReportItems>
  <Selection>
    <Type>SelectionTournament</Type>
    <Probability>0.99</Probability>
    <UseDirectlyPassed>False</UseDirectlyPassed>
    <TournamentSize>3</TournamentSize>
  </Selection>
</EaWebParams>
```

Informacije o strukturi formule (naravno, kako se radi o GP, formula je predstavljena drvetom) koje su poznate pre početka uklapanja se nalaze u sekciji `InitData` XML parametara – tu je maksimalni i minimalni broj čvorova drveta, ciljna raspodela veličina drveta, kao i opis svih vrednosti koje se mogu naći u datom čvoru drveta.
Tokom izvršenja EA, deo izveštaja o dobijenim rešenjima može biti sledećeg oblika:

```
<?xml version='1.0'?>
<!--Report: Timestamp 2006 02 02 22:14:25:947 -->
<EaWebReport xmlns='EaWebReportsSchema.xsd'>
  </AfterIteration>
    </PopulationAll>
       ...
      <Item>
        <Id>74</Id>
        <Data>
          <b>
        </Data>
        <IsValid>True</IsValid>
        <ObjectionValue>29.9583333333333</ObjectionValue>
        <FitnessValue>-29.9583333333333</FitnessValue>
      </Item>
      <Item>
        <Id>75</Id>
        <Data>
          <Divide>
            <a>
            <b>
          </Divide>
        </Data>
        <IsValid>True</IsValid>
        <ObjectionValue>35.9583333333333</ObjectionValue>
        <FitnessValue>-35.9583333333333</FitnessValue>
      </Item>
    </PopulationAll>
  </AfterIteration>
  <AfterAlgorithm>
    <BestAll>
      <Data>
        <Divide>
          <b>
          <a>
        </Divide>
      </Data>
      <IsValid>True</IsValid>
      <ObjectionValue>0</ObjectionValue>
      <FitnessValue>0</FitnessValue>
```

```
    </BestAll>
    <FinishReason>
      Number of generations is reached!
    </FinishReason>
    <OverallTime>
      <Days>0</Days>
      <Hours>0</Hours>
      <Minutes>0</Minutes>
      <Seconds>7</Seconds>
      <Miliseconds>981</Miliseconds>
      <Summary>00:00:07.9814768</Summary>
    </OverallTime>
    <TimeInAlgorithm>
      <Days>0</Days>
      <Hours>0</Hours>
      <Minutes>0</Minutes>
      <Seconds>7</Seconds>
      <Miliseconds>931</Miliseconds>
      <Summary>00:00:07.9314048</Summary>
    </TimeInAlgorithm>
    <TimeInIteration>
      <Days>0</Days>
      <Hours>0</Hours>
      <Minutes>0</Minutes>
      <Seconds>0</Seconds>
      <Mili1seconds>350</Miliseconds>
      <Summary>00:00:00.3505040</Summary>
     </TimeInIteration>
  </AfterAlgorithm>
</EaWebReport>
```

5.5. Realni NP-teški problemi

De Jongove test-funkcije, kao i ostale prethodno pomenute funkcije su nešto što je karakteristično samo za poređenje performansi raznih EA, a prava mera korisnosti i kvaliteta predloženih modifikacija vidi se pri rešavanju realnih problema. Rešavanje nekog NP teškog problema i poređenje rezultata sa svim ostalim postojećim metodama za rešavanje datog problema, na realnim instancama velike dimenzije, predstavlja vrlo dobar pokazatelj kvaliteta primenjenog EA.
U ovoj sekciji će biti opisani razni problemi gde je FGTS do sada primenjivana i postigla najbolje rezultate u poređenju sa svim ostalim postojećim operatorima selekcije. Ne samo to, već je korišćenje FGTS, uz adekvatan i pažljiv izbor svih ostalih aspekata EA, doprinelo da rezultati takvog EA budu uporedivi ili bolji u odnosu na sve prethodno postojeće metode za rešavanje dole pomenutih problema. Na žalost, detaljno predstavljanje i analiza svih takvih doprinosa daleko prevazilazi obim ovog rada.
Instance problema koji su rešavani su obično uzimane iz ORLIB biblioteke [BeaslJ96], osim u situacijama kada u ORLIB-u nije bilo instanci izabranih problema, ili kada instance problema iz ove biblioteke nisu dovoljno teški za pravljenje razlike među algoritmima (to je bio slučaj kod SPLP i UNDP). Tada su autori bili prinuđeni da sami oforme dovoljno teške instance i te instance su potom bile uzimane od strane drugih istraživača kao referenca, tj. osnova za poređenje kvaliteta algoritama.

5.5.1. Prosti lokacijski problem (SPLP)

Lokacijski problemi predstavljaju posebnu klasu zadataka kombinatorne optimizacije. Oni imaju veliku primenu u raznim oblastima, a posebno u snabdevanju, planiranju zaliha i računarskim mrežama. Prost lokacijski problem (eng. Simple Plant Location Problem – SPLP, a ponegde se još naziva i Uncapacitated Warehouse Location Problem, Uncapacitated Facility Location Problem ili Uncapacitated Plant Location Problem) je osnovni predstavnik ove klase problema, i mnogi lokacijski problemi se mogu formulisati pomoću njega.

Matematička formulacija

Prost lokacijski problem može biti formulisan na sledeći način: 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 $b_j = 1$.

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:

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

uz uslove:

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

$0 \le x_{ij} \le y_i \ \wedge \ y_i \in \{0,1\}$, za svakog snabdevača $i \in I$ i svakog korisnika $j \in J$;

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 ne ($y_i = 0$).

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

Metode rešavanja problema

Pri rešavanju SPLP problema primenjivani su mnogobrojni raznovrsni pristupi. Prikaz svih važnih doprinosa koji se odnose na SPLP prevazilazi okvire ovog rada. Stoga će se u radu pobrojati samo nekoliko najpoznatijih i najefikasnijih metoda rešavanja SPLP.

Algoritam Dualoc, čiji je autor Erlenkotter (videti [Erlen78]) je tokom dugog perioda bio najbrži algoritam za rešavanje SPLP. Osnova za datu metodu je dualna formulacija pridruženog problema linearnog programiranja (eng. LP dual) u skraćenoj formi, koji doprinosi prostoj implementaciji dualne metode penjanja (eng. dual ascent) i odgovarajuće dualne metode poravnanja (eng. dual adjustment). Ako rešenje dobijeno primenom tih dveju metoda nije optimalno, proces se nastavlja primenom metode grananja i ograničavanja (eng. branch-and-bound - BnB). 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

Krajem 80-tih je predloženo poboljšanje Lagranževe dualne metode penjanja, korišćenjem Lagranževe relaksacije uz pomoć Benderovih nejednakosti, generisanih u toku rada. Takođe je dat način za smanjivanje jaza između polaznog i dualnog problema (eng. integrality gap) - povezivanjem date metode sa dobrom heuristikom polaznog problema (videti [Guigna88]).

Koristeći formulaciju karakterističnu za problem pokrivanja, u [Simao89] je prikazana jedna varijanta dualnog simplex algoritma (eng. stramlined dual simplex method). Rezultati na standardnim instancama ovog problema ukazuju na superiornost dualnih metoda.

U radu [Körk89], 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.

Alves i Almeida su u radu [Alves92] poredili rezultate četiri verzije algoritma simuliranog kaljenja (eng. 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.

Predloženo je, u radu [Holmbe95], korišćenje primalno-dualne metode zasnovane na dekompoziciji. U polaznom (tj. primalnom) podproblemu fiksiraju se neke promenljive, pri čemu se relaksiraju neki uslovi u dualnom podproblemu, ali se fiksiraju njihovi Lagranževi množioci. U takvim slučajevima problem je mnogo lakši za rešavanje, u odnosu na polaznu postavku. Rezultati prikazani u radu sugerišu da su neke kombinacije ovakvih pristupa bolje u odnosu na Erlenkotter-ovu dualnu metodu penjanja.

Jedan uspešan pokušaj hibridizacije Add-heuristike i dualnih metoda je prikazan u [Tcha88]. Iako se tim načinom rešavanja dobijaju približna (suboptimalna) rešenja, koja mogu biti i nešto lošijeg kvaliteta kod instanci veće dimenzije, vreme izvršavanja je obično vrlo kratko. Ovaj pristup, hibridizacija Add heuristike sa evolutivnim (tj. genetskim) algoritmom je realizovan i detaljno opisan i analiziran u radu [Kratic98a].

Instance problema

Instance 41-134 i A-C su preuzete iz ORLIB-a (videti [BeaslJ90], [BeaslJ96]).

Instanca problema	Veličina	Veličina datoteke
41 - 44, 51, 61 - 64, 71-74	16×50	10 KB
81 - 84, 91 - 94, 101 – 104	25×50	15 KB
111-114,121-124,131-134	50×50	31 KB
A – C	100×1000	1.2 MB

tabela 5.2. Parametri SPLP instanci preuzeti iz ORLIB-a

Najveći broj gornjih instanci nije dovoljno velik da bi se ponašanje algoritma moglo adekvatno testirati. Stoga su selekcione strategije, pored ORLIB instanci, testirane i na generisanim instancama (koje su detaljno opisane u [Kratic99a], [Kratic00], [Kratic01], [Kratic01b]).

Kako su dati problemi relativno manje veličine, pa ne mogu realno predstaviti mogućnosti EA pri rešavanju SPLP, od generisane su 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 $[b_{min}, b_{max}]$;
- Cena 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 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 sledeće formule

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

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.

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 5.3. Tipovi parametara SPLP instanci

Instanca	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

tabela 5.4. Karakteristike generisanih SPLP instanci

Generisane instance, prikazane u tabeli 5.3, imaju mali broj beskorisnih lokacija snabdevača (lokacija snabdevača za koje nema šanse da budu uspostavljena) i veoma veliki broj suboptimalnih rešenja. One se stoga jako teško rešavaju korišćenjem dualno zasnovanih metoda i drugih tehnika tipa grananja i ograničavanja.

EA implementacija

Za rešavanje problema SPLP pomoću EA je korišćeno prethodno opisan GANP sistem. Što se strukture samog evolutivnog procesa i izbora veličine parametara tiče, posle pažljive analize donesene su sledeće odluke (detaljno opisane i obrazložene u [Kratic00]):

- 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. Iz genetskog koda se direktno nalazi niz indikatora y_i (i=1, 2, ... , m) koji ukazuju da li je na odgovarajućoj potencijalnoj lokaciji postavljen snabdevač ili ne. Budući da za snabdevača na određenoj lokaciji ne postoji ograničenje u kapacitetu lokacije i da 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.
- Implementacija efikasne funkcije za evaluaciju, tj. računanje prilagođenosti jedinke, odnosno funkcije objekcije zahteva da se izvrši pravilan izbor strukture podataka. Radi ubrzanja pri računanju vrednosne funkcije, koristi se 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 bibliotečke `qsort()` C funkcije. Pri tome je zauzeće memorije oko 50% veće, ali je izvršavanje programa nekoliko puta brže.
- 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.

  1. 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. Kako 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 indeksa, pri nalaženju najpogodnije lokacije za snabdevanje. Zbog toga je vremenska složenost vrednosne funkcije u ovom slučaju jednaka $O\left(m + n \cdot \frac{m}{e}\right)$.

  2. 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 $O(m + n \cdot e)$.

- Za rešavanje datog problema u ovom eksperimentu primenjena je selekcija bazirana na rangu, sa linearnim smanjenjem rangova, sa korakom 0.012, od nivoa 2.5 za najbolju jedinku, do nivoa od 0.712 za najlošiju jedinku.
- Pri izvršavanju EA implementacije za rešavanje SPLP, 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:

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

  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.
- 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, što je praktično anuliralo prednosti bolje funkcije prilagođenosti u prvoj generaciji. Pri generisanju početne populacije je primenjena ista učestalost 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.
- Da bi se izbegla preuranjena konvergencija, 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.
- Implementirani evolutivni algoritam za rešavanje SPLP kreira populaciju od 150 jedinki, i koristi je elitističku strategiju. 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:

$$(novo)\, f_i = \begin{cases} f_i - \bar{f}, & f_i > \bar{f} \\ 0, & f_i \le \bar{f} \end{cases}, \text{ gde je } \bar{f} = \frac{1}{N_{pop}} \cdot \sum_{i=1}^{N_{pop}} f_i$$

- Jedinke koje su direktno prešle u sledeću generaciju, zatim, zajedno sa ostalim (nepovlašćenim) jedinkama učestvuju u izboru preostalih 50 jedinki (1/3) populacije u novoj generaciji.

Poređenje algoritama za rešavanje SPLP i eksperimentalni rezultati

Od svih prethodnih metoda za rešavanje prostog lokacijskog problema u opštem slučaju, u praksi su do sada najbolje rezultate dale dualne. 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.

Dualoc

Jedna efikasna implementacija za rešavanja datog problema, koja je javno dostupna, je Erlenkotter-ov Dualoc, detaljno opisan u radovima [Erlen78]. Da bi poređenje performansi bilo verodostojnije, na SPLP instancama su izvršene dve njegove varijante:

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

Izvršavanje je obavljeno na PC kompatibilnom računaru sa procesorom AMD 80486 na 133 MHz sa 64MB osnovne memorije (64 MIPS, 21.8 Mflops). Rezultati izvršavanja na prethodno opisanim SPLP instancama su dati 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, odnosno najbolje dobijeno rešenje – ako nije moguće proveriti optimalnost.

Tokom izvršavanja je daleko najbolje rezultate pokazala varijanta Dualoc-a pri kojoj se u početnom čvoru metode grananja i ograničavanja vrši maksimalno dualno poboljšavanje, dok se u kasnijim čvorovima vrši jednostruko dualno poboljšavanje. 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) Koristi se termin najbolje dobijeno rešenje (a ne optimalno rešenje), jer je to rešenje dobijeno pomoću EA, pa nije bilo moguće dokazati njegovu optimalnost.

Instanca	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	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%)

tabela 5.5 Rezultati Dualoc-a

Redukovani Dualoc

Budući da redukovana verzija ne sadrži grananje i ograničavanje, 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.

Kao što je već istaknuto, 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).

Instanca	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%)

tabela 5.6 Rezultati Dualoc-a bez BnB

EA

Izvršavanje je i za EA obavljeno na istoj PC konfiguraciji kao i u prethodna dva slučaja. Kako genetski operatori daju nedeterminističke rezultate, svaka instanca problema je izvršavana više puta. Radi bolje preglednosti, u tabeli 5.7. su dati samo prosečni rezultati svake grupe instanci.

Analizirajući podatke iz tabele 5.7, vidimo da je u svim izvršavanjima rezultat bilo optimalno rešenje (ili NDR), uz relativno kratko vreme izvršavanja, čak i za SPLP instance vrlo velike dimenzije (videti [Filipo03]).

Instanca	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	-

tabela 5.7 Rezultati EA

Kao što vidimo iz tabela 5.5 – 5.7, obe verzije Dualoc-a su skoro trenutno (manje od 0.01 sekundi), u samo nekoliko iteracija rešile SPLP instance 41-134. GANP implementacija EA 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 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 EA (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. Primećuje se da porastom dimenzije problema vreme izvršavanja redukovanog Dualoc-a mnogo brže raste u odnosu EA, tako da se već MS instance sporije rešavaju pomoću redukovanog Dualoc-a u odnosu na evolutivni algoritam (uz ogromnu razliku u kvalitetu dobijenog rešenja). Zbog toga će u narednom delu biti upoređene samo performanse originalnog Dualoc-a i GANP implementacije EA.

Oba algoritma (Dualoc i EA) su optimalno rešila sve MO-MQ instance, ali je vreme izvršavanja evolutivnog algoritma 10-200 puta kraće. Razlika u brzini je još veća (oko 1800 puta) za instance iz klase MR i stiče se utisak da dalje eksponencijalno 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 EA.

Dobijeni rezultati se mogu analizirati imajući u vidu način na koji date metode (Dualoc i EA) dolaze do rešenja.

- Dualoc primenjuje grananje i ograničavanje, 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 iteracija u procesu grananja i ograničavanja.
- EA, nasuprot tome, predstavlja robustan način rešavanja NP-kompletnih problema (videti [Filipo00], [Kratic00]), 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 EA, jer se vrlo brzo nakon početka izvršavanja EA u populaciji pojavljuju, a zatim i preovlađuju, jedinke sa vrednostima bliskim optimalnom rešenju. Njihovom rekombinacijom se zatim vrlo lako poboljšavaju date vrednosti, i postupak najčešće dosta brzo konvergira prema optimalnom rešenju.

Poređenje selekcionih operatora kod EA na SPLP i eksperimentalni rezultati

Implementacija EA koja se koristi za rešavanje SPLP i za poređenje selekcionih metoda je zasnovana na binarnoj reprezentaciji, uniformnom ukrštanju, prostoj mutaciji, stacionarnoj zameni jedinki u populaciji, elitnoj strategiji i keširanju. U ovom eksperimentu su fiksirani svi oni parametri koji su se pokazali jako dobrim kako tokom prethodnog eksperimenta, tako i tokom drugih proučavanja rešavanja SPLP koje je autor preduzimao (videti [Filipo00], [Filipo01] [Kratic96], [Kratic98a], [Kratic01], [Kratic01b]). Drugim rečima, EA je pri poređenju drugim tehnikama za rešavanje SPLP u seriji eksperimenata baš sa tim parametrima dala izuzetno dobre rezultate. Jedino što se varira u eksperimentu koji sledi jeste operator selekcije. Porediće se tri tipa selekcionih metoda: selekcija bazirana na rangu, gde rang opada od 2.5 (za najbolju jedinku) do 0.7 (za najgoru) sa korakom 0.012; klasična turnirska selekcija, sa veličinama turnira $N_{tour} \in \{5, 6\}$; fino gradirana turnirska selekcija sa željenom srednjom veličinom turnira $F_{tour} \in \{4.5, 5.5, 5.6, 5.8, 6.2, 6.4\}$.

Da još jednom rekapituliramo osobine evolutivnog algoritma (detaljno opisane u [Filipo03]):

- Već je istaknuto da se kao operator ukrštanja koristi uniformno ukrštanje, što obezbeđuje minimalno prekidanje gena jedinke. Ovo je važan elemenat uspeha algoritma, jer interakcija između gena u genetskom kodu jedinke nije suviše velika.
- Implementacija EA koristi prostu mutaciju, čije je vreme izvršavanja poboljšano realizacijom preko normalne raspodele. Nivo mutacije zavisi od dimenzije problema i on eksponencijalno opada od $0.4/n$ prema $0.15/n$.
- Politika zamene populacija u populaciji je stacionarna, sa primenom elitne strategije. Preciznije, u svakoj generaciji se zameni samo 1/3 jedinki u populaciji. Preostalih 2/3 jedinki se direktno prosledi u sledeću generaciju. Za te elitne jedinke ne treba ponovo računati prilagođenost.
- Najveći broj generacija je $2000n$. Ako je najbolja jedinka nepromenjena tokom $1000n$ generacija, EA će završiti sa izvršavanjem.
- Veličina populacije je 150 jedinki i jedinke se u prvoj generaciji inicijalizuju na slučajan način. Jedinke su kodirane binarnim kodom.
- Na kraju, mada ne najmanje značajno, treba istaći da je efikasnost algoritma poboljšana njegovim keširanjem. Za keširanje EA je korišćena jednostavna, ali veoma efikasna strategija LRU (eng. Least Recently Used). Heširanje je implementirano korišćenjem heš-red strukture podataka i detaljno je opisano i analizirano u [Kratic99] i [Kratic00].

Rezultati prikazani u tabelama su dobijeni izvršavanjem EA (GANP implementacija) na Pentium III/600MHz PC, sa 330 megabajta unutrašnje memorije.

Sve prikazane vrednosti su dobijene kao prosek u 20 nezavisnih izvršavanja instance problema (videti [Filipo03]).

Radi bolje vidljivosti, rezultati su u tabelama 5.8. do 5.11 sumarno prikazani za svaku od grupa instanci, a ne za svaku od instanci ponaosob. Kolone tabela predstavljaju grupe instanci, a svaka od vrsta predstavlja selekcioni metod.

Tabele sadrže rezultate za selekciju bazirana na rangu (gde rang linearno opada od 2.5 do 0.7); klasičnu turnirski selekciju (gde $N_{tour} \in \{5, 6\}$) i fino gradiranu turnirsku selekciju (gde je $F_{tour} \in \{4.5, 5.5, 5.6, 5.8, 6.2, 6.4\}$).

U svakoj ćeliji tabele smeštene su dve vrednosti. Gornja vrednost je prosečan broj generacija potreban za izvršenje algoritma, a donji je prosečno vreme izvršavanja (u sekundama).

U svim izvršavanjima svih programskih instanci dobijen je isti rezultat – optimalan ili NDR. Najbolja vremena za svaku grupu instanci su podebljana.

Selekcija	41-74	81-104
r. b.	17.7	33.6
(r = 2.5 to 0.7)	0.10	0.17
f. g. t. (Ftour=4.5)	10.4	34.4
	0.07	0.17
t. (Ntour=5)	9.1	49.1
	0.06	0.21
f. g. t. (Ftour=5.5)	9	53.5
	0.06	0.24
f. g. t. (Ftour=5.6)	8.8	41.3
	0.05	0.18
f. g. t. (Ftour=5.8)	8.9	46.6
	0.05	0.20
t. (Ntour=6)	8.8	67.1
	0.05	0.30
f. g. t. (Ftour=6.2)	8.5	77.1
	0.05	0.34
f. g. t. (Ftour=6.4)	8.8	87.1
	0.05	0.38

tabela 5.8. Poređenje selekcionih operatora na ORLIB instancama 41-74 i 81-104

Selekcija	111-134	A-C
r. b.	109.2	1328
(r = 2.5 to 0.7)	0.51	16.28
f. g. t. (Ftour=4.5)	145.3	1633
	0.65	16.78
t. (Ntour=5)	128.9	1433
	0.52	12.72
f. g. t. (Ftour=5.5)	136.6	1209
	0.62	12.24
f. g. t. (Ftour=5.6)	136.3	1694
	0.55	12.47
f. g. t. (Ftour=5.8)	151.2	2347
	0.61	16.81
t. (Ntour=6)	146	1078
	0.65	10.74
f. g. t. (Ftour=6.2)	164.4	1890
	0.73	18.22
f. g. t. (Ftour=6.4)	148.6	2076
	0.66	20.06

tabela 5.9. Poređenje selekcionih operatora na ORLIB instancama 111-134 i A-C

Rezultati u prethodnim tabelama još jednom ukazuju da skoro sve ORLIB instance nemaju dovoljnu veličinu za adekvatno testiranje ponašanja algoritama na ogromnim instancama. Stoga su selekcioni metodi testirani i na prethodno opisanim generisanim instancama ([Filipo00], [Filipo03], [Kratic99a], [Kratic00]).

Selekcija	MO	MP	MQ	MR	MS
r. b.	112.4	181.7	269.9	423.7	879.4
(r = 2.5 to 0.7)	0.59	1.18	2.69	4.68	23.18
f. g. t. (Ftour=4.5)	76.8	131.6	205.9	347	746.9
	0.46	0.83	1.51	3.47	17.24
t. (Ntour=5)	85.2	128.6	206.2	357	732.4
	0.36	0.67	1.28	3.14	15.78
f. g. t. (Ftour=5.5)	96.4	127.9	208.6	340.6	756.1
	0.49	0.79	1.51	3.31	16.77
f. g. t. (Ftour=5.6)	83.6	131.5	192.1	343.4	723.6
	0.35	0.55	0.93	2.52	14.47
f. g. t. (Ftour=5.8)	92.2	122.8	204.4	345.7	766.2
	0.38	0.51	0.99	2.53	14.94
t. (Ntour=6)	96.7	131.3	210.4	332.4	743.8
	0.49	0.81	1.50	3.19	16.31
f. g. t. (Ftour=6.2)	83.5	122.6	207	345.6	740.5
	0.43	0.76	1.47	3.31	16.16
f. g. t. (Ftour=6.4)	87.4	130.7	200.3	340.7	721.3
	0.44	0.81	1.42	3.24	15.65

tabela 5.10. Poređenje operatora selekcije na generisanim instancama

Dobijeni rezultati se mogu lakše analizirati ako se smeste u istu, sumarnu tabelu.

Selekcija	41-74	81-104	111-134	A-C	MO	MP	MQ	MR	MS
r. b.	17.7	33.6	109.2	1328	112.4	181.7	269.9	423.7	879.4
(r = 2.5 to 0.7)	0.10	0.17	0.51	16.28	0.59	1.18	2.69	4.68	23.18
f. g. t.	10.4	34.4	145.3	1633	76.8	131.6	205.9	347	746.9
(Ftour=4.5)	0.07	0.17	0.65	16.78	0.46	0.83	1.51	3.47	17.24
t. (Ntour=5)	9.1	49.1	**128.9**	1433	85.2	128.6	206.2	357	732.4
	0.06	0.21	**0.52**	12.72	0.36	0.67	1.28	3.14	15.78
f. g. t.	9	53.5	136.6	1209	96.4	127.9	208.6	340.6	756.1
(Ftour=5.5)	0.06	0.24	0.62	12.24	0.49	0.79	1.51	3.31	16.77
f. g. t.	**8.8**	**41.3**	136.3	1694	**83.6**	131.5	**192.1**	**343.4**	**723.6**
(Ftour=5.6)	**0.05**	**0.18**	0.55	12.47	**0.35**	0.55	**0.93**	**2.52**	**14.47**
f. g. t.	8.9	46.6	151.2	2347	92.2	**122.8**	204.4	345.7	766.2
(Ftour=5.8)	0.05	0.20	0.61	16.81	0.38	**0.51**	0.99	2.53	14.94
t. (Ntour=6)	**8.8**	67.1	146	**1078**	96.7	131.3	210.4	332.4	743.8
	0.05	0.30	0.65	**10.74**	0.49	0.81	1.50	3.19	16.31
f. g. t.	**8.5**	77.1	164.4	1890	83.5	122.6	207	345.6	740.5
(Ftour=6.2)	**0.05**	0.34	0.73	18.22	0.43	0.76	1.47	3.31	16.16
f. g. t.	**8.8**	87.1	148.6	2076	87.4	130.7	200.3	340.7	721.3
(Ftour=6.4)	**0.05**	0.38	0.66	20.06	0.44	0.81	1.42	3.24	15.65

tabela 5.11. Sumarno poređenje operatora selekcije na ORLIB instancama 41-74, 81-104, 111-134, A-C i na generisanim instancama MO, MP, MQ, MR, MS

Prvi, generalni zaključak je da FGTS na SPLP daje bolje rezultate i u odnosu na klasičnu turnirsku selekciju i u odnosu na selekciju zasnovanu na rangu. Teorijski se lako dalo pokazati da rezultati FGTS ne mogu biti gori od rezultata klasične turnirske selekcije (jer je ova specijalni slučaj FGTS). Izvršeni eksperiment (kao i drugi eksperimenti koji će biti kasnije opisani) potvrđuju da nadogradnja klasične turnirske selekcije do FGTS daje novi kvalitet, odnosno novu vrednost. Takođe je važno istaći da je pri rešavanju ovog problema FGTS prevazišla i jednu od najpopularnijih selekcija u EA – selekciju zasnovanu na rangu. Pri tome, parametri selekcije bazirane na rangu (rang najbolje jedinke, rang najgore jedinke, način opadanja ranga) koja je poređena u eksperimentu su izabrani nakon pažljivog proučavanja – oni su na SPLP-u davali bolje rezultate od ma kog drugog rang-baziranog operatora selekcije koji je isproban (videti [Filipo03]).

Iz tabele 5.11. se vidi da se u najvećem broju slučajeva najbolji rezultati (ili rezultati koji su vrlo bliski najboljim) dobijaju primenom FGTS sa željenom srednjom veličinom turnira 5.6. Prednost koju FGTS ima u odnosu na druge selekcione metode je značajna (obično 10%-20%) a na nekim (najčešće ogromnim) instancama je čak i veća.

5.5.2. Problem neograničene višestruke alokacije pozicije habova (UMAHLP)

Računarski i telekomunikacioni sistemi, DHL servisi i poštanske mreže, kao i transportni sistemi mogu da se posmatraju kao mreža habova. Svi ovi sistemi sadrže skup čvorova (lokacija) koji imaju međusobnu interakciju i koji su na zadatom rastojanju i sa zadatom cenom puta. Podskup tih čvorova u mreži, označen kao hab čvorovi, sliži kao skup tačaka konsolidacije i veze između dve lokacije. Svakom čvoru mreže dodeljuje se jedan ili više habova. Sav saobraćaj između ma koja dva čvora mreže mora biti realizovan preko specificiranih habova. Kako je cena transporta između habova niža, to konsolidacija saobraćaja kroz habove dovodi do ukupnih nižih transportnih troškova i do efikasne eksploatacije mreže.

Postoji veliki broj raznovrsnih problema lokacije habova (videti [Sohn98]), zavisno od ograničenja koja se nameću mreži habova. Na primer, broj habova može biti unapred određen, mogu se postaviti ograničenja na kapacitet ili na fiksne troškove bilo za hab čvorove, bilo za čvorove koji nisu habovi, itd.

Problemi pozicioniranja habova podrazumevaju jednu od sledeće dve alokacione sheme:

- Jednostruka alokaciona shema, gde se svakom čvoru dodeljuje tačno jedan hab čvor. U tom slučaju, sav saobraćaj u/iz tog čvora prolazi isključivo kroz dodeljeni hab.
- Višestruka alokaciona shema, koja dopušta da svaka lokacija komunicira sa više od jednog haba.

U ovom poglavlju se razmatra problem neograničene višestruke alokacije pozicije habova (eng. uncapacitated multiple allocation hub location problem – UMAHLP). Tu se ne postavljaju nikakva ograničenja na kapacitet čvorova, broj hab-čvorova nije unapred fiksiran i svaki od ne-hab čvorova može biti dodeljen većem broju habova (višestruka alokaciona shema). Saobraćaj između izvornog i odredišnog čvora može biti usmeren preko jednog ili više habova. Uspostavljanje svakog haba donosi izvesne fiksne troškove. Zadatak je da se

izabere skup habova i alociraju ne-hab čvorovi elementima tog izabranog skupa, tako da je suma transportnih troškova i fiksnih troškova minimalna.

Matematička formulacija

UMAHLP može da se formuliše na sledeći način: Uočimo skup $I=\{1,...,n\}$ koji sadrži n različitih čvorova mreže, gde svaki čvor predstavlja izvor/odredište ili potencijalnu lokaciju haba. Rastojanje između čvorova i i j je C_{ij}, a može se pretpostaviti da važi nejednakost trougla. Saobraćaj (tj. tok) od lokacije i do j je označen sa W_{ij}. Promenljive odluke y_k i x_{ijkm}, koje se koriste u formulaciji problema, definisane su na sledeći način:

- $y_k = 1$ ako je hab lociran u čvoru, 0 ako nije;
- x_{ijkm} je deo toka od W_{ij} iz čvora i koji je prikupljen u habu k, distribuisan prema habu m i prosleđen u čvor j.

Svaki put od zahtevanog do odredišnog čvora sastoji se od tri dela: prikupljanje od izvora do prvog haba, prenos između habova i, na kraju, raspodela (ili distribucija) od poslednjeg haba do odredišne lokacije. Parametri χ i δ predstavljaju jedinične troškove prikupljanja i raspodele, dok $1-\alpha$ predstavlja popust u ceni za transport između habova. Fiksna cena uspostavljanja haba k (tj. postavljanja $y_k=1$) se označava sa f_k. Problem sada glasi:

$$min \sum_{i,j,k,m} W_{ij}\left(\chi C_{ik} + \alpha C_{km} + \delta C_{mj}\right)x_{ijkm} + \sum_{k} f_k y_k$$

pri čemu je:

$\sum_{k,m} x_{ijkm} = 1$, za svako i, j

$\sum_{m} x_{ijkm} + \sum_{m,m\neq k} x_{ijkm} \leq y_k$, za svako i, j, k

$y_k \in \{0,1\}$, za svako k

$x_{ijkm} \geq 0$, za svako i, j, k, m

Metode rešavanja problema

U literaturi postoji više radova u kojima se predlažu algoritmi za rešavanje UMAHLP. Dualna metoda penjanja, zajedno sa tehnikom grananja i ograničavanja je korišćena za rešavanje ORLIB instanci (kod kojih je $n \leq 25$) problema UMAHLP u radu [Klince96]. Sličan pristup je primenjen i u radu [Mayer02], pri čemu su popravljene gornje i donje granice tj. procene. U tom radu su prezentovani rezultati izvršavanja algoritma na instancama veličine do 40 čvorova. Reformulisanjem problema u kvadratni celobrojni oblik, omogućeno je da problem postane pogodan za postupak grananja i ograničavanja (videti [Abdinn98]). Autori su predstavili rezultate izvršavanja algoritma na instancama problema koje su sami kreirali, a čija je dimenzija $n \leq 80$.

U radu [Boland04] je korišćena formulacija mešovitog celobrojnog linearnog programiranja (eng. Mixed Integer Linear Programming - MILP) za tri problema višestruke alokacije pozicija habova, uključujući UMAHLP. Za svaki od ta tri problema su razvijeni procedura preprocesiranja i stezanje uslova. Ovaj pristup je testiran na standardnim hab instancama veličine do 50 čvorova.

Još jedna ideja, prezentovana u [Canov04], je da se pri rešavanju UMAHLP razmotri dualni problem MLIP formulacije. Autori prvo formiraju heuristički metod, baziran na tehnici dualnog penjanja, koja u skoro 70% slučajeva dovodi do optimalnog rešenja na ORLIB instancama sa ne više od 120 čvorova. Heuristika je potom umetnuta u algoritam grananja i ograničavanja, što garantuje postizanje optimalnog rešenja u svim slučajevima.

Instance problema

Eksperimenti su bazirani na standardnoj ORLIB instanci [BeaslJ96] AP (eng. Australian Post) koja se obično i koristi za testiranje većih instanci ovog i sličnih problema. Podaci su dobijeni iz Poštanskog sistema Australije, i sadrže do 200 čvorova koji predstavljaju poštanske oblasti. AP instance manje veličine se dobijaju agregacijom osnovnog AP skupa podataka.

Rastojanja među gradovima zadovoljavaju nejednakost trougla, ali saobraćaj (tok) između uređenih parova izvor-odredište nije simetričan. U AP instance su uključeni i fiksni troškovi. Koeficijenti χ , δ i α koji odgovaraju prikupljanju, distribuciji i transportu između habova imaju iste vrednosti kao i u radu [Canov04].

EA implementacija

Za rešavanje problema UMAHLP pomoću EA korišćen je GANP sistem, sa sledećim karakteristikama (detaljne informacije u [Kratic05]):

- Jedinke su kodirane binarnim niskama dužine *n*. Cifra 1 na datoj poziciji genetskog koda označava da je u tu uspostavljen hab, dok cifra 0 označava da hab nije uspostavljen. Kako korisnici (tj. lokacije) mogu biti dodeljene postavljenim habovima, to se iz genetskog koda samo dobija niz y_k . S obzirom da nema ograničenja vezanih za kapacitet, to se vrednosti x_{ijkm} mogu računati tokom određivanja prilagođenosti jedinke.
- Prilagođenost jedinke, tj. kvalitet rešenja se određuje na sledeći način: kada se fiksira skup habova $\{y_k\}$, odrede se najkraći putevi korišćenjem modifikacije dobro poznatog Flojd-Varšalovog algoritma. Potom se prilagođenost jedinke računa tako što se sumiraju najkraća rastojanja pomnožena sa koeficijentima toka χ , δ i α i na tu sumu se dodaju fiksni troškovi f_k za svaki uspostavljeni hab (tamo gde je $y_k = 1$)
- Kao i u primeni na prethodni problem (SPLP) eksperimentisano je sa selekcijom baziranom na rangu, klasičnom turnirskom selekcijom i FGTS. Opet su najbolji rezultati dobijeni korišćenjem fino gradirane turnirske selekcije (FGTS) sa željenom srednjom veličinom turnira 5.4 (videti [Filipo03]).
- Što se operatora ukrštanja tiče, koristi se jednopoziciono ukrštanje sa verovatnoćom 0.85 (što znači da približno 85% parova jedinki razmeni svoj genetski materijal) ;
- Ova EA implementacija koristi modifikovan operator mutacije . Naime, tokom izvršenja EA može se dogoditi da sve jedinke u populaciji imaju isti gen na određenoj poziciji. Ovakvi geni se nazivaju zamrznutim. Ako je broj zamrznutih gena l, tada prostor pretraživanja postaje 2^l puta manji i verovatnoća preuranjene konvergencije brzo raste. Operatori selekcije i ukrštanja ne mogu promeniti vrednost ma kog zamrznutog gena, a nivo klasične mutacije je često suviše mali da bi omogućio restauraciju izgubljenih podregiona u prostoru pretraživanja. Ako se nivo mutacije znatno podigne, tada se EA pretvara u slučajnu pretragu. Stoga se nivo mutacije uvećava samo na zamrznutim genima. U ovoj EA implementaciji, nivo mutacije za zamrznute gene ($1.0/n$) je dva i po puta veći od nivoa mutacije ostalih gena ($0.4/n$).
- Veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Dve trećine populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- Inicijalna populacija se generiše na pseudoslučajan način. S obzirom da je broj habova koje treba alocirati značajno manji od ukupnog broja čvorova, to se pri generisanju inicijalne populacije verovatnoća jedinice postavi na $3.0/n$. Ovim se dobijaju „bolje" jedinke za početak EA.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se radi postavljanjem njihove prilagođenosti na 0, tako da „otpadnu" u sledećoj generaciji. Na ovaj način se čuva raznovrsnost genetskog materijala i sprečava prerana konvergencija.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. Ako su im genetski kodovi slični, to može naterati EA da se pri izvršavanju veže za lokalni optimum. Da bi se izbegle takve situacije, ograničava se broj jedinki koje imaju istu prilagođenost, a različite genetske kodove (pri rešavanju instanci ovog problema na 40).
- Kod ove implementacije izvršavanje EA se zaustavlja posle 1000 generacija ili ako se prilagođenost najbolje jedinke nije popravila tokom 500 generacija.
- Performanse EA su nadalje poboljšane keširanjem EA (videti [Kratic00]) sa kešom veličine 5000.

Eksperimentalni rezultati pri poređenju algoritama za UMAHLP

Podaci koji se prezentuju u ovom poglavlju dobijeni su u eksperimentima na AMD Athlon K7/1.33GHz računaru, sa 256 MB unutrašnje memorije.

Kolone u tabelama 5.12.-5.15. (videti [Kratic05]) sadrže sledeće podatke (po redosledu pojave sleva udesno):

- dimenziju AP instance, gde sufiks L označava lake a sufiks T teške fiksne troškove;
- optimalno rešenje Opt.reš, ako je poznato unapred, inače crticu „-“;
- najbolje EA rešenje (EA naj.), pri čemu „opt“ označava da je EA dostigao optimum.
- prosečno vreme t (u sekundama) potrebno da EA dostigne rešenje;
- prosečno ukupno vreme t_{tot} (u sekundama) potrebno za završetak EA.
- prosečan ukupan broj generacija;
- prosečno odstupanje rešenja dobijenog preko EA od optimalnog (u procentima);
- standardna devijacija odstupanja od optimuma;

EA se izvršavao 20 puta nad svakom AP instancom.

Inst	Opt.reš	EA naj.	t[s]	t_{tot}[s]	gen	jaz	σ
10L	221032.734	opt	0.003	0.113	503	0.000	0.000
10T	257558.086	opt	0.001	0.114	501	0.000	0.000
20L	230385.454	opt	0.007	0.206	504	0.000	0.000
20T	266877.485	opt	0.010	0.204	506	0.000	0.000
25L	232406.746	opt	0.015	0.313	505	0.000	0.000
25T	292032.080	opt	0.014	0.295	506	0.000	0.000
40L	237114.749	opt	0.065	0.833	517	0.000	0.000
40T	293164.836	opt	0.017	0.792	501	0.000	0.000
50L	233905.303	opt	0.072	1.434	510	0.000	0.000
50T	296024.896	opt	0.072	1.339	512	0.000	0.000
60L	225042.310	opt	0.075	2.149	506	0.000	0.000
60T	243416.450	opt	0.130	2.417	516	0.000	0.000
70L	229874.500	opt	0.309	3.691	531	0.000	0.000
70T	249602.845	opt	0.152	3.629	513	0.000	0.000
80L	225166.922	opt	0.809	5.119	565	0.000	0.000
80T	268209.406	opt	0.515	4.992	539	0.000	0.000
90L	226857.465	opt	0.368	6.693	518	0.000	0.000
90T	277417.972	opt	0.424	6.619	522	0.000	0.000
100L	235097.228	opt	1.205	8.381	561	0.000	0.000
100T	305097.949	opt	0.155	7.946	505	0.000	0.000
110L	218661.965	opt	0.557	9.695	517	0.000	0.000
110T	223891.822	opt	1.103	10.731	539	0.000	0.000
120L	222238.922	opt	0.885	12.609	524	0.000	0.000
120T	229581.755	opt	2.343	15.077	564	0.000	0.000
130L	-	223814.109	3.117	21.566	563	0.000	0.000
130T	-	230865.451	2.789	22.765	552	0.000	0.000
200L	-	230204.343	25.202	81.456	667	0.696	1.239
200T	-	268787.633	28.688	93.926	701	0.000	0.000

tabela 5.12. Rezultati EA na AP instancama sa χ =3, α=0.75 i δ=2

Kao što se može videti u tabelama 5.12.-5.15, a što je podrobno analizirano i u [Kratic05], predloženi EA brzo dostiže sva poznata optimalna rešenja ($n \le 120$) za $t \le 3.5$ sekundi. Za ostale instance velikih dimenzija, za koje optimum nije poznat, EA dovodi do rešenja za $t \le 28.7$ sekundi.

Kako se pomoću EA ne može dokazati optimalnost dobijenog rešenja, to ne postoji adekvatan kriterijum završetka, pa (kako je u tabelama 5.12.-5.15. prikazano) EA i po dobijanju rešenja nastavlja da se izvršava dodatnih $\mathbf{t_{tot}-t}$ sekundi, sve dok ne bude ispunjen kriterijum završetka.

Iako predloženi EA pristup ne verifikuje optimalnost dobijenog rešenja, on predstavlja značajan doprinos u rešavanju UMAHLP, jer omogućuje rešavanje instanci velike dimenzije, koje do tada nisu bile rešene.

Inst	Opt.reš	EA naj.	t[s]	t_{tot}[s]	gen	jaz	σ
10L	122038.940	opt	0.002	0.107	501	0.000	0.000
10T	127425.939	opt	0.005	0.109	503	0.000	0.000
20L	125309.816	opt	0.009	0.182	506	0.000	0.000
20T	129079.794	opt	0.004	0.182	501	0.000	0.000
25L	126821.800	opt	0.012	0.254	506	0.000	0.000
25T	143422.390	opt	0.016	0.254	509	0.000	0.000
40L	124994.499	opt	0.075	0.689	527	0.000	0.000
40T	140962.910	opt	0.017	0.674	501	0.000	0.000
50L	120871.926	opt	0.054	1.143	508	0.000	0.000
50T	152294.536	opt	0.024	1.114	501	0.000	0.000
60L	112991.944	opt	0.086	1.708	510	0.000	0.000
60T	124961.384	opt	0.092	2.076	511	0.000	0.000
70L	114595.951	opt	0.238	2.702	527	0.000	0.000
70T	134324.296	opt	0.168	3.162	516	0.000	0.000
80L	116505.953	opt	0.553	3.985	554	0.000	0.000
80T	138970.736	opt	0.300	4.295	523	0.000	0.000
90L	115225.601	opt	0.195	5.149	509	0.000	0.000
90T	130558.600	opt	0.428	5.958	526	0.000	0.000
100L	123822.587	opt	0.714	7.028	540	0.000	0.000
100T	143119.855	opt	0.152	7.351	505	0.000	0.000
110L	110192.705	opt	0.989	8.662	544	0.000	0.000
110T	114895.505	opt	0.642	9.162	524	0.000	0.000
120L	111758.347	opt	2.845	12.624	620	0.000	0.000
120T	118376.769	opt	1.011	11.567	531	0.000	0.000
130L	-	115286.957	1.863	18.069	543	0.000	0.000
130T	-	119538.946	0.688	17.641	511	0.000	0.000
200L	-	120377.895	15.616	69.237	625	0.000	0.000
200T	-	133716.442	9.944	67.294	573	0.000	0.000

tabela 5.13. Rezultati EA na AP instancama sa χ =1, α=0.1 i δ=1

Inst	Opt.reš	EA naj.	t[s]	t_{tot}[s]	gen	jaz	σ
10L	125591.591	opt	0.003	0.107	501	0.000	0.000
10T	127425.939	opt	0.002	0.106	503	0.000	0.000
20L	126058.465	opt	0.004	0.176	501	0.000	0.000
20T	129079.794	opt	0.005	0.184	501	0.000	0.000
25L	126900.890	opt	0.020	0.247	514	0.000	0.000
25T	143422.390	opt	0.016	0.261	509	0.000	0.000
40L	125199.814	opt	0.015	0.630	501	0.000	0.000
40T	140962.910	opt	0.016	0.686	501	0.000	0.000
50L	124917.187	opt	0.024	1.108	501	0.000	0.000
50T	152294.536	opt	0.025	1.132	501	0.000	0.000
60L	116799.121	opt	0.073	1.623	507	0.000	0.000
60T	124961.384	opt	0.090	2.124	510	0.000	0.000
70L	120503.243	opt	0.340	2.742	543	0.000	0.000
70T	135016.621	opt	0.138	3.148	512	0.000	0.000
80L	119405.594	opt	0.100	3.623	504	0.000	0.000
80T	138970.736	opt	0.249	4.374	518	0.000	0.000
90L	118611.695	opt	0.554	5.270	539	0.000	0.000
90T	130558.600	opt	0.410	6.035	524	0.000	0.000
100L	125484.484	opt	0.888	6.890	554	0.005	0.023
100T	143119.855	opt	0.161	7.485	505	0.000	0.000
110L	116255.117	opt	0.906	8.330	541	0.000	0.000
110T	121484.974	opt	1.252	9.516	553	0.000	0.000
120L	118048.051	opt	3.500	12.866	651	0.000	0.000
120T	122850.043	opt	1.274	11.522	541	0.000	0.000
130L	-	120773.444	0.922	16.714	519	0.000	0.000
130T	-	126138.979	0.561	17.268	508	0.000	0.000
200L	-	122401.965	13.764	65.355	614	0.201	0.412
200T	-	133772.797	0.637	58.482	501	0.000	0.000

tabela 5.14. Rezultati EA na AP instancama sa χ =1, α=0.5 i δ=1

Inst	Opt.reš	EA naj.	t[s]	t_{tot}[s]	gen	jaz	σ
10L	125591.591	opt	0.001	0.105	501	0.000	0.000
10T	127425.939	opt	0.001	0.108	503	0.000	0.000
20L	126058.465	opt	0.003	0.179	501	0.000	0.000
20T	129079.794	opt	0.004	0.184	501	0.000	0.000
25L	126900.890	opt	0.021	0.252	513	0.000	0.000
25T	143422.390	opt	0.017	0.266	508	0.000	0.000
40L	125199.814	opt	0.018	0.642	501	0.000	0.000
40T	140962.910	opt	0.018	0.704	501	0.000	0.000
50L	124917.187	opt	0.023	1.134	501	0.000	0.000
50T	152294.536	opt	0.025	1.154	501	0.000	0.000
60L	116799.121	opt	0.088	1.645	510	0.000	0.000
60T	124961.384	opt	0.090	2.178	509	0.000	0.000
70L	121858.663	opt	0.236	2.683	527	0.000	0.000
70T	135016.621	opt	0.194	3.295	518	0.000	0.000
80L	119405.594	opt	0.101	3.710	504	0.000	0.000
80T	138970.736	opt	0.280	4.540	520	0.000	0.000
90L	118611.695	opt	0.520	5.245	536	0.000	0.000
90T	130558.600	opt	0.323	6.085	517	0.000	0.000
100L	125484.484	opt	1.025	7.044	563	0.000	0.000
100T	143119.855	opt	0.186	7.701	506	0.000	0.000
110L	119007.810	opt	0.864	8.363	538	0.000	0.000
110T	122257.504	opt	0.309	8.712	509	0.000	0.000
120L	119561.474	opt	2.130	11.762	588	0.000	0.000
120T	122850.043	opt	1.188	11.610	537	0.000	0.000
130L	-	120773.444	0.907	16.659	519	0.000	0.000
130T	-	126138.979	0.642	17.569	510	0.000	0.000
200L	-	122401.965	20.330	71.403	675	0.050	0.225
200T	-	133772.797	0.647	59.471	501	0.000	0.000

tabela 5.15. Rezultati EA na AP instancama sa χ =1, α=0.9 i δ=1

5.5.3. Problem neograničene jednostruke alokacije pozicije habova (USAHLP)

Kao što je u prethodnom poglavlju istaknuto, uvođenje mreže habova smanjuje ukupni trošak saobraćaja i povećava stepen iskorišćenosti sistema. Problemi postavljanja habova se bave lokacijama habova i dodelama ne-hab čvorova habovima.

Postoji više vrsta problema pozicioniranja habova, zavisno od konkretnih karakteristika mreže habova.

U ovom poglavlju se razmatra problem neograničene jednostruke alokacije pozicije habova (eng. uncapacitated single allocation hub location problem – USAHLP). Tu se ne postavljaju nikakva ograničenja na kapacitet čvorova, broj hab-čvorova nije unapred fiksiran i svaki od ne-hab čvorova može biti dodeljen tačno jednom od habova (jednostruka alokaciona shema). Saobraćaj između izvornog i odredišnog čvora može biti usmeren preko većeg broja habova. Uspostavljanje svakog haba donosi izvesne fiksne troškove. Zadatak je da se izabere skup habova i alociraju ne-hab čvorovi elementima tog izabranog skupa, tako da je suma transportnih troškova i fiksnih troškova minimalna.

Matematička formulacija

Problem neograničene jednostruke alokacije pozicije habova može biti formalno zapisan na sledeći način: Uočimo skup $I=\{1,...,n\}$ koji sadrži n različitih čvorova mreže, gde svaki čvor predstavlja izvor/odredište ili potencijalnu lokaciju haba. Rastojanje između čvorova i i j je C_{ij}, a može se pretpostaviti da važi nejednakost trougla. Saobraćaj (tj. tok) od lokacije i do j je označen sa W_{ij}.

Svaki put od zahtevanog do odredišnog čvora sastoji se od tri dela: prikupljanje od izvora do prvog haba, prenos između habova i, na kraju, raspodela (ili distribucija) od poslednjeg haba do odredišne lokacije. Parametri χ i δ predstavljaju jedinične troškove prikupljanja i raspodele, dok $1-\alpha$ predstavlja popust u ceni za transport između habova.

Neka je promenljiva odluke X_{ik} definisana na sledeći način: $X_{ik}=1$ ako je čvor i pridružen habu koji je uspostavljen kod čvora k, a inače je $X_{ik}=0$.

Fiksna cena uspostavljanja haba k (tj. postavljanja y_k=1) se označava sa f_k. Problem sada glasi:

$$min \sum_i \sum_j W_{ij} \sum_k \chi C_{ik} X_{ik} + \sum_i \sum_j W_{ji} \sum_l \delta C_{jl} X_{jl} + \sum_i \sum_k X_{ik} \sum_j \sum_l X_{jl} \alpha C_{kl} W_{ij} + \sum_j X_{jj} f_j$$

pri čemu je:

$\sum_k X_{ik} = 1$, za svako i

$X_{kk} - X_{ik} \geq 0$, za svako i, k

$X_{ik} \in \{0,1\}$, za svako i, k.

Metode rešavanja problema

U literaturi je poznato nekoliko metoda za rešavanje ovog problema (videti [Aykin95], [Abdinn98], [Labbe04]).

S obzirom na složenost i težinu ovog problema, veći broj autora je prepoznao da se pri njegovom rešavanju mogu postići dobri rezultati korišćenjem evolutivno inspirisanih tehnika.

U radu [Abdinn98a] se za rešavanje USAHLP problema predlaže hibrid genetskog algoritma i tabu pretrage i analiziraju dobijeni rezultati kada je nekoliko varijanti ovakvog hibrida korišćeno pri rešavanju CAB instanci problema. Autor u radu nije dao podatke o rezultatima primene hibridnog algoritma pri rešavanju AP instanci problema.

Rad [Topcu05] predlaže primenu GA u rešavanju USAHLP problema na napredniji način – koristi se efikasnije kodiranje jedinki, bolje inicijalno postavljanje broja habova u jedinki-rešenju, napredan operator ukrštanja koji je prilagođen domenu problema itd. Međutim, i nadalje se, kao i u [Abdinn98a], koristi proporcionalna, tj. rulet selekcija kao selekcioni operator. Autori u ovom radu publikuju rezultate dobijene pri rešavanju i CAB instanci i instanci AP problema.

U radu [Cheng05] razvijena je heuristika SATLUHLP za rešavanje USAHLP problema. Ova heuristika je hibrid simulacije kaljenja (eng. simulated annealing - SA) i tabu pretrage.

Instance problema

Algoritmi su testirani na skupovima ORLIB instanci, koji su preuzeti iz [BeaslJ96].

CAB (eng. Civil Aeronautics Board) skup podataka je baziran na informacijama o civilnom putničkom saobraćaju među gradovima u USA. On sadrži 60 instanci, sa do 25 čvorova i do četiri haba. Podrazumeva se da su jedinične cene prikupljanja i raspodele jednake jedinici, dok cena transporta među habovima α uzima vrednosti između 0.2 i 1.

AP je, kao što je već opisano, izveden iz studije o poštanskom sistemu u Australiji. Najveća instanca sadrži 200 čvorova (svaki čvor predstavlja poštansku oblast), sa parametrima χ=3, α=0.75 i δ=2. Instance manje veličine mogu biti dobijene agregacijom iz većih instanci. Broj habova u testnim instancama iznosi do 20.

EA implementacija

Evolutivno inspirisana heuristička metoda, implementirana na GANP sistemu, dizajnirana je na sledeći način:

Genetski kod svake jedinke se sastoji iz dva segmenta dužine *n*.Prvi segment sadrži 1 ili 0 u zavisnosti od toga da li je na odgovarajućem čvoru uspostavljen hab, ili ne. Drugi segment, čiji su elementi brojevi iz skupa {0, 1, ..., *n*-1} koji sadrži habovima koji su pridruženi ne-hab čvorovima. Ako je ne-hab čvor pridružen najbližem habu, na odgovarajućoj poziciji se smešta nula, ako je pridružen habu koji je dalji od najbližeg ali bliži od ostalih smešta se jedinica, itd.

- Prilagođenost jedinke se određuje na sledeći način: Iz prvog segmenta genetskog koda dobijaju se indeksi uspostavljenih habova, (ako se indeks uspostavljenog haba ponovi, bira se sledeći elemenat koji se do tada nije pojavio ukoliko postoji – a ako nema takvog biraće se prethodni indeks koji dotada nije bio uzet.

  Po dobijanju skupa uspostavljenih habova se, za svaki ne-hab čvor, niz uspostavljenih habova sortira u neopadajući poredak po rastojanju do tog ne-hab čvora. Iz drugog segmenta genetskog koda se potom uzima element koji odgovara tekućem ne-hab čvoru. Ako taj elemenat ima vrednost *r*, ne-hab čvoru se alocira *r*-ti hab iz prethodno sortiranog niza (r = 0, 1, ..., n-1). Ovo sortiranje uspostavljenih habova se izvršava za svaku jedinku u svakoj generaciji, što zahteva dodatni rad procesora, ali ovaj dodatni rad ne utiče bitno na vreme izvršavanja algoritma.

Kad je sve ovo određeno, jedinka se evaluira prostim sumiranjem rastojanja izvor-hab, hab-hab i hab-odredište pomnoženih sa saobraćajem i sa odgovarajućim parametrima χ, α i δ.

- EA koristi FGTS kao selekcioni operator. I u ovoj EA implementaciji se pokazalo najboljim da vrednost parametra selekcije F_{tour} je postavljena na 5.4.
- Po izvršenju selekcije primenjuje se (na početku rada opisano) jednopoziciono ukrštanje. Verovatnoća ukrštanja je 085, što znači da će oko 85% jedinki učestvovati u proizvodnji potomaka, dok u oko 15% slučajeva neće biti ukrštanja i potomci će biti identični roditeljima.
- Evolutivni metod koristi prostu mutaciju – pseudoslučajno se menja po jedan bit u oba segmenta genetskog koda. Nivoi mutacije se razlikuju među segmentima u genetskom kodu:
    - geni u prvom segmentu mutiraju sa verovatnoćom $0.6/n$;
    - za svaki gen u drugom segmentu, prvi bit se menja sa verovatnoćom $0.3/n$. Bitovi koji slede u tom genu mutiraju na dvostruko nižem nivou nego što je to slučaj sa bitovima koji neposredno prethode (0.15/n , 0.075/n , 0.0375/n , 0.01875/n , ...).

  Tokom izvršenja EA može se dogoditi da sve jedinke u populaciji imaju isti gen na određenoj poziciji. Ovakvi geni se nazivaju zamrznutim. Ako je broj zamrznutih gena l, tada prostor pretraživanja postaje 2^l puta manji i verovatnoća preuranjene konvergencije brzo raste. Operatori selekcije i ukrštanja ne mogu promeniti vrednost ma kog zamrznutog gena, a nivo klasične mutacije je često suviše mali da bi omogućio restauraciju izgubljenih podregiona u prostoru pretraživanja. Ako se nivo mutacije znatno podigne, tada se EA pretvara u slučajnu pretragu. Stoga se nivo mutacije uvećava samo na zamrznutim genima. U ovoj implementaciji, u prvom segmentu je nivo mutacije za zamrznute gene ($1.5/n$) dva i po puta veći od nivoa mutacije ostalih gena ($0.6/n$). Kod mutacije u drugom segmentu genetskog koda, svaki zamrznuti gen se menja sa 1.5 puta većom učestalošću (0.45/n , 0.225/n , 0.1175/n , 0.055875/n , ...). U drugom segmentu se postavljanju niži nivoi mutacije zato što je važno da taj segment genetskog koda sadrži mnogo nula. (nula odgovara habu koji je najbliži tom konkretnom ne-hab čvoru).
- Prvi segment inicijalnog genetskog koda jedinke se generiše pseudoslučajno. S obzirom da, pri alociranju habova ne-hab čvorovima, treba favorizovati „bliže" habove, poželjno je da drugi segment genetskog koda sadrži mnogo nula. Zato se verovatnoća generisanja bitova sa vrednošću 1 postavlja na $1.0/n$ za svaki prvi bit u svakom genu. Bitovi koji slede će imati dvostruko manju verovatnoću da budu jednaki 1 u odnosu na bit-prethodnik (dakle, verovatnoće su $\frac{1}{2n}, \frac{1}{4n}, \frac{1}{8n}, \ldots$ respektivno).
- Veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Sto jedinki iz populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se radi postavljanjem prilagođenosti duplikata na nulu, tako da ne budu izabrani u sledećoj generaciji. Na ovaj način se čuva raznovrsnost genetskog materijala i sprečava prerana konvergencija.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. Ako su im genetski kodovi slični, to može naterati EA da se pri izvršavanju veže za lokalni optimum. Da bi se izbegle takve situacije, ograničava se broj jedinki koje imaju istu prilagođenost, a različite genetske kodove (kod ove implementacije na 40).
- Kod ove implementacije izvršavanje EA se zaustavlja posle 1000 generacija ako se rešavaju veće instance ili posle 500 generacija ako se rešavaju male instance. Algoritam se zaustavlja i ako se prilagođenost najbolje jedinke nije popravila tokom 200.
- Performanse EA su nadalje poboljšane keširanjem EA sa kešom veličine 5000.
- Način kodiranja, te mehanizmi inicijalizacije, selekcije, ukrštanja i mutacije koji su primenjeni u ovoj EA implementaciji onemogućuju pojavu nekorektnih jedinki, pa ne treba preduzimati posebne mere za popravljanje takvih.

Eksperimentalni rezultati pri poređenju algoritama za USAHLP

Ovde su prikazani rezultati eksperimenta za opisani EA metod i izvršeno je poređenje sa postojećim metodama.

Rezultati su dobijeni na računari sa AMD Sempron 2.4+ procesorom, na 1578MHz, koji ima 512 MB operativne memorije. Tokom izvođenja eksperimenata računar je radio pod na UNIX (Knoppix 3.7) operativnim sistemom. Pri prevođenju su primenjene su sve optimizacije, uključujući optimizacije za AMD procesore.

Tabele 5.16. - 5.18. prikazuju rezultate EA pristupa za CAB instance, a tabele 5.19. - 5.22. prikazuju rezultate dobijene na AP instancama. Novodizajnirani EA metod se na svakoj od instanci izvršavao 20 puta.

Prve dve, kolone tabela 5.16.-5.18. sadrže dimenzije instance (**n**, **α** i **f**). Sledeća kolona **Naj. reš.** sadrži dotad najbolje rešenje za tu instancu problema. Najbolje rešenje do kog je došao EA se nalazi u koloni **EA best** (ako se najbolje rešenje poklapa sa optimalnim, u ćeliju je upisana oznaka „best"). Kolona **t** sadrži prosečno vreme (iskazano u sekundama) da EA dostigne najbolje rešenje, dok kolona $\mathbf{t_{tot}}$ sadrži vreme (u sekundama) potrebno za okončanje izvršavanja EA. U proseku EA završava izvršavanje posle **gen** generacija.

Kvalitet dobijenog rešenja se iskazuje kao prosečno odstupanje **s.jaz** (iskazano u procentima) i računa se po formuli:

$$s.jaz = \frac{1}{20}\sum_{i=1}^{20} jaz_i$$

gde je odstupanje u *i*-tom izvršavanju dato bilo u odnosu na optimalno rešenje *Opt.reš* kao:

$$jaz_i = \frac{reš_i - Opt.reš}{Opt.reš}100$$

bilo kao odstupanje u odnosu na najbolje nađeno rešenje *Naj. reš.* (u slučajevima kada optimalno rešenje nije poznato):

$$jaz_i = \frac{reš_i - Naj.reš}{Naj.reš}100$$

U tabelama koje slede je takođe iskazana i standardna devijacija **σ** prosečnog odstupanja, koja se računa po formuli:

$$\sigma = \sqrt{\frac{1}{20}\left(jaz_i - s.jaz\right)^2}$$

Kao što je već istaknuto, EA ne može dokazati optimalnost rešenja, pa se stoga EA izvršava tokom vremena t_{tot}-t (dok se ne zadovolji kriterijum završetka) iako je optimalno (odnosno najbolje) rešenje već dostignuto.

n	α	f	Naj. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
10	0.2	100	791.934	*best*	0.004	0.112	207	0.000	0.000
10	0.2	150	915.990	*best*	0.003	0.106	206	0.000	0.000
10	0.2	200	1015.990	*best*	0.009	0.112	219	0.000	0.000
10	0.2	250	1115.990	*best*	0.009	0.114	222	0.000	0.000
10	0.4	100	867.913	*best*	0.003	0.105	205	0.000	0.000
10	0.4	150	974.308	*best*	0.004	0.107	209	0.000	0.000
10	0.4	200	1074.308	*best*	0.008	0.112	218	0.000	0.000
10	0.4	250	1174.308	*best*	0.021	0.123	253	0.086	0.210
10	0.6	100	932.625	*best*	0.004	0.108	206	0.000	0.000
10	0.6	150	1032.625	*best*	0.011	0.113	220	0.000	0.000
10	0.6	200	1131.054	*best*	0.002	0.102	201	0.000	0.000
10	0.6	250	1181.054	*best*	0.001	0.098	201	0.000	0.000
10	0.8	100	990.943	*best*	0.011	0.114	222	0.000	0.000
10	0.8	150	1081.054	*best*	0.002	0.099	201	0.000	0.000
10	0.8	200	1131.054	*best*	0.001	0.099	201	0.000	0.000
10	0.8	250	1181.054	*best*	0.001	0.100	201	0.000	0.000
10	1	100	1031.054	*best*	0.002	0.100	201	0.000	0.000
10	1	150	1081.054	*best*	0.000	0.099	201	0.000	0.000
10	1	200	1131.054	*best*	0.001	0.099	201	0.000	0.000
10	1	250	1181.054	*best*	0.003	0.099	201	0.000	0.000
15	0.2	100	1030.072	*best*	0.038	0.190	251	0.000	0.000
15	0.2	150	1239.775	*best*	0.022	0.183	232	0.000	0.000
15	0.2	200	1381.281	*best*	0.009	0.149	212	0.000	0.000
15	0.2	250	1481.281	*best*	0.012	0.154	216	0.000	0.000
15	0.4	100	1179.712	*best*	0.028	0.191	238	0.127	0.130
15	0.4	150	1355.096	*best*	0.069	0.208	314	0.167	0.261
15	0.4	200	1462.627	*best*	0.009	0.149	213	0.000	0.000
15	0.4	250	1556.664	*best*	0.002	0.136	201	0.000	0.000
15	0.6	100	1309.927	*best*	0.026	0.183	241	0.243	0.554
15	0.6	150	1443.972	*best*	0.020	0.158	234	0.220	0.390
15	0.6	200	1506.664	*best*	0.002	0.139	201	0.000	0.000
15	0.6	250	1556.664	*best*	0.000	0.136	201	0.000	0.000
15	0.8	100	1390.769	*best*	0.001	0.143	201	0.000	0.000
15	0.8	150	1456.664	*best*	0.001	0.137	201	0.000	0.000
15	0.8	200	1506.664	*best*	0.000	0.137	201	0.000	0.000
15	0.8	250	1556.664	*best*	0.002	0.136	201	0.000	0.000
15	1	100	1406.664	*best*	0.002	0.138	201	0.000	0.000
15	1	150	1456.664	*best*	0.002	0.133	201	0.000	0.000
15	1	200	1506.664	*best*	0.001	0.133	201	0.000	0.000
15	1	250	1556.664	*best*	0.001	0.134	201	0.000	0.000

tabela 5.16. Rezultati EA na CAB instancama manjih dimenzija

n	α	f	Naj. reš.	GA best	t (sec)	t_{tot} (sec)	gen	s.jaz (%)	σ (%)
20	0.2	100	967.744	*best*	0.055	0.280	251	0.102	0.314
20	0.2	150	1174.538	*best*	0.045	0.267	243	0.053	0.108
20	0.2	200	1324.538	*best*	0.049	0.243	251	0.000	0.000
20	0.2	250	1474.538	*best*	0.032	0.232	235	0.403	0.429
20	0.4	100	1127.099	*best*	0.060	0.292	254	0.000	0.000
20	0.4	150	1297.767	*best*	0.038	0.235	240	0.000	0.000
20	0.4	200	1442.566	*best*	0.033	0.234	236	0.070	0.315
20	0.4	250	1542.566	*best*	0.039	0.236	245	0.473	0.756
20	0.6	100	1269.157	*best*	0.068	0.298	270	0.046	0.205
20	0.6	150	1406.044	*best*	0.022	0.223	222	0.075	0.334
20	0.6	200	1506.044	*best*	0.030	0.220	232	0.592	0.496
20	0.6	250	1570.916	*best*	0.003	0.189	205	0.000	0.000
20	0.8	100	1369.523	*best*	0.013	0.214	213	0.000	0.000
20	0.8	150	1469.523	*best*	0.010	0.200	211	0.057	0.048
20	0.8	200	1520.916	*best*	0.007	0.193	206	0.000	0.000
20	0.8	250	1570.916	*best*	0.008	0.193	205	0.000	0.000
20	1	100	1410.077	*best*	0.045	0.241	252	0.231	0.361
20	1	150	1470.916	*best*	0.006	0.194	205	0.000	0.000
20	1	200	1520.916	*best*	0.004	0.190	204	0.000	0.000
20	1	250	1570.916	*best*	0.004	0.190	204	0.000	0.000

tabela 5.17. Rezultati EA na CAB instancama dimenzije 20

n	α	f	Naj. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
25	0.2	100	1029.634	*best*	0.120	0.418	287	0.060	0.269
25	0.2	150	1217.349	*best*	0.071	0.317	256	0.025	0.062
25	0.2	200	1367.349	*best*	0.117	0.362	298	0.130	0.548
25	0.2	250	1500.907	*best*	0.047	0.300	238	0.140	0.443
25	0.4	100	1187.515	*best*	0.086	0.383	263	0.000	0.000
25	0.4	150	1351.699	*best*	0.111	0.356	292	0.037	0.165
25	0.4	200	1501.629	*best*	0.081	0.334	269	0.000	0.001
25	0.4	250	1601.629	*best*	0.070	0.323	259	0.210	0.432
25	0.6	100	1333.565	*best*	0.065	0.321	250	0.046	0.090
25	0.6	150	1483.565	*best*	0.093	0.343	279	0.329	0.518
25	0.6	200	1601.206	*best*	0.127	0.382	307	0.000	0.000
25	0.6	250	1701.206	*best*	0.094	0.350	282	0.029	0.130
25	0.8	100	1458.831	*best*	0.112	0.358	293	0.019	0.029
25	0.8	150	1594.085	*best*	0.130	0.385	310	0.151	0.405
25	0.8	200	1690.576	*best*	0.003	0.237	201	0.000	0.000
25	0.8	250	1740.576	*best*	0.003	0.237	201	0.000	0.000
25	1	100	1556.630	*best*	0.086	0.341	273	0.341	0.082
25	1	150	1640.576	*best*	0.002	0.236	201	0.000	0.000
25	1	200	1690.576	*best*	0.002	0.238	201	0.000	0.000
25	1	250	1740.576	*best*	0.003	0.237	201	0.000	0.000

tabela 5.18. Rezultati EA na CAB instancama dimenzije 25

na ovim instancama su i opisana EA implementacija i implementacije opisane u [Topcu05] i [Cheng05] dostigle najbolje rezultate, sa vremenima izvršavanja koja se veoma malo razlikuju. Na instancama CAB problema izvršavanje traje suviše kratko, pa ne mogu jasno da se sagledaju vrednosti.

Na većim i težim instancama, instancama AP problema, gde rešavanje duže traje, javlja se sledeći problem: dobijeni rezultati koji su opisani u radu [Topcu05] se razlikuju od rezultata koje je dala EA implementacija. Upitan u ličnoj, direktnoj komunikacijo o uzrocima tih neslaganja, Topcuoglu je pretpostavio da je razlog za to akumulacija greške pri zaokruživanju, jer je on sve rezultate zaokruživao na tri decimale. Ni rezultati publikovani u [Cheng05] se ne poklapaju u potpunosti sa dobijenim rezultatima, mada stepen odstupanja nije tako veliki kao što je to slučaj kod [Topcu05].

Ipak, da bi se u potpunosti otklonile dileme, pristupilo se rešavanju potproblema USAHLP problema (nazvanog HCSAP, koji je dobijen fiksiranjem habova – on samo određuje raspored čvor-hab) na AP instancama klasičnim BnB algoritmom. Kao što nam je poznato, BnB algoritam garantuje dobijanje tačnog rešenja. Rezultati EA implementacije i BnB pri rešavanju su prikazani u tabelama 5.19. – 5.22. Prvih sedam

kolona u ovim tabelama imaju isto značenje kao i kolone u tabelama 5.16. – 5.18. Kolona **Hubs** daje informaciju o broju uspostavljenih habova, kolona **HCSAP BnB** sadrži informacije o dobijenom tačnom rešenju, a sledeća kolona informacije o vremenu potrošenom radi dobijanja rešenja. Pri tome, skraćenica n.t. (eng. not tested) označava da instanca problema nije rešavana, n.n. (eng. not necessary) označava da rešavanje nije bilo neophodno jer je dobijeni skup habova jednočlan (pa je jasno kom se habu pridružuju čvorovi), n.f. (eng. not finished) označava da izvršavanje algoritma nije završeno, a time označava da je vreme izvršavanja trajalo duže od jednog dana. Kolona **Topcu** sadrži rezultate do kojih je došao Topcuoglu, a **Chen** rezultate do kojih je došao Chen.

Inst.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]	Hubs	HCSAP BnB	t[s]	Topcu	Chen
10L	224250.055	0.009	0.101	217	0.000	0.000	3,4,7	224250.055	0.04	224249.82	224250.06
10T	263399.943	0.020	0.113	249	0.000	0.000	4,5,10	**263399.943**	0.07	**263402.13**	263399.95
20L	234690.963	0.016	0.206	216	0.000	0.000	7,14	234690.963	0.11	234690.11	234690.96
20T	271128.176	0.029	0.213	229	0.909	1.271	7,19	**271128.176**	0.11	**263402.13**	271128.18
25L	236650.627	0.035	0.275	228	0.000	0.000	8,18	**236650.627**	0.20	**236649.69**	236650.63
25T	295667.835	0.004	0.233	201	0.000	0.000	13	n.t.	n.n.	**295670.39**	295667.84
40L	240986.233	0.101	0.554	244	0.221	0.235	14,28	**240986.233**	1.90	**240985.51**	**240986.24**
40T	293164.836	0.069	0.500	231	0.000	0.000	19	n.t.	n.n.	**293163.38**	293164.83
50L	237421.992	0.298	0.904	298	0.327	0.813	15,36	**237421.992**	4.16	**237420.69**	237421.99
50T	**300420.993**	0.008	0.592	201	0.000	0.000	24	n.t.	n.n.	**300420.87**	**300420.98**
60L	228007.900	0.415	1.205	306	0.546	0.919	18,41	228007.900	7.93	n.t.	n.t.
60T	246285.034	0.231	1.016	258	0.356	1.593	19,41	246285.034	7.54	n.t.	n.t.
70L	233154.289	0.451	1.489	286	0.000	0.000	19,52	233154.289	9.84	n.t.	n.t.
70T	252882.633	0.360	1.397	269	0.000	0.000	19,52	252882.633	9.88	n.t.	n.t.
80L	229240.373	1.143	2.397	383	1.143	1.286	22,55	229240.373	21.34	n.t.	n.t.
80T	274921.572	0.633	1.818	300	0.249	0.765	5,41,52	274921.572	4455	n.t.	n.t.
90L	231236.235	0.919	2.463	319	0.841	0.865	26,82	231236.235	87.23	n.t.	n.t.
90T	280755.459	0.437	1.934	257	0.133	0.395	5,41	280755.459	16.64	n.t.	n.t.
100L	238016.277	1.382	3.221	349	0.381	0.757	29,73	**238016.277**	69.69	**238017.53**	**238015.38**
100T	**305097.949**	0.365	2.180	239	0.000	0.000	52	**n.t.**	n.n.	**305101.07**	**305096.76**
110L	222704.770	3.025	5.205	478	1.430	1.611	32,77	222704.770	7 045	n.t.	n.t.
110T	227934.627	2.604	4.761	438	4.846	5.774	32,77	227934.627	6 718	n.t.	n.t.
120L	225801.362	2.304	4.775	384	0.392	0.778	32,85	225801.362	2 896	n.t.	n.t.
120T	232460.818	3.440	5.913	475	1.741	2.972	32,85	232460.818	2 934	n.t.	n.t.
130L	227884.626	3.563	6.661	428	1.098	1.037	36,88	n.f.	time	n.t.	n.t.
130T	234935.968	3.108	6.181	399	0.398	0.459	36,88	n.f.	time	n.t.	n.t.
200L	**233802.976**	11.521	19.630	482	0.398	0.815	43,148	**n.f.**	time	**228944.77**	**228944.18**
200T	**272188.113**	10.981	19.221	463	0.326	0.215	54,122	n.f.	time	**233537.93**	**233537.33**

tabela 5.19. Rezultati poređenja EA sa BnB na AP instancama za χ=3, α=0.75 i δ=2

U prethodnoj tabeli su podebljane sve vrednosti, gde su različiti algoritmi doveli do različitih rezultata, a gde se razlika teško može objasniti samo greškom zaokruživanja. Postoji još mogućnost da je razlog za toliko neslaganje dobijenih rešenja različitost nekih od preuzetih AP instanci (ili pogrešna agregacija). S obzirom na ovako veliku razliku u kvalitetu dobijenih rešenja, nema mnogo smisla detaljno porediti vremena izvršavanja ovih heurističkih algoritama.

U radovima [Topcu05] i [Cheng05] su prikazani rezultati samo za AP instance gde je χ=3, α=0.75 i δ=2,tako da samo tabela 5.19. sadrži podatke gde se direktno porede tri algoritma za rešavanje. Međutim, dobro je proveriti i kako se EA implementacija ponaša za druge vrednosti χ, α i δ. Ti podaci su prikazani u tabelama 5.20. – 5.22.

Inst.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]	Hubs	HCSAP BnB	t[s]
10L	122038.940	0.002	0.091	205	0.000	0.000	4,7	122038.940	0.02
10T	127425.939	0.001	0.088	201	0.000	0.000	5	n.t.	n.n.
20L	125408.048	0.013	0.194	216	0.130	0.230	7,14	125408.048	0.10
20T	129079.794	0.004	0.175	204	0.000	0.000	10	n.t.	n.n.
25L	126900.890	0.004	0.229	201	0.000	0.000	13	n.t.	n.n.
25T	143422.390	0.002	0.220	201	0.000	0.000	13	n.t.	n.n.
40L	125199.814	0.087	0.529	238	0.017	0.013	19	n.t.	n.n.
40T	140962.910	0.047	0.465	221	0.236	1.055	19	n.t.	n.n.
50L	121052.452	0.165	0.765	254	0.452	0.989	17,48	121052.452	3.62
50T	152294.536	0.007	0.580	201	0.000	0.000	24	n.t.	n.n.
60L	113244.343	0.298	1.083	276	0.054	0.244	18,41	113244.343	8.46
60T	124961.384	0.054	0.804	213	0.000	0.000	41	n.t.	n.n.
70L	114759.092	0.435	1.472	284	0.000	0.000	19,52	114759.092	9.99
70T	134487.436	0.452	1.485	288	0.000	0.000	19,52	134487.436	9.99
80L	116712.263	0.642	1.891	304	0.309	0.595	5,55	116712.263	7.75
80T	138970.736	0.130	1.331	220	1.213	1.524	41	n.t.	n.n.
90L	115453.910	0.761	2.284	300	0.113	0.316	5,82	115453.910	11.49
90T	130558.600	0.113	1.595	214	2.375	7.309	41	n.t.	n.n.
100L	123996.531	0.569	2.390	262	0.136	0.377	29,73	123996.531	30.80
100T	143119.855	0.372	2.183	240	0.000	0.000	52	n.t.	n.n.
110L	110501.046	2.256	4.418	408	1.738	2.445	32,77	110501.046	42.42
110T	115033.408	1.163	3.324	306	2.066	2.032	11,77	115033.408	38.43
120L	111884.028	1.713	4.181	338	2.586	2.324	32,85	111884.028	82.70
120T	118543.483	1.802	4.260	345	1.551	2.529	32,85	118543.483	82.69
130L	115604.360	2.096	5.174	334	0.226	0.522	36,88	n.f.	time
130T	119716.859	1.677	4.757	307	1.166	2.435	11,116	n.f.	time
200L	120500.406	5.992	14.149	344	0.930	0.863	43,148	n.f.	time
200T	133772.797	0.126	8.069	201	0.000	0.000	120	n.f.	time

tabela 5.20. Rezultati poređenja EA sa BnB na AP instancama za χ=1, α=0.1 i δ=1

Inst.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]	Hubs	HCSAP BnB	t[s]
10L	125591.591	0.002	0.080	201	0.000	0.000	7	n.t.	n.n.
10T	127425.939	0.001	0.081	201	0.000	0.000	5	n.t.	n.n.
20L	126058.465	0.013	0.176	211	0.000	0.000	10	n.t.	n.n.
20T	129079.794	0.004	0.171	204	0.000	0.000	10	n.t.	n.n.
25L	126900.890	0.003	0.216	201	0.000	0.000	13	n.t.	n.n.
25T	143422.390	0.003	0.216	201	0.000	0.000	13	n.t.	n.n.
40L	125199.814	0.096	0.506	246	0.156	0.697	19	n.t.	n.n.
40T	140962.910	0.089	0.501	242	0.944	1.937	19	n.t.	n.n.
50L	124917.187	0.008	0.572	201	0.000	0.000	24	n.t.	n.n.
50T	152294.536	0.009	0.573	201	0.000	0.000	24	n.t.	n.n.
60L	116799.121	0.037	0.776	208	0.000	0.000	41	n.t.	n.n.
60T	124961.384	0.041	0.787	209	0.000	0.000	41	n.t.	n.n.
70L	121858.663	0.023	0.996	203	0.000	0.000	52	n.t.	n.n.
70T	135016.621	0.026	1.010	204	0.000	0.000	52	n.t.	n.n.
80L	119405.594	0.015	1.182	201	0.000	0.000	54	n.t.	n.n.
80T	138970.736	0.267	1.444	244	0.000	0.000	41	n.t.	n.n.
90L	118611.695	0.142	1.587	218	0.000	0.000	61	n.t.	n.n.
90T	130558.600	0.059	1.529	206	0.000	0.000	41	n.t.	n.n.
100L	125484.484	0.389	2.158	242	0.098	0.155	69	n.t.	n.n.
100T	143119.855	0.358	2.163	239	0.000	0.000	52	n.t.	n.n.
110L	119666.970	1.509	3.622	340	0.595	0.700	32,77	119666.970	274 715
110T	122257.504	0.371	2.447	234	0.399	0.835	77	n.t.	n.n.
120L	119561.474	0.944	3.355	276	2.892	2.308	85	n.t.	n.n.
120T	122850.043	0.614	3.010	249	0.220	0.678	85	n.t.	n.n.
130L	120773.444	1.179	4.227	276	1.138	1.126	90	n.t.	n.n.
130T	126138.979	1.635	4.676	306	0.279	0.652	90	n.t.	n.n.
200L	122401.965	1.260	9.116	230	0.703	0.472	122	n.t.	n.n.
200T	133772.797	0.127	8.003	201	0.000	0.000	120	n.t.	n.n.

tabela 5.21. Rezultati poređenja EA sa BnB na AP instancama za χ=1, α=0.5 i δ=1

Inst.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]	Hubs	HCSAP BnB	t[s]
10L	125591.591	0.001	0.089	201	0.089	0.000	7	n.t.	n.n.
10T	127425.939	0.001	0.089	201	0.089	0.000	5	n.t.	n.n.
20L	126058.465	0.015	0.191	216	0.191	0.000	10	n.t.	n.n.
20T	129079.794	0.003	0.177	203	0.177	0.000	10	n.t.	n.n.
25L	126900.890	0.002	0.226	201	0.226	0.000	13	n.t.	n.n.
25T	143422.390	0.003	0.225	201	0.225	0.000	13	n.t.	n.n.
40L	125199.814	0.089	0.510	242	0.510	0.959	19	n.t.	n.n.
40T	140962.910	0.109	0.532	252	0.532	1.055	19	n.t.	n.n.
50L	124917.187	0.008	0.582	201	0.582	0.000	24	n.t.	n.n.
50T	152294.536	0.008	0.584	201	0.584	0.000	24	n.t.	n.n.
60L	116799.121	0.104	0.849	227	0.849	0.000	41	n.t.	n.n.
60T	124961.384	0.045	0.799	210	0.799	0.000	41	n.t.	n.n.
70L	121858.663	0.110	1.089	221	1.089	0.190	52	n.t.	n.n.
70T	135016.621	0.025	0.998	204	0.998	0.000	52	n.t.	n.n.
80L	119405.594	0.017	1.189	201	1.189	0.000	54	n.t.	n.n.
80T	138970.736	0.386	1.580	264	1.580	0.000	41	n.t.	n.n.
90L	118611.695	0.224	1.683	230	1.683	1.523	61	n.t.	n.n.
90T	130558.600	0.149	1.629	219	1.629	0.000	41	n.t.	n.n.
100L	125484.484	0.629	2.419	269	2.419	0.156	69	n.t.	n.n.
100T	143119.855	0.521	2.349	257	2.349	0.000	52	n.t.	n.n.
110L	120398.533	0.447	2.528	241	2.528	1.338	77	n.t.	n.n.
110T	122257.504	0.384	2.471	235	2.471	1.274	77	n.t.	n.n.
120L	119561.474	0.290	2.699	223	2.699	2.395	85	n.t.	n.n.
120T	122850.043	0.512	2.906	241	2.906	1.948	85	n.t.	n.n.
130L	120773.444	1.188	4.243	276	4.243	1.255	90	n.t.	n.n.
130T	126138.979	1.212	4.269	277	4.269	0.352	90	n.t.	n.n.
200L	122401.965	0.940	8.796	222	8.796	0.472	122	n.t.	n.n.
200T	133772.797	0.127	8.010	201	8.010	0.000	120	n.t.	n.n.

tabela 5.22. Rezultati poređenja EA sa BnB na AP instancama za χ=1, α=0.9 i δ=1

Iz prethodne tri tabele se lako uočava da, kad god je tačan BnB algoritam došao do rešenja, do istog rešenja je stigla i EA implementacija. Takođe, vidi se da postoji jako mnogo slučajeva kod kojih je BnB izračunavanje predugo trajalo, a EA implementacija je u dosta kratkom roku došla do rešenja.

5.5.4. Problem neograničene jednostruke alokacije p-hab medijane (USApHMP)

Prenosne mreže se koriste u transportnim i telekomunikacionim sistemima (avio prevoz, kopneni prevoz, poštanski sistemi, računarske mreže itd.). Problemi postavljanja habova se bave lokacijama habova i dodelama ne-hab čvorova habovima.

Postoji više vrsta problema pozicioniranja habova, zavisno od konkretnih karakteristika mreže habova. Ako je broj habova fiksiran na p, tada se radi o p-hab lokacijskim problemima. Svaki ne-hab čvor može biti alociran tačno jednom habu (tada se radi o jednostrukoj alokacionoj shemi) ili većem broju habova (pa se radi o višestrukoj alokacionoj shemi). Kapacitivne verzije hab problema mogu uključiti različite vrste ograničenja kapaciteta u mreži. U analizu se mogu uključiti i fiksni troškovi za uspostavljanje haba.

Ovo poglavlje razmatra problem neograničene jednostruke alokacije p-hab medijane (eng. Uncapacitated Single Allocation p-hub Median Problem - USApHMP). Cilj je da se minimizuje cena celokupnog saobraćaja kroz mrežu, pod sledećim pretpostavkama:

- broj habova je unapred određen i iznosi p ;
- nema ograničenja vezanih za kapacitet, niti fiksnih troškova;
- svaki izvor/odredište čvor (odnosno ne-hab čvor) je dodeljen tačno jednom habu;
- nije dopušten direktan saobraćaj između ne-hab čvorova.

Matematička formulacija

Problem neograničene jednostruke alokacije p-hab medijane može biti formalno zapisan na sledeći način: Neka $H_{ij} \in \{0, 1\}$ ima vrednost 1 ako je čvor i alociran habu j , a 0 inače. Uslov $H_{kk} = 1$ ukazuje da je čvor k hab. Neka je C_{ij} rastojanje između čvorova i i j , i neka W_{ij} predstavlja količinu saobraćaja od čvora i do čvora

j. Parametri χ , α i δ, isto kao u prethodnom poglavlju, odslikavaju jediničnu cenu za prikupljanje (saobraćaj izvor-hab), prenos (saobraćaj hab-hab) i raspodelu (hub-destination) respektivno. Generalno, α se koristi kao faktor popusta koji obezbeđuje smanjenu cenu po jedinici u saobraćaju između habova, pa treba da bude $\alpha < \chi$ and $\alpha < \delta$.

Koristeći upravo definisane oznake, problem se može zapisati na sledeći način:

$$min \sum_{i,j,k,l} W_{ij} \left(\chi C_{ik} H_{ik} + \alpha C_{kl} H_{kl} + \delta C_{lj} H_{lj} \right)$$

uz uslov:

$$\sum_{k=1}^{n} H_{kk} = p$$

$$\sum_{k=1}^{n} H_{ik} = 1 \quad \text{za sve } i = 1, \ldots, n$$

$H_{ik} \leq H_{kk}$ za sve $i, k = 1, \ldots, n$

$H_{ik} \in \{0,1\}$ za sve $i, k = 1, \ldots, n$

Dakle, treba minimizirati sumu troškova saobraćaja izvor-hab, hab-hab i hab-odredište, koji su pomnoženi sa faktorima χ, α i δ respektivno. Istovremeno, ostali uslovi ukazuju da mora da bude uspostavljeno tačno *p* habova, da bude realizovana jednostruka alokaciona shema i da nema direktnog saobraćaja između čvorova koji nisu habovi.

Metode rešavanja problema

U literaturi je poznato nekoliko metoda za rešavanje ovog problema. Klincewicz je u [Klince91] koristio nekoliko heuristika baziranih na lokalnom pretraživanju susedstva (eng. local neighbourhood search) i na klasterizaciji čvorova. Neke manje instance ovog hab-problema su rešavane pomoću neuralnih mreža (videti [Doming03]). Primena tabu pretrage za rešavanje ovog problema je predstavljena u radu [Skorin94].

U radu [Ernst98] je izvedena nova MILP formulacija problema, sa manjim brojem promenljivih i uslova i autori su pristupili rešavanju tako reformulisanog problema. Oni su koristili simulaciju kaljenja za određivanje gornje granice kod LP baziranog metoda grananja i ograničavanja. Prikazani su rezultati izvršavanja takvog algoritma na standardnim ORLIB instancama sa do 200 čvorova i pokazano je da su optimalna rešenja pronađena (uz nekoliko izuzetaka) za $n \leq 50$.

U radu [Perez04] je primenjen metod ponovnog povezivanja puteva (eng. path relinking method – PR) za rešavanje USApHMP. PR je evolutivna metaheuristika koja koristi prostore susedstva kao osnovu za rekombinaciju rešenja. Predloženi PR metod je testiran na hab instancama sa $n \leq 50$ čvorova i upoređen sa popravljenom verzijom metoda tabu pretrage TABUHUB, koji je opisan u [Ernst05].

Instance problema

Algoritmi su testirani na dva skupa ORLIB instanci, preuzeta iz [BeaslJ96].

CAB sadrži 60 instanci, sa do 25 čvorova i do četiri haba. Podrazumeva se da su jedinične cene prikupljanja i raspodele jednake jedinici, dok cena transporta među habovima α uzima vrednosti između 0.2 i 1.

U AP skupu instanci najveća instanca sadrži 200 čvorova, sa parametrima χ=3, α=0.75 i δ=2. Instance manje veličine mogu biti dobijene agregacijom iz većih instanci. Broj habova u testnim instancama iznosi do 20.

EA implementacija - GAHUB1 heuristička metoda

GAHUB1 heuristička metoda, implementirana na GANP sistemu, detaljno opisana u [Kratic06a], realizovana je na sledeći način:

- Genetski kod svake jedinke se sastoji iz dva segmenta. Prvi segment je niz brojeva dužine *p* u kom elementi niza odgovaraju rednim brojevima ustanovljenih habova. Elementi tog segmenta su brojevi iz skupa {0, 1, ..., *n*-1} koji označavaju redne brojeve čvorova u mreži. Dugi segment sadrži tačno *n-p* elemenata. Svaki elemenat u tom segmentu odgovara jednom ne-hab čvoru i sadrži indeks uspostavljenog haba koji je dodeljen baš tom čvoru. Za konkretni ne-hab čvor se kreira niz uspostavljenih habova sortiran u neopadajući poredak po rastojanju do tog ne-hab čvora. Habovi se dodeljuju sami sebi.

U slučajevima kada je $\alpha < \chi$ i $\alpha < \delta$ optimalno rešenje obično ne uključuje alokaciju svakog ne-hab čvora svom najbližem habu. Indeksi habova koji su bliži ne-hab čvorovima se često pojavljuju u optimalnom rešenju, a indeksi udaljenih habova se retko javljaju. Zbog toga se pretraga usmerava prema „bližim" habovima, dok se „udaljeni" habovi razmatraju sa malom verovatnoćom. Sortiranje niza habova u neopadajući poredak po rastojanju od ne-hab čvora obezbeđuje se da bliži habovi imaju viši prioritet u dodeli.

- Prilagođenost jedinke se određuje na sledeći način: Iz prvog segmenta genetskog koda dobijaju se indeksi uspostavljenih habova, (ako se indeks uspostavljenog haba ponovi, bira se sledeći elemenat koji se do tada nije pojavio ukoliko postoji – a ako nema takvog biraće se prethodni indeks koji dotada nije bio uzet. Kako je p manje od n, to će se uvek naći „slobodnih" indeksa čvorova koji će zameniti duplirani. Na ovaj način je obezbeđeno da tačno p različitih indeksa čvorova gde se uspostavljaju habovi bude dobijeno iz genetskog koda.

  Po dobijanju skupa uspostavljenih habova se, za svaki ne-hab čvor, niz uspostavljenih habova sortira u neopadajući poredak po rastojanju do tog ne-hab čvora. Iz drugog segmenta genetskog koda se potom uzima element koji odgovara tekućem ne-hab čvoru. Ako taj elemenat ima vrednost r, ne-hab čvoru se alocira r-ti hab iz prethodno sortiranog niza ($r = 0, 1, ..., p-1$). Ovo sortiranje uspostavljenih habova se izvršava n-p puta za svaku jedinku u svakoj generaciji. Ovo, naravno, zahtevati dodatni rad procesora, ali p je relativno malo kod USApHMP instanci na kojima je algoritam testiran ($p \leq 20$), pa ovaj dodatni rad ne utiče bitno na vreme izvršavanja algoritma.

  Kad je sve ovo određeno, jedinka se evaluira prostim sumiranjem rastojanja izvor-hab, hab-hab i hab-odredište pomnoženih sa saobraćajem i sa odgovarajućim parametrima χ, α i δ.
- GAHUB1 koristi FGTS kao selekcioni operator. U GAHUB1 implementaciji vrednost parametra selekcije F_{tour} je postavljena na 5.4.
- Po izvršenju selekcije primenjuje se operator ukrštanja, koji proizvodi dva potomka. Ovde se koristi operator dvostrukog jednopozicionog ukrštanja – na slučajan način se dva puta bira tačka ukrštanja – po jednom u svakom segmentu genetskog koda. Verovatnoća ukrštanja je 085, što znači da će oko 85% jedinki učestvovati u proizvodnji potomaka, dok u oko 15% slučajeva neće biti ukrštanja i potomci će biti identični roditeljima.
- GAHUB1 koristi dvostruku prostu mutaciju – pseudoslučajno se menja po jedan bit u oba segmenta genetskog koda. Nivoi mutacije se razlikuju među segmentima u genetskom kodu:
  - geni u prvom segmentu mutiraju sa verovatnoćom $0.7/n$;
  - za svaki gen u drugom segmentu, prvi bit se menja sa verovatnoćom $0.1/n$. Bitovi koji slede u tom genu mutiraju na dvostruko nižem nivou nego što je to slučaj sa bitovima koji neposredno prethode (0.05/n , 0.025/n , 0.0125/n , ...).

  Tokom izvršenja GAHUB1 može se dogoditi da sve jedinke u populaciji imaju isti gen na određenoj poziciji. Ovakvi geni se nazivaju zamrznutim. Ako je broj zamrznutih gena l, tada prostor pretraživanja postaje 2^l puta manji i verovatnoća preuranjene konvergencije brzo raste. Operatori selekcije i ukrštanja ne mogu promeniti vrednost ma kog zamrznutog gena, a nivo klasične mutacije je često suviše mali da bi omogućio restauraciju izgubljenih podregiona u prostoru pretraživanja. Ako se nivo mutacije znatno podigne, tada se EA pretvara u slučajnu pretragu. Stoga se nivo mutacije uvećava samo na zamrznutim genima. U GAHUB1 implementaciji, u prvom segmentu je nivo mutacije za zamrznute gene ($1.75/n$) dva i po puta veći od nivoa mutacije ostalih gena ($0.7/n$). Kod mutacije u drugom segmentu genetskog koda, svaki zamrznuti gen se menja sa 1.5 puta većom učestalošću (0.15/n , 0.075/n , 0.0375/n , 0.01875/n , ...). U drugom segmentu se postavljanju niži nivoi mutacije zato što je važno da taj segment genetskog koda sadrži mnogo nula. (nula odgovara habu koji je najbliži tom konkretnom ne-hab čvoru).
- Prvi segment inicijalnog genetskog koda jedinke se generiše pseudoslučajno. S obzirom da, pri alociranju habova ne-hab čvorovima, treba favorizovati „bliže" habove, poželjno je da drugi segment genetskog koda sadrži mnogo nula. Zato se verovatnoća generisanja bitova sa vrednošću 1 postavlja na $1.0/n$ za svaki prvi bit u svakom genu. Bitovi koji slede će imati dvostruko manju

verovatnoću da budu jednaki 1 u odnosu na bit-prethodnik (dakle, verovatnoće su $\frac{1}{2n}, \frac{1}{4n}, \frac{1}{8n}, \ldots$ respektivno).

- Veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Sto jedinki iz populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se radi postavljanjem prilagođenosti duplikata na nulu, tako da ne budu izabrani u sledećoj generaciji. Na ovaj način se čuva raznovrsnost genetskog materijala i sprečava prerana konvergencija.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. Ako su im genetski kodovi slični, to može naterati EA da se pri izvršavanju veže za lokalni optimum. Da bi se izbegle takve situacije, ograničava se broj jedinki koje imaju istu prilagođenost, a različite genetske kodove (kod GAHUB1 implementacije na 40).
- Kod ove implementacije izvršavanje EA se zaustavlja posle 5000 generacija ako se rešavaju veće instance ili posle 500 generacija ako se rešavaju male instance. Algoritam se zaustavlja i ako se prilagođenost najbolje jedinke nije popravila tokom 2000 generacija kod većih instanci, odnosno posle 200 generacija pri rešavanju manjih instanci.
- Performanse EA su nadalje poboljšane keširanjem EA sa kešom veličine 5000.
- Način kodiranja, te mehanizmi inicijalizacije, selekcije, ukrštanja i mutacije koji su primenjeni u implementaciji GAHUB1 onemogućuju pojavu nekorektnih jedinki, pa ne treba preduzimati posebne mere za popravljanje takvih.

EA implementacija - GAHUB2 heuristička metoda

GAHUB2 heuristička metoda, implementirana na GANP sistemu, je takođe opisana u [Kratic06a] i realizovana na sledeći način:

- Genetski kod jedinke se sastoji od n gena, gde svaki od gena ukazuje na jedan čvor mreže. Prvi bit u svakom genu uzima vrednost 1 ako taj čvor jeste hab, inače uzima vrednost 0. Razmotrivši vrednosti ovih bitova lako se formira niz uspostavljenih habova. Ostali bitovi u genu referišu na hab koji je dodeljen tekućem čvoru. primećuje se da kod ovakve reprezentacije ne može doći do dupliranja indeksa habova, pa je samim tim nepotrebna i procedura uklanjanja duplikata koja je opisana kod GAHUB1 metode.
  Međutim, može se dogoditi da neke jedinke postanu nekorektne zato što je kod njih broj uspostavljenih habova različit od p. Budući da su operatori ukrštanja i mutacije dizajnirani tako da čuvaju korektnost jedinke, to je potrebno da se eventualne nekorektne jedinke poprave samo jednom – u okviru inicijalne populacije.
  Za svaki od ne-hab čvorova, dobijeni niz uspostavljenih habova se sortira u neopadajući poredak po rastojanju do tog ne-hab čvora.
- Prilagođenost jedinke se određuje na isti način kao što je to opisano kod GAHUB1 metode.
- Heuristički metod GAHUB2 takođe koristi FGTS kao selekcioni metod i vrednost parametra selekcije je ista kao kod GAHUB1 (F_{tour}=5.4).
- Operator ukrštanja kod GAHUB2 je realizovan tako da očuva korektnost jedinki. Tokom svog izvršavanja, operator simultano prolazi s desna ulevo kroz genetske kodove roditelja dok ne nađe poziciju i u kojoj prvi roditelj u prvom bitu gena ima jedinicu, a drugi roditelj u prvom bitu ima nulu. Po pronalaženju takve pozicije, izvrši se razmena roditeljskih gena na toj poziciji. Potom se analogan postupak uradi sleva udesno, samo što se traži prva pozicija j takva da prvi roditelj na prvom bitu gena ima nulu a drugi jedinicu – i kad se nađe takva pozicija izvrši se razmena gena. Postupak će se ponavljati sve dok je $j \leq i$. Ovakvom razmenom gena nije promenjen broj uspostavljenih habova u genetskom kodu, pa korektnost jedinke ostaje očuvana. Nivo ukrštanja i kod ovog metoda ostaje 0.85.
- Operator mutacije je dizajniran kao dvonivoovski operator sa zamrznutim genima (detaljno opisana u [Kratic06a]). .U svakom od gena, osnovni nivo mutacije je:
  - $0.4/n$ za bit na prvoj poziciji;

- $0.1/n$ za bit na drugoj poziciji. Sledeći bitovi u genu imaju svaki put dvostruko smanjen nivo mutacije ($0.05/n$, $0.025/n$, $0.0125/n$,...)

U poređenju sa normalnim nivoom mutacije, zamrznuti bitovi mutiraju sa većim nivoom:

- dva i po puta višim nivoom ($1.0/n$ umesto $0.4/n$) ako se radi o bitu na prvoj poziciji u genu;
- jedan i po put višim nivoom, inače.

Prilikom izvršenja mutacije, kod svake jedinke se prebrojava broj mutiranih jedinica i nula na prvim bitovnim pozicijama gena. U slučaju kada ta dva broja nisu jednaka, nužno je izvršiti mutaciju na još tačno jednom vodećem bitu gena u genetskom kodu. Ujednačavanjem broja mutiranih jedinica i nula na vodećim pozicijama, operator mutacije očuvava broj uspostavljenih habova, tj. očuvava korektnost mutirane jedinke.

- Svaka jedinka inicijalne populacije se generiše korišćenjem sledeće strategije.(videti [Kratic06a]):
  - prvi bit u svakom genu se postavlja pseudoslučajno i dobija vrednost 1 sa verovatnoćom p/n ;
  - drugi bit u genu biva jednak jedinici sa verovatnoćom $1.0/n$, dok sledeći bitovi uzimaju vrednost 1 svaki put sa dvostruko manjom verovatnoćom u odnosu na svog neposrednog prethodnika $(1/2n, 1/4n, 1/8n, \ldots)$.

  Iako verovatnoća p/n ukazuje da bi se trebalo formirati p jedinica u n pseudoslučajnih pokušaja (tj. p uspostavljenih lokacija među n čvorova mreže), u praksi ponekad ne biva tako. Naime, može se dogoditi da se broj gena u jedinke sa jedinicama na vodećoj poziciji (označen sa k) ne poklopi sa p, pa da jedinka ne bude korektna. Tada se jedinka koriguje dodavanjem (odnosno brisanjem) $|p-k|$ jedinica u genima, počevši od gena na kraju genetskog koda.
- Veličina populacije kod GAHUB2 je takođe 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija na isti način kao kod GAHUB1.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se i ovde radi postavljanjem prilagođenosti duplikata na nulu.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. I ovde se, radi izbegavanja takvih situacija, ograničava (na 40) broj jedinki koje imaju istu prilagođenost, a različite genetske kodove.
- Kod GAHUB2, isto kao kod GAHUB1 izvršavanje staje posle 5000 generacija (ako se rešavaju veće instance) ili posle 500 generacija (ako se rešavaju male instance), odnosno, ako se prilagođenost najbolje jedinke nije popravila tokom 2000 generacija (kod većih instanci) ili posle 200 generacija (kod manjih instanci).
- Performanse metoda su i ovde poboljšane keširanjem EA sa kešom veličine 5000.

Eksperimentalni rezultati pri poređenju algoritama za USApHMP

Ovde su prikazani rezultati eksperimenta za oba EA metoda i izvršeno je poređenje sa postojećim metodama. Sva izvršavanja algoritama su realizovana na računaru sa AMD Athlon procesorom na 1.33 GHz, koji ima 256 MB operativne memorije.

Tabele 5.23. i 5.24. prikazuju rezultate EA pristupa za CAB instance, a tabele 5.25. i 5.26. prikazuju rezultate dobijene na AP instancama. Svaki od EA metoda se na svakoj od instanci izvršavao 20 puta.

Značenje kolona u tabelama sa rezultatima je isto kao kod prethodnog problrma. Prve dve, odnosno tri, kolone tabela 5.23.-5.27. sadrže dimenzije instance (**n**, **p** i možda **α**). Sledeća kolona **Opt.reš.** sadrži optimalnu vrednost za tu instancu, ukoliko je unapred poznata (a ako nije, napisana je crtica „-“). Najbolje rešenje do kog je došao EA se nalazi u koloni **EA best** (ako se najbolje rešenje poklapa sa optimalnim, u ćeliju je upisana oznaka „opt“. Kolona **t** sadrži prosečno vreme (iskazano u sekundama) da EA dostigne najbolje rešenje, dok kolona $\mathbf{t_{tot}}$ sadrži vreme (u sekundama) potrebno za izvršavanje EA. U proseku izvršavanje EA biva završeno posle **gen** generacija.

Kvalitet dobijenog rešenja se iskazuje kao prosečno odstupanje **s.jaz** (iskazano u procentima) i računa se po formuli:

$$s.jaz = \frac{1}{20}\sum_{i=1}^{20} jaz_i$$

gse je odstupanje u *i*-tom izvršavanju dato bilo u odnosu na optimalno rešenje *Opt. reš.* kao:

$$jaz_i = \frac{reš_i - Opt.reš}{Opt.reš}100$$

bilo kao odstupanje u odnosu na najbolje nađeno rešenje *Naj. reš.* (u slučajevima kada optimalno rešenje nije poznato):

$$jaz_i = \frac{reš_i - Naj.reš}{Naj.reš}100$$

U tabelama 5.23.-5.27. je takođe iskazana i standardna devijacija **σ** prosečnog odstupanja, koja se računa po formuli:

$$\sigma = \sqrt{\frac{1}{20}\left(jaz_i - s.jaz\right)^2}$$

Kao što je već istaknuto, EA ne može dokazati optimalnost rešenja, pa se stoga EA izvršava tokom vremena t_{tot}-t (dok se ne zadovolji kriterijum završetka) iako je optimalno (odnosno najbolje) rešenje već dostignuto.

n	p	α	Opt. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
20	2	0.2	979.087	opt	0.010	0.083	213	0.000	0.000
20	2	0.4	1042.566	opt	0.010	0.082	211	0.000	0.000
20	2	0.6	1106.044	opt	0.011	0.085	214	0.000	0.000
20	2	0.8	1169.523	opt	0.009	0.082	211	0.000	0.000
20	2	1.0	1210.077	opt	0.013	0.086	224	0.000	0.000
20	3	0.2	724.538	opt	0.017	0.143	221	0.000	0.000
20	3	0.4	847.767	opt	0.016	0.144	220	0.000	0.000
20	3	0.6	970.996	opt	0.023	0.148	229	0.000	0.000
20	3	0.8	1091.050	1094.225	0.056	0.181	278	0.351	0.062
20	3	1.0	1156.072	opt	0.033	0.154	241	0.055	0.244
20	4	0.2	577.621	opt	0.032	0.180	232	0.000	0.000
20	4	0.4	727.099	opt	0.027	0.177	226	0.000	0.000
20	4	0.6	869.157	opt	0.034	0.185	236	0.000	0.000
20	4	0.8	1008.492	opt	0.082	0.227	299	0.310	0.352
20	4	1.0	1111.015	1111.798	0.106	0.249	327	0.278	0.260
25	2	0.2	1000.907	opt	0.009	0.099	212	0.000	0.000
25	2	0.4	1101.629	opt	0.014	0.100	219	0.076	0.342
25	2	0.6	1201.206	opt	0.021	0.108	230	0.000	0.000
25	2	0.8	1294.085	opt	0.032	0.122	253	0.000	0.000
25	2	1.0	1359.190	1362.156	0.015	0.107	220	0.218	0.000
25	3	0.2	767.349	opt	0.025	0.170	223	0.000	0.000
25	3	0.4	901.699	opt	0.033	0.180	233	0.000	0.000
25	3	0.6	1033.565	opt	0.045	0.192	249	0.040	0.180
25	3	0.8	1158.831	opt	0.060	0.206	264	0.032	0.144
25	3	1.0	1256.630	opt	0.060	0.208	265	0.276	0.566
25	4	0.2	629.634	opt	0.092	0.258	288	0.123	0.300
25	4	0.4	787.515	opt	0.062	0.235	257	0.044	0.198
25	4	0.6	939.206	opt	0.053	0.232	245	0.102	0.456
25	4	0.8	1087.662	opt	0.056	0.229	249	0.444	0.334
25	4	1.0	1211.232	opt	0.090	0.261	285	0.000	0.000

tabela 5.23. Rezultati GAHUB1 na CAB instancama

n	p	α	Opt. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
20	2	0.2	979.087	opt	0.013	0.097	221	0.000	0.000
20	2	0.4	1042.566	opt	0.013	0.099	218	0.000	0.000
20	2	0.6	1106.044	opt	0.011	0.097	213	0.000	0.000
20	2	0.8	1169.523	opt	0.011	0.360	214	0.000	0.000
20	2	1.0	1210.077	opt	0.018	0.104	228	0.000	0.000
20	3	0.2	724.538	opt	0.027	0.152	233	0.000	0.000
20	3	0.4	847.767	opt	0.029	0.156	238	0.000	0.000
20	3	0.6	970.996	opt	0.044	0.169	259	0.000	0.000
20	3	0.8	1091.050	opt	0.055	0.179	281	0.328	0.101
20	3	1.0	1156.072	opt	0.048	0.173	266	0.000	0.000
20	4	0.2	577.621	opt	0.028	0.158	232	0.000	0.000
20	4	0.4	727.099	opt	0.038	0.171	243	0.000	0.000
20	4	0.6	869.157	opt	0.033	0.168	236	0.000	0.000
20	4	0.8	1008.492	opt	0.057	0.192	272	0.041	0.099
20	4	1.0	1111.015	opt	0.100	0.223	332	0.076	0.042
25	2	0.2	1000.907	opt	0.009	0.117	209	0.000	0.000
25	2	0.4	1101.629	opt	0.009	0.117	208	0.000	0.000
25	2	0.6	1201.206	opt	0.029	0.567	241	0.000	0.000
25	2	0.8	1294.085	opt	0.056	0.160	286	0.139	0.285
25	2	1.0	1359.190	opt	0.023	0.132	229	0.207	0.049
25	3	0.2	767.349	opt	0.039	0.200	239	0.000	0.000
25	3	0.4	901.699	opt	0.041	0.203	242	0.000	0.000
25	3	0.6	1033.565	opt	0.071	0.230	277	0.000	0.000
25	3	0.8	1158.831	opt	0.098	0.255	311	0.065	0.199
25	3	1.0	1256.630	opt	0.124	0.370	337	0.135	0.417
25	4	0.2	629.634	opt	0.049	0.362	244	0.000	0.000
25	4	0.4	787.515	opt	0.049	0.221	244	0.000	0.000
25	4	0.6	939.206	opt	0.054	0.301	248	0.000	0.000
25	4	0.8	1087.662	opt	0.068	0.240	263	0.068	0.210
25	4	1.0	1211.232	opt	0.081	0.255	277	0.029	0.103

tabela 5.24. Rezultati GAHUB2 na CAB instancama

n	p	Opt. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
10	2	167493.065	opt	0.002	0.051	209	0.000	0.000
10	3	136008.126	-	-	-	-	-	-
10	4	112396.068	opt	0.018	0.108	233	0.000	0.000
10	5	91105.371	opt	0.033	0.160	247	0.014	0.044
20	2	172816.690	opt	0.011	0.084	215	0.000	0.000
20	3	151533.084	opt	0.020	0.141	222	0.000	0.000
20	4	135624.884	opt	0.025	0.175	224	0.000	0.000
20	5	123130.095	opt	0.058	0.267	247	0.000	0.000
25	2	175541.977	opt	0.034	0.122	258	0.000	0.000
25	3	155256.323	opt	0.038	0.185	239	0.023	0.070
25	4	139197.169	opt	0.072	0.242	267	0.035	0.024
25	5	123574.289	opt	0.033	0.281	220	0.010	0.025
40	2	177471.674	opt	0.048	0.185	241	0.000	0.000
40	3	158830.545	opt	0.089	0.337	256	0.000	0.000
40	4	143968.876	opt	0.052	0.350	224	0.037	0.164
40	5	134264.967	opt	0.177	0.573	275	0.118	0.283
50	2	178484.286	opt	0.136	0.309	312	0.000	0.000
50	3	158569.933	opt	0.126	0.424	261	0.000	0.000
50	4	143378.046	opt	0.196	0.547	284	0.222	0.629
50	5	132366.953	opt	0.229	0.733	274	0.172	0.338
100	2	-	180223.801	0.510	3.821	2141	0.173	0.661
100	3	-	160847.001	0.941	8.203	2195	0.000	0.000
100	4	-	145896.578	1.230	9.581	2241	0.018	0.037
100	5	-	136929.444	2.146	13.585	2319	0.309	0.309
100	10	-	106829.151	11.147	32.590	2927	1.597	1.150
100	15	-	90534.785	13.179	48.500	2718	1.145	0.625
100	20	-	80471.845	34.362	81.860	3378	0.874	0.540
200	2	-	182459.254	3.795	19.143	2244	0.026	0.030
200	3	-	162887.031	6.652	36.796	2316	0.851	1.080
200	4	-	147767.303	15.155	48.338	2692	0.948	1.342
200	5	-	140450.089	18.344	63.417	2701	0.493	0.646
200	10	-	110648.724	57.337	135.878	3201	1.611	1.387
200	15	-	95857.693	81.918	199.222	3241	1.309	1.007
200	20	-	86069.213	151.197	297.513	3675	1.652	1.042

tabela 5.25. Rezultati GAHUB1 na AP instancama

Kao što se može uočiti u tabelama 5.23. i 5.25. GAHUB1 brzo dostiže sva prethodno poznata optimalna rešenja, osim kod tri CAB instance. On takođe daje rezultate i na velikim AP instancama (n=100,200), što se može videti u tabeli 5.25. U tri slučaja (n=100, p=15; n=100, p=20 i n=200, p=10) GAHUB1 daje rešenja koja su bolja od najboljih dobijenih dotadašnjim metodama (SA, TABUHUB i PR).

Rezultati prikazani u tabelama 5.24 i 5.26. pokazuju da GAHUB2 ima čak i bolje performanse. On dostiže sva optimalna rešenja i na CAB i na AP instancama. Kod većih AP instanci gde nije poznato optimalno rešenje, GAHUB2 daje bolja (ili ista) rešenja kao i sve druge heuristike za rešavanje USApHMP.

n	p	Opt. reš.	EA best	t[s]	t_{tot}[s]	gen	s.jaz[%]	σ[%]
10	2	167493.065	opt	0.003	0.054	206	0.000	0.000
10	3	136008.126	opt	0.005	0.074	213	0.000	0.000
10	4	112396.068	opt	0.006	0.074	216	0.000	0.000
10	5	91105.371	opt	0.013	0.102	221	0.000	0.000
20	2	172816.690	opt	0.020	0.107	235	0.000	0.000
20	3	151533.084	opt	0.018	0.144	222	0.000	0.000
20	4	135624.884	opt	0.032	0.163	236	0.000	0.000
20	5	123130.095	opt	0.038	0.213	231	0.000	0.000
25	2	175541.977	opt	0.037	0.150	253	0.000	0.000
25	3	155256.323	opt	0.039	0.198	238	0.000	0.000
25	4	139197.169	opt	0.059	0.233	256	0.000	0.000
25	5	123574.289	opt	0.049	0.280	233	0.010	0.025
40	2	177471.674	opt	0.143	0.344	316	0.000	0.000
40	3	158830.545	opt	0.114	0.404	266	0.000	0.000
40	4	143968.876	opt	0.155	0.476	278	0.000	0.000
40	5	134264.967	opt	0.351	0.750	344	0.018	0.045
50	2	178484.286	opt	0.237	0.516	342	0.018	0.019
50	3	158569.933	opt	0.450	0.808	394	0.010	0.028
50	4	143378.046	opt	0.328	0.764	331	0.011	0.044
50	5	132366.953	opt	0.514	1.053	365	0.049	0.085
100	2	-	180223.801	4.236	13.293	2854	0.030	0.018
100	3	-	160847.001	4.271	16.640	2638	0.000	0.000
100	4	-	145896.578	3.863	17.761	2513	0.000	0.000
100	5	-	136929.444	5.769	22.108	2652	0.116	0.394
100	10	-	106469.566	18.782	40.733	3494	0.240	0.347
100	15	-	90533.523	28.853	57.453	3744	0.426	0.320
100	20	-	80270.962	43.603	79.200	4173	0.166	0.281
200	2	-	182459.254	51.870	100.722	3906	0.044	0.026
200	3	-	162887.031	46.213	111.774	3329	0.000	0.000
200	4	-	147767.303	61.439	131.161	3531	0.033	0.008
200	5	-	140175.645	89.729	169.733	4022	0.212	0.108
200	10	-	110147.657	182.210	259.142	4794	0.213	0.187
200	15	-	94496.406	217.431	313.017	4829	0.504	0.339
200	20	-	85129.343	310.625	374.751	4937	0.674	0.268

tabela 5.26. Rezultati GAHUB2 na AP instancama

n	p	Ernst SA heur		TABUHUB		Pérez PR		GAHUB1		GAHUB2	
		Naj. reš.	Alpha 200 MHz	Naj. reš.	Intel 1.4 GHz	Naj. reš.	Intel 1.4 GHz	Naj. reš.	AMD 1.33 GHz	Naj. reš.	AMD 1.33 GHz
50	5	**opt**	19.5	**opt**	1240	**opt**	204	**opt**	0.733	**opt**	1.053
100	5	**136929.44**	80.6	**136929.44**	805	**136929.44**	301	**136929.444**	13.585	**136929.444**	22.108
100	10	**106469.57**	161.9	**106469.57**	1259	**106469.57**	396	106829.151	32.590	**106469.566**	40.733
100	15	90605.10	279.8	90605.10	1480	90605.10	590	90534.785	48.500	**90533.523**	57.453
100	20	80682.71	522.7	80682.71	2330	80682.71	1008	80471.845	81.860	**80270.962**	79.200
200	5	140409.41	399.7	-	-	-	-	140450.089	63.417	**140175.645**	169.733
200	10	111088.33	776.6	-	-	-	-	110648.724	135.878	**110147.657**	259.142
200	15	95460.54	1105.3	-	-	-	-	95857.693	199.222	**94496.406**	313.017
200	20	85560.39	1555.9	-	-	-	-	86069.213	297.513	**85129.343**	374.751

tabela 5.27. Poređenje rezultata na većim AP instancama

Tabela 5.27. sadrži detaljno poređenje metoda GAHUB1 i GAHUB2 sa sledećim heurističkim metodama:

- Simulirano kaljenje (SA), predloženo u [Ernst98], testirano na DEC 3000/7000 (sa 200MHz Alpha procesorom);
- Heuristika tabu pretrage (TABUHUB), opisana u [Ernst05], testirana na računaru sa Intel XEON procesorom, koji radi na 1.4 GHz;
- Metod ponovnog povezivanja puteva (PR), predložen u [Perez04], takođe testiran na računaru sa Intel XEON procesorom, koji radi na 1.4 GH.

U tabelu su, naravno uključeni i rezultati EA metoda GAHUB1 i GAHUB2 i svi najbolji rezultati su podebljani.

Prema uporednim testovima brzine računara (eng. benchmark) SPEC_fp95 i SPEC_fp2000 koji su dostupni na sajtu www.spec.org, u poređenju sa računarom DEC 3000/700 računari Intel XEON 1.4 GHz i AMD 1.33 GHz imaju prosečno ubrzanje 6.9 i 7.3 respektivno. Uzevši te vrednosti u obzir, tj. podelivši SA vremena sa 7.3, dobijamo da je vreme izvršavanja SA približno jednako vremenima koja postižu GAHUB1 i GAHUB2. na isti način se dobija da su metodi TABUHUB i PR značajno sporiji od SA, GAHUB1 i GAHUB2.

Iz prethodno prezentovanih rezultata proizilazi da GAHUB2 ima bolje performanse od GAHUB1. To se može objasniti na sledeći način: GAHUB1 koristi reprezentaciju sa dva segmenta, a GAHUB2 sa jednim segmentom. Celobrojna reprezentacija kod GAHUB1 ukazuje da je informacija o tome da li je čvor hab ili ne, izdvojena u poseban segment i razdvojena od informacije o dodeli čvora habu. Kod binarne reprezentacije korišćene u GAHUB2, sve informacije o svakom od čvorova su grupisane zajedno u genetskom kodu, pa su gradivni blokovi kod GAHUB2 kraći nego kod GAHUB1. Stoga, objašnjenje dobijenih rezultata može biti to što hipoteza o gradivnim blokovima i teorema o shemama favorizuju kraće gradivne blokove. Potpunija argumentacija tj. objašnjenje je dato u [Kratic06a].

5.5.5. Problem dizajniranja mreže neograničenog kapaciteta (UNDP)

Problemi optimizacije na mrežama su izuzetno značajni u praksi, pogotovo u poslednje vreme. Problem dizajniranja mreže neograničenog kapaciteta (eng. 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.

Matematička formulacija

Problem dizajniranja mreže neograničenog kapaciteta može biti formulisan na sledeći način: 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:

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

uz uslove

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

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

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

gde je x_{ij}^k količina artikla $k \in C$ koja se transportuje preko grane $(i,j) \in A$. Promenljiva y_{ij} ukazuje da li je grana (i,j) uspostavljena, a promenljiva b_i^k predstavlja protok artikla k kroz čvor i:

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

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 $y_{ij} = 1$.

Metode rešavanja problema

U ovom poglavlju će biti opisane samo neke metode rešavanja problema, koje su davale rešenja relativno dobrog kvaliteta i za probleme veće dimenzije.

U radu [Magna86] primenjeno je Bender-ovo razlaganje 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. 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, pokazalo se da su nedostatci samo ublaženi, a ne i otklonjeni pa su rezultati nešto lošiji od direktnog pristupa pri metodi grananja i sečenja.

Unakrsno razlaganje (eng. cross decomposition) je takođe primenjivano za rešavanje datog problema, kao kombinacija Benderovog razlaganja i Lagranžove relaksacije. Ova metoda je posebno razvijena za rešavanje problema mešovitog celobrojnog programiranja (eng. Mixed Integer Programming - MIP) i detaljno opisana u radu [VanRo83]. Međutim, pokazalo se da iako nešto bolji od prethodne metode (Bender-ovog razlaganja), pri rešavanju UNDP ipak zadržava neke nedostatke koji karakterišu Benderova razlaganja. Pomenimo i neka unapređenja unakrsnog razlaganja: metoda unakrsnog razlaganja pomoću srednje vrednosti (eng. mean value cross decomposition) opisanu u [Holmbe91].

Jedna od značajnih heuristika za rešavanje UNDP je i dualna metoda penjanja, opisana u radu [Balakr89]. Ona vrlo brzo daje rešenje relativno dobrog kvaliteta (greška 1% - 4% od optimalnog 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 [Holmbe97a]).

Lagranževa heuristika koja je uspešno korišćena za rešavanje lokacijskih problema ([Holmbe97]), prilagođena je datom problemu i primenjena uz grananje i ograničavanje. 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.

Za rešavanje UNDP mogu se koristiti i programski paketi za rešavanje opštijeg problema mešovitog celobrojnog programiranja (eng. mixed integer programming - MIP). Takvi programski paketi su 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.

Instance problema

Pošto ne postoji biblioteka UNDP instanci dostupna preko Interneta, a pokušaji da se one dobiju ličnim kontaktima nisu bili uspešni, izvršeno je njihovo generisanje, slično generisanju kod instanci SPLP problema.

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 5.28, a postupak generisanja je sličan postupku generisanja SPLP instanci (detaljno je opisan u radu [Kratic02]). 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}. .

Grupa instanci	Dimenzija C, N, A	f	c	Veličina datoteke
MA	10×5×15	2-6	1-3	2.5 KB
MB	10×10×32	2-6	1-3	5 KB
MC	10×15×50	2-6	1-3	7.7 KB
MD	10×20×70	2-6	1-3	10.7 KB
ME	10×30×120	2-6	1-3	18.2 KB
MF	50×30×120	2-6	1-3	72 KB
MG	10×50×250	2-6	1-3	37 KB
MH	50×50×250	2-6	1-3	148 KB
MI	10×100×700	2-6	1-3	103 KB
MJ	50×100×700	2-6	1-3	412 KB

tabela 5.28. Osobine UNDP instanci

Svaka grupa sadrži pet različitih instanci, generisanih sa različitim pseudoslučajnim semenom (100, 200, 300, 400, 500). Ulazni parametri za ovaj generator su celi brojevi |C|, |N|, |A| i brojevi u pokretnom zarezu c_{min}, c_{max}, f_{min}, f_{max}. Kao što je prethodno opisano, elementi matrice transportnih troškova se biraju pseudoslučajno iz intervala $[c_{min}, c_{max}]$. Posle toga, za svaku granu (i,j) se računa suma $S_{ij} = \sum_{k \in C} c_{ij}^{\ k}$. Suma S_{ij} označava kumulativnu vrednost svih transportnih troškova za konkretnu granu. U završnoj fazi, vrši se obratno skaliranje u interval $[f_{min}, f_{max}]$, prema sledećoj formuli:

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

Ovakav način generisanja test instanci odgovara realnim situacijama, gde manji troškovi transporta c_{ij} proizvode veće fiksne troškove f_{ij} i obratno.

EA implementacija

Za rešavanje problema UNDP pomoću EA korišćen je GANP sistem, sa sledećim dizajnom (videti [Kratic02]):

- Ako se rešava problem shodno prethodnoj formulaciji, tada ima ukupno $(|C|+1) \cdot |A|$ promenljivih (y_{ij} i $x_{ij}^k : (i,j) \in A, k \in C$), što nije pogodno za primenu EA 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 promenljive 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 Dijkstrin algoritam. U ovom radu je, pri rešavanju UNDP problema, primenjivana osnovna varijanta Dijkstrinog algoritma, a u budućnosti je moguće unapređenje korišćenjem neke od modifikacija (jer uporedni pregled osnovne varijante i modifikacija ukazuje da razlika u vremenima izvršavanja postaje značajna tek kod grafova koji su bar za red veličine veći od grafova – instanci UNDP problema).
- I pri rešavanju ovog problema eksperimentisano je sa više vrsta selekcije, a najbolji rezultati dobijeni korišćenjem fino gradirane turnirske selekcije (FGTS) sa željenom srednjom veličinom turnira 5.6.
- 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:

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

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

- 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. Pri rešavanju SPLP korišćeno je generisanje početne populacije sa istom učestalošću 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 EA 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 EA 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 učestalošću 0 i 1, indukovani podgraf G' sadrži isti broj čvorova kao i polazni graf ali samo približno 50% njegovih grana (videti [Kratic02]). 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 gore pomenuta anomalija.
  Stoga se pokazalo neophodnim generisanje početne populacije sa različitim učestalostima 0 i 1 u genetskom kodu. Vrlo je efikasno implementirana varijanta sa učestalostima $p_{b0} = 1/4$, $p_{b1} = 3/4$ i $p_{b0} = 1/8$, $p_{b1} = 7/8$.
  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 sprečena mogućnost preuranjene konvergencije kod ovakvih UNDP instanci.
- GA primenjen sa elitističkom strategijom je i pri rešavanju UNDP postigao dobar kompromis između komponenti istraživanja novih regiona pretrage i eksploatacije tj. korišćenja već dobijenih dobrih jedinki. Pri tome populacija sadrži 150 jedinki od čega 2/3 populacije (100 jedinki) direktno prelazi 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 pri rešavanju SPLP-a, uklanjanje višestruke pojave jedinki u populaciji doprinosi većoj raznolikosti genetskog materijala i sprečavanju preuranjene konvergencije.

Eksperimentalni rezultati pri poređenju algoritama za UNDP

Kako nijedna konkretna implementacija za rešavanje UNDP nije bila javno izložena i dostupna, to je EA (preciznije, GANP implementacija EA) poređena sa opštim paketima za rešavanje mešovitog celobrojnog programiranja MIP. Instance UNDP problema su pretvorene u MIP format i rešavani korišćenjem programa OSL (koji je napravio IBM) i CPLEX (verzija 6.5.3). Pri poređenju se pojavio problem i u tome što OSL ne izveštava o vremenu izvršavanja.

Sva izvršavanja u ovom eksperimentu vezana za OSL i GANP su realizovana na PC kompatibilnom Pentium III računaru sa AMD-ovim procesorom na 750MHz i sa 1 GB RAM-a (videti [Kratic02]). Kako je sam evolutivni proces nedeterministički, svaka instanca problema je izvršena 20 puta (osim grupe MJ, gde su sve instance izvršavane tačno jednom).

Tabele 5.29. i 5.30. sadrže dobijene rezultate. Ove tabele sadrže sledeće kolone:

- ime UNDP instance;

- optimalno rešenje dobijeno MIP paketima;
- broj OSL iteracija;
- broj CPLEX iteracija;
- vreme izvršavanja CPLEX-a (u sekundama);
- prosečan broj generacija potreban za završetak EA izvršavanja;
- prosečno vreme izvršavanja EA (u sekundama);
- najbolje rešenje do kog je stigao EA.

OSL nije bio u mogućnosti da dobije optimalna rešenja za instance u grupama MF, MH i MJ (osim instance MF1 koju je rešio za 5 sati i 41 minut, mereno časovnikom sa strane).
Paralelna verzija CPLEX-a 6.5.3, koja se izvršavala na Sun Solaris računaru sa 4 procesora od 400MHz i sa 4×1 GB RAM-a. CPLEX je, pored instanci koje je rešio OSL, rešio još i MF2-MF5, ali su grupe MH i MJ ostale nerešene.

UNDP instanca	Optimalno rešenje	OSL iter.	CPLEX iter.	CPLEX vreme (s)	EA pros. gener.	EA pros. vreme (s)	EA naj. rešenje
MA1	55.193	51	66	<0.01	31.9	0.020	optimalno
MA2	53.221	51	55	<0.01	32.5	0.020	optimalno
MA3	57.684	44	61	0.01	36.9	0.020	optimalno
MA4	56.047	60	67	0.02	38.6	0.020	optimalno
MA5	55.215	36	46	0.01	34.8	0.020	optimalno
MB1	81.755	106	107	0.03	104.8	0.102	optimalno
MB2	85.945	344	121	0.09	78.0	0.082	optimalno
MB3	81.445	335	120	0.07	84.5	0.089	optimalno
MB4	94.471	105	115	0.03	94.6	0.097	optimalno
MB5	90.216	101	112	0.03	102.0	0.095	optimalno
MC1	87.974	254	119	0.03	305.1	0.321	optimalno
MC2	107.735	207	164	0.05	262.1	0.293	optimalno
MC3	85.750	270	120	0.04	233.3	0.237	optimalno
MC4	87.560	112	91	0.03	252.9	0.279	optimalno
MC5	110.498	143	135	0.04	248.9	0.284	optimalno
MD1	105.877	396	186	0.05	1056	1.041	optimalno
MD2	133.368	629	127	0.05	915.8	0.859	optimalno
MD3	114.454	347	177	0.06	896.9	0.907	optimalno
MD4	105.776	400	145	0.05	920.3	0.926	optimalno
MD5	116.365	488	144	0.05	829.8	0.843	optimalno
ME1	137.823	828	225	0.11	3418	4.532	optimalno
ME2	129.454	825	280	0.13	4377	5.601	optimalno
ME3	127.715	2144	462	1.82	4182	4.946	optimalno
ME4	135.951	775	236	0.10	4206	4.258	optimalno
ME5	140.869	2397	346	0.89	5506	6.457	optimalno

tabela 5.29. Dobijeni rezultati za grupe instanci MA-ME.

UNDP instanca	Optimalno rešenje	OSL iter.	CPLEX iter.	CPLEX vreme (s)	EA pros. gener.	EA pros. vreme (s)	EA naj. rešenje
MF1	704.810	2176540	339852	1926	5288	54.400	optimalno
MF2	713.520	-	17839	226	4305	47.146	optimalno
MF3	720.708	-	2660714	14027	5792	58.901	optimalno
MF4	671.535	-	1448541	8651	5943	67.383	optimalno
MF5	739.462	-	2855116	16395	5527	57.136	741.679
MG1	111.160	2394	274	0.18	6848	12.031	optimalno
MG2	144.824	9621	508	3.35	7135	16.386	145.658
MG3	129.799	6927	376	0.84	6743	11.668	132.167
MG4	126.965	2853	438	0.27	6833	14.500	optimalno
MG5	132.544	2310	387	0.22	6808	12.947	135.200
MH1	-	-	≈10mil	≈2days	12428	358.154	885.605
MH2	-	-	-	-	10724	307.772	878.704
MH3	-	-	-	-	13916	401.364	886.907
MH4	-	-	-	-	12715	379.558	875.131
MH5	-	-	-	-	12838	337.036	848.895
MI1	125.352	14675	725	0.89	13072	58.729	128.972
MI2	117.254	13560	540	0.58	11512	53.040	118.451
MI3	136.258	103555	702	0.79	12596	61.498	143.132
MI4	147.563	15182	1142	1.40	12959	66.826	157.165
MI5	136.423	17164	647	0.58	13443	60.518	155.468
MJ1	-	-	-	-	25801	2414.86	1248.595
MJ2	-	-	-	-	13544	1214.35	1252.019
MJ3	-	-	-	-	26730	1917.38	1164.988
MJ4	-	-	-	-	32524	2480.68	1136.297
MJ5	-	-	-	-	38286	3261.73	1192.727

tabela 5.30. Dobijeni rezultati za grupe instanci MF-MJ.

Kao što se može videti iz tabele 5.29. 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. Kao što je ranije istaknuto, kod MH i MJ grupa instanci, samo je EA došao do rešenja, pa se kvalitet tih rešenja nema sa čime porediti.

5.5.6. Primene FGTS u algoritmima za rešavanje još nekih NP teških problema

Procena kvaliteta FGTS u primeni na realnim problemima u ovoj sekciji ne bi bila kompletna ako se (u najkraćim crtama) ne bi opisali i rezultati koji su dobijeni pri rešavanju ISP, UMApHMP, UMApHCP i DOMP.

Problem izbora indeksa (ISP)

Problem izbora indeksa (eng. 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, bude minimizirano vreme odziva sistema za upravljanje bazom podataka.

Matematička formulacija

Problem izbora indeksa može biti formulisan na sledeći način: 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 sledećih formula, koje detaljnije opisuju sve uslove koje treba rešenje da ispunjava:

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

uz uslove

$$\sum_{k\in P} x_{ik} \le 1 \qquad \forall i \in M$$

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

Rešenje je zadato preko sledećih promenljivih:

- y_j 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.

$$y_j = \begin{cases} 1, & ako\ je\ indeks\ j\ postavljen \\ 0, & ako\ nije\ postavljen \end{cases}$$

$$x_{ik} = \begin{cases} 1, & ako\ su\ postavljeni\ indeksi\ konfiguracije\ k\ pri\ upitu\ i \\ 0, & ako\ nisu\ postavljeni \end{cases}$$

Metode rešavanja problema

Dve metode za rešavanje problema selekcije indeksa su date u radu [Caprar95]. Jedna je heuristika zasnovana na Lagrange-ovoj dekompoziciji pogodno definisanog potproblema ranca (eng. knappsac problem). 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 zasnovana na LP relaksaciji, koja daje optimalno rešenje. Prezentirani su rezultati izvršavanja na ISP instancama gde su m, n i p su nekoliko stotina.
U [Caprar95a] je prikazano jedno poboljšanje prethodne metode pomoću tehnike grananja i odsecanja (eng. Branch and Cut - BnC), pri čemu je ISP formulisan kao problem pakovanja skupa (eng. set packing problem). Ova metoda daje optimalno rešenje, a primenjena je na ISP instance dimenzije do m = 100, n = 100 i p = 1000.

Instance problema

Za razliku od prethodno rešavanih problema koji su bili nešto opštijeg tipa i koji su mogli modelirati veći broj praktičnih problema, ovaj problem je specifičan i njegova osnovna namena je smanjivanje vremena izvršavanja kod sistema za upravljanje baze podataka. Zbog toga vrednosti ulaznih podataka moraju biti posebno izabrane, tako da što vernije preslikavaju osobine realnih baza podataka.
Budući da su u radu [Caprar95] detaljno opisani svi aspekti ovog problema vezani za primenu na konkretnim bazama podataka u praksi, oni su zatim iskorišćeni za generisanje verodostojnih ISP instanci. Napomenimo da su instance na kojima je izvršavan GA istih osobina kao originalne iz rada [Caprar95a], iako nisu potpuno iste, zbog različito postavljene početne vrednosti za generator pseudoslučajnih brojeva.

EA implementacija

Za rešavanje problema ISP pomoću EA korišćen je GANP sistem, sa sledećim elementima (detaljno opisanim u [Kratic03]):

- Kako je matrica problema dosta retka, tj. broj nenula vrednosti g_{ik} ($i \in$ M, $k \in$ P) je relativno mali u odnosu na dimenziju matrice, memorišu se samo nenulte vrednosti.
- Pri rešavanju datog problema izabrano je 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).
- Pri rešavanju ISP su primenjeni isti genetski operatori koji su se pokazali najboljim i u slučaju prethodnih problema (fino gradirana turnirska selekcija sa željenom srednjom veličinom turnira 5.5 , uniformno ukrštanje sa verovatnoćom ukrštanja 0.85 i prosta mutacija realizovana korišćenjem normalne raspodele). Takođe je korišćena elitistička strategija zamene jedinki i uklanjanje višestruke pojave jedinki u populaciji, što je i ovog puta efikasno sprečilo pojavu preuranjene konvergencije.

Primenjena je ponovo veličina populacije od 150 jedinki (od čega 100 elitnih), kao i generisanje početne populacije sa istom učestalošću 0 i 1 u genetskom kodu, kao pri rešavanju SPLP.
U ovoj EA implementaciji je korišćen nivo 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.

- I pri rešavanju ISP problema korišćeno je keširanje EA, sa veličinom keša 5000. Izvršavanje algoritma je prekidano posle 2000 generacija.

Eksperimentalni rezultati pri poređenju algoritama za ISP

Tabela 5.31. prikazuje prosečne rezultate EA implementacije, koji su dobijeni rešavanjem 40 instanci klase A i detaljno prikazani u [Kratic03]. Instance su rešavane na računaru sa AMD Athlon K7 procesorom, koji radi na frekvenciji 1.333 GHz (u klasi Pentium IV), sa 256 MB SDRAM operativne memorije. EA je izvršavan po 20 puta na svakoj instanci.

OSL je izvršavan na istoj mašini (AMD Athlon K7/1.333GHz), ali je CPLEX, shodno licenci za njegovo korišćenje, izvršavan na računaru sa Intel Pentium III procesorom na 800MHz, koji ima 1GB operativne memorije.

Rezultati algoritma grananja i odsecanja, opisanog u [Caprar95a] su, radi poređenja, prikazani u tabeli 5.31. direktno naspram rezultata koje je dao EA. Prve dve kolone koje opisuju BnC rezultate su vreme izvršavanja LP heuristike(root-t[s]) i procentima iskazan kvalitet LP heuristikom dobijenog rešenja (root-gap[%]). Sledeća kolona sadrži ukupno vreme za izvršenja BnC algoritma i dobijanje optimalnog rešenja. Sledeće dve kolone (koje se odnose na EA) imaju isto značenje kao i istoimene kolone u tabeli 5.31: prosečno vreme potrebno da EA dostigne najbolje rešenje i prosečno vreme potrebno da se izvrši svih 2000 generacija.

Za instance tipa i200, i175, i150 i i125, rešenja dobijena LP heuristikom su već optimalna, pa se ne izvršava nikakvo grananje. Za neke od ovih instanci, BnC algoritam je čak brži od EA.

Sa druge strane, instance tipa i100, i075, i050, i i025 predstavljaju veliki izazov za algoritme koji određuju tačan rezultat, jer tu vreme izvršavanja grananja i isecanja eksponencijalno raste. Za ove instance čak i vreme inicijalizacije algoritma bar četvorostruko prevazilazi prosečno vreme izvršavanja EA. Nadalje, kvalitet inicijalnih rešenja je veoma loš (npr. za grupu instanci i025, inicijalno rešenje je čak za više od 30% lošije od optimalnog.

Nasuprot eksponencijalnoj prirodi BnC-a, EA se izvršava veoma efikasno, održavajući zadovoljavajući kvalitet dobijenih rešenja (u najgorem slučaju nivo uspeha je bio 10% i ukupno vreme izvršavanja je bilo kraće od dve sekunde).

Kod 50% instanci, EA je imao nivo uspeha 100%, dok u 70% instanci prosečno odstupanje od optimalnog bilo ne veće od 0.1%. U samo jednoj od 40 testiranih instanci (i025.444), odstupanje je bilo veće od 3%.

Instance	BnC (HP9000/720)			EA (AMD K5)	
	root-t[s]	root-gap[%]	t_{tot}[s]	t[s]	t_{tot}[s]
I200.111	1.1	0	1.1	1.5	22.9
I200.222	1.0	0	1.0	1.6	21.2
I200.333	0.9	0	0.9	2.3	28.0
I200.444	1.0	0	1.0	1.3	20.1
i200.555	1.4	0	1.4	1.5	18.7
i175.111	1.2	0	1.2	2.0	22.5
i175.222	1.3	0	1.3	1.6	21.4
i175.333	1.1	0	1.1	3.8	25.4
i175.444	1.1	0	1.1	1.6	21.0
i175.555	2.0	0	2.0	1.3	19.1
i150.111	2.0	0	2.1	1.6	20.6
i150.222	2.5	0	2.5	1.9	24.1
i150.333	1.3	0	1.3	3.3	24.4
i150.444	2.1	0	2.1	1.6	21.3
i150.555	3.3	0	3.3	1.4	20.7
i125.111	4.7	0	4.8	1.5	20.5
i125.222	5.2	0	5.2	2.0	25.0
i125.333	1.6	0	1.6	2.8	23.2
i125.444	4.3	0	4.3	1.5	20.3
i125.555	7.4	0	7.4	2.0	24.1
i100.111	32.6	0	33.3	8.1	24.9
i100.222	25.4	0	25.4	6.9	27.8
i100.333	4.8	0	4.8	2.4	22.4
i100.444	31.6	0.2	37.6	1.5	18.8
i100.555	26.6	0.3	45.2	5.2	26.8
i075.111	83.8	1.9	391.2	3.6	22.3
i075.222	78.9	1.8	773.4	9.7	26.0
i075.333	47.5	0.5	75.9	6.1	27.4
i075.444	92.1	2.3	815.2	1.9	20.3
i075.555	79.1	1.9	457.2	3.4	26.0
i050.111	139.6	7.4	7553.5	4.5	23.8
i050.222	152.4	7.1	6583.4	7.6	25.6
i050.333	134.2	5.2	1692.0	7.7	24.9
i050.444	192.5	8.0	9284.9	5.8	24.5
i050.555	157.4	8.1	10566.7	2.8	23.4
i025.111	565.3	37.7	>50000	7.0	23.2
i025.222	595.5	38.4	>50000	6.3	23.3
i025.333	585.5	36.0	>50000	7.0	22.7
i025.444	691.3	35.8	>50000	7.6	21.4
i025.555	584.6	34.9	>50000	5.2	22.2

tabela 5.31. Poređenje rezultata dobijenih LP-heuristikama, BnC i GA

Problem neograničene višestruke alokacije p-hab medijane (UMApHMP)

Postoje različite formulacije za probleme izbora. Ove formulacije uključuju eventualne restrikcije kapaciteta habova (to mogu biti restrikcije na kapacitet haba, na količinu saobraćaja između habova ili na količinu prikupljanja odnosno raspodele tj. saobraćaja habova sa izvornim, odnosno odredišnim ne-hab čvorovima), cenu uspostavljanja haba, unapred određen broj habova (i tada se radi o tzv. p-hab problemu), kao i razne druge aspekte.

Pri formulisanju hab problema mora da se odluči i o tome da li se radi o jednostrukoj ili višestrukoj alokacionoj shemi (ove sheme alokacije su opisane u prethodnim poglavljima). U ovom poglavlju se proučava Problem neograničene višestruke alokacije p-hab medijane (eng. Uncapacitated Multiple Allocation p-hub Median Problem - UMApHMP).

Matematička formulacija

Problem neograničene višestruke alokacije p-hab medijane se matematički može formulisati koristeći formulaciju mešanog celobrojnog linearnog programiranja - MILP:

Razmotrimo skup $I=\{1,..,n\}$ od n različitih čvorova mreže, gde svaki od čvorova predstavlja izvor/odredište ili hab. Rastojanje od čvora i do čvora j iznosi C_{ij} i može se pretpostaviti da važi nejednakost trougla. Zahtev

od lokacije i do lokacije j označen je sa W_{ij}. Broj habova koje treba uspostaviti je fiksan i jednak p. Promenljive odluke Z_{ik}, $Y_{kl}{}^{i}$ i $X_{lj}{}^{i}$ koje se koriste u formulaciji problema uvode se na sledeći način:

- $H_j = 1$, ako je hab uspostavljen u lokaciji j, 0 inače;
- Z_{ik} je količina saobraćaja iz čvora i koja je prikupljena u habu k.
- $Y_{kl}{}^{i}$ je količina saobraćaja iz čvora i koja je prikupljena u habu k i raspodeljena kroz hab l.
- $X_{lj}{}^{i}$ je količina saobraćaja iz čvora i koja je upućena prema čvoru j i raspodeljen kroz hab l.

Isto kao i kod prethodno opisanih hab problema, parametri γ i δ predstavljaju jediničnu cenu za prikupljanje i raspodelu, dok 1-α predstavlja faktor sniženja cene za transport između habova.

Zadatak kod ovog problema je uspostavljanje p habova tako da cena ukupnog saobraćaja bude minimalna. Sada se problem može ovako zapisati:

$$min \sum_i \left[\chi \sum_k C_{ik} Z_{ik} + \alpha \sum_k \sum_l C_{kl} Y_{kl}{}^{i} + \delta \sum_l \sum_j C_{lj} X_{lj}{}^{i} \right]$$

pri čemu moraju da važe sledeći uslovi:

$$\sum_i H_i = p;$$

$$\sum_k Z_{ik} = \sum_j W_{ij}, \text{ za svako } i;$$

$$\sum_l X_{lj}{}^{i} = W_{ij}, \text{ za svako } i, j;$$

$$\sum_l Y_{kl}{}^{i} + \sum_j X_{kj}{}^{i} - \sum_l Y_{lk}{}^{i} - Z_{ik} = 0 \text{ za svako } i, k;$$

$$Z_{ik} \le \sum_j W_{ij} H_k \text{ za svako } i, k;$$

$$\sum_i X_{lj}{}^{i} \le \sum_i W_{ij} H_l \text{ za svako } l, j;$$

$$X_{lj}{}^{i}, Y_{kl}{}^{i}, Z_{ik} \ge 0, \qquad H_k \in \{0,1\} \qquad \text{za svako } i, j, k, l;$$

Dakle, traži se minimum funkcije koja predstavlja cenu celokupnog saobraćaja u mreži. Prvi uslov ograničava broj uspostavljenih habova na p, sledeća tri uslova predstavljaju jednačine divergencije za mrežni saobraćaj kroz svaki od čvorova, dva uslova koji potom slede sprečavaju direktnu komunikaciju između ne-hab čvorova, a poslednji uslov odslikava nenegativnost i/ili binarnu reprezentaciju promenljivih odluke.

Metode rešavanja problema

Hab problemi se u poslednje vreme veoma intenzivno izučavaju, ali najveći deo izučavanja se odnosi na jednostruke alokacione sheme.

Ernst and Krishnamoorthy su, u radu [Ernst98a], koristili metod najkraćih puteva (SP) da enumerišu sve moguće lokacije za habove kod UMApHMP. Taj algoritam je eksponencijalan po p i polinomijalan po n, pa u slučajevima kada je broj habova u instanci problema relativno mali ($p \le 5$) ovaj pristup daje tačno rešenje za relativno kratko vreme izračunavanja. Kod većih instanci SP pristup može da se kombinuje sa tehnikom grananja i ograničavanja.

Za rešavanje ovog problema je predložen i dvonivoovski algoritam tabu pretrage (videti [Skorin94]). Posle izbora skupa od p habova, ne-hab čvorovi se dodeljuju najbližim habovima i tako se dobija polazno rešenje za tabu pretragu. Tokom svog izvršavanja, tabu pretraga istražuje i menja dodele kako bi se pronašlo bolje rešenje. Isti autori su, dve godine kasnije, u radu [Skorin96] dokazali da je tabu pristup doveo do optimalnog rešenja kod svih ORLIB CAB instanci.

U radu [Boland04] je razvijena heuristika i egzaktan metod grananja i ograničavanja kojim se rešava više raznovrsnih hab-problema, korišćenjem preprocesiranja i algoritama isecanja. Tu se, u prvoj fazi, heuristikom dobiju dobre gornje granice, koje se potom koriste za isecanje drveta grananja i ograničavanja.

Instance problema

Algoritmi su testirani na dva skupa ORLIB instanci CAB i AP, preuzeta iz [BeaslJ96] i opisana u prethodnom poglavlju. :

EA implementacija

Predloženi EA za rešavanje UMApHMP (opisan u radu [Stan06a] i implementiran preko GANP softverskog sistema) dizajniran je na sledeći način:

- U ovoj implementaciji, jedinke su binarno kodirane – dakle, radi se o GA. Svako od rešenja (odnosno svaka jedinka populacije EA) predstavljena je binarnom niskom dužine *n*. Jedinica na datoj poziciji binarne niske ukazuje da je na odgovarajućem čvoru uspostavljen hab, a nula označava da nije.
  Za određivanje dodele kada je fiksiran skup habova koristi se dobro poznat Flojd-Warsallov algoritam za određivanje najkraćeg puta
- EA implementacija koristi fino gradiranu turnirsku selekciju (FGTS) sa vrednošću parametra 5.4 kao selekcioni operator.
- Jednopoziciono ukrštanje, kakvo postoji kod SGA, nije adekvatan operator ukrštanja za ovaj problem, sličan operatoru ukrštanja za GAHUB2 implementaciju kod rešavanja USApHMP. Naime, prosta razmena dva segmenta bitova može od dve korektne jedinke proizvesti nekorektan potomak (koji nema tačno *p* jedinica u genetskom kodu). Stoga se u ovoj EA implementaciji koristi modifikovano ukrštanje (detaljno opisano u [Stan06a] . Verovatnoća primene ovog operatora ukrštanja je takođe ostala nepromenjena u odnosu na ranije opisane primene i iznosi 0.85.
- Ova EA implementacija koristi isti operator mutacije koji je sličan mutaciji kod EA implementacije za rešavanje UMAHLP, a koji je detaljno opisan u [Stan06a].
- Svaka jedinka inicijalne populacije se generiše korišćenjem sledeće strategije:svaki bit u genetskom genu se postavlja pseudoslučajno i dobija vrednost 1 sa verovatnoćom p/n , a potom se jedinka koriguje tako da ima tačno *p* jedinica. Prethodno opisana korekcija se izvršava samo jednom (pri kreiranju inicijalne populacije), a potom se, s obzirom da su ukrštanje i mutacija dizajnirani tako da čuvaju korektnost jedinki, nekorektne jedinke više ne mogu pojaviti, pa nema više potrebe za popravljanjem jedinki.
- Veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Sto jedinki iz populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se i ovde radi postavljanjem prilagođenosti duplikata na nulu.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. I ovde se, radi izbegavanja takvih situacija, ograničava (na 40) broj jedinki koje imaju istu prilagođenost, a različite genetske kodove.
- I u ovoj implementaciji se izvršavanje EA zaustavlja posle 5000 generacija (ako se rešavaju veće instance) ili posle 500 generacija (ako se rešavaju male instance), odnosno, ako se prilagođenost najbolje jedinke nije popravila tokom 2000 generacija (kod većih instanci) ili posle 200 generacija (kod manjih instanci).
- Performanse metoda su i ovde poboljšane keširanjem EA sa kešom veličine 5000.

Eksperimentalni rezultati pri poređenju algoritama za UMApHMP

U ovom poglavlju se prikazuju rezultati izvršavanja EA i porede sa drugim algoritmima koji su korišćeni za rešavanje UMApHMP. Sva izvršavanja EA su realizovana na računaru sa AMD K7 procesorom na 1.33GHz, koji ima 256MB operativne memorije.

Algoritam je (videti [Stan06a]) izvršavan 20 puta na svakoj instanci (osim AP instanci gde je $n \geq 100$ - tu je izvršavan sam 10 puta).

Tabele 5.32. sadrži rezultate izvršavanja EA na velikim AP instancama i ima sledeću strukturu: prve dve kolone sadrže dimenzije instance problema; kolona koja sledi sadrži optimalno rešenje za tu instancu, ako je unapred poznato (ako nije upisuje se crtica „-“); sledeća kolona sadrži najbolje rešenje do koga je došao EA (odnosno „opt“ ako se najbolje rešenje poklopilo sa optimalnim); potom kolona **t** sadrži vreme potrebno da EA dođe od najboljeg rešenja, a $\mathbf{t_{tot}}$ vreme potrebno za izvršenje svih 500/5000 generacija; kolona koja sledi, označena sa **gen**, sadrži prosečan broj generacija potreban da EA dobije najbolje rešenje; na kraju, poslednje dve kolone sadrže prosečno procentualno odstupanje od najboljeg/optimalnog rešenja u svih 20/10 izvršavanja i disperziju procentualnog odstupanja od najboljeg/optimalnog rešenja.

n	p	Opt. reš.	EA naj.	t[s]	ttot[s]	gen	jaz[%]	σ [%]
40	6	122171.26	opt	0.247	4.834	2039	0.000	0.000
40	7	-	116036.38	0.467	6.002	2086	0.000	0.000
40	8	-	109971.92	0.579	7.655	2085	0.000	0.000
40	9	-	104212.42	0.884	9.010	2127	0.000	0.000
40	10	-	99452.67	0.779	9.491	2085	0.000	0.000
50	6	121671.76	opt	1.537	9.150	2284	0.000	0.000
50	7	-	115911.64	4.872	15.725	2851	0.000	0.000
50	8	-	109926.60	4.294	17.188	2591	0.000	0.000
50	9	-	104968.27	2.869	17.252	2298	0.000	0.000
50	10	100508.95	opt	4.333	21.136	2412	0.000	0.000
50	11	-	96186.22	5.294	24.675	2454	0.000	0.000
50	12	-	93171.96	3.714	23.870	2267	0.000	0.000
50	13	-	90409.79	4.255	27.221	2281	0.000	0.000
50	14	-	87654.61	3.972	29.098	2238	0.000	0.000
50	15	-	85032.89	7.463	35.493	2456	0.000	0.000
50	20	-	73490.33	2.824	39.859	2094	0.000	0.000
100	2	176245.38	opt	0.639	2.736	2089	0.000	0.000
100	3	157869.93	opt	2.195	13.227	2207	0.000	0.000
100	4	143004.31	opt	9.007	32.848	2652	0.000	0.000
100	5	133482.57	opt	20.067	54.389	3097	0.000	0.000
100	6	-	126107.56	58.421	99.973	4350	0.000	0.000
100	7	-	120165.15	45.945	100.118	3553	0.011	0.024
100	8	-	114295.92	77.750	125.793	3891	0.228	0.355
100	9	-	109448.87	54.651	126.037	3409	0.002	0.005
100	10	-	104794.05	63.355	146.263	3421	0.001	0.002
100	15	-	88882.05	150.193	270.956	4004	0.093	0.162
100	20	-	79191.02	195.747	377.160	3828	0.139	0.152
200	2	178093.99	opt	8.123	35.686	2129	0.000	0.000
200	3	159725.11	opt	43.393	174.900	2520	0.000	0.000
200	4	-	144508.20	172.663	376.815	3585	0.001	0.002
200	5	-	136761.83	357.326	562.245	4231	0.096	0.092
200	6	-	129560.60	393.868	681.338	4281	0.046	0.062
200	7	-	123609.44	460.543	766.016	4219	0.051	0.070
200	8	-	117709.98	566.177	879.377	4237	0.213	0.189
200	9	-	112380.66	869.886	1096.180	4809	0.066	0.146
200	10	-	107846.82	847.216	1157.049	4591	0.090	0.189
200	15	-	92669.64	1246.186	1750.105	4699	0.397	0.275
200	20	-	83385.94	1935.840	2425.588	4924	0.169	0.232

tabela 5.32. Rezultati EA na velikim AP instancama

Tabela 5.33. sadrži već istaknute rezultate izvšavanja EA na većim AP instancama uporedo sa rezultatima rada drugih metoda rešavanja UMApHMP na istim instancama.

Prva kolona tabele 5.33. sadrži parametre izabrane instance (*n* i *p*), a sledeće kolone redom označavaju:

- odstupanje najboljeg EA rešenja od optimalnog i prosečno vreme izvršavanja na AMD K7/1.33GHz računaru;
- rezultate SP heuristike (opisane u [Ernst98]), čiji su autori Ernst i Krishnamoorty (EK – SP – heur), izvršavane na DEC 3000/700 računaru baziranom na Alpha čipu brzine 200MHz;
- rezultate tačnog BnB metoda (opisanog u [Ernst98]), čiji su autori Ernst i Krishnamoorty (EK – BnB – opt), koji je izvršavan na istom DEC 3000/700 računaru;
- rezultate tačnog BnB metoda (opisanog u [Boland04]), čiji je autor Boland sa saradnicima (BKEE – meth - II), izvršavanog na DEC radnoj stanici baziranoj na Alpha čipu frekvencije 500MHz.

n	p	GA		EK - SP - heur		EK - BnB - opt		BKEE - meth - II		
		jaz	CPU	jaz	CPU	nodes	CPU	root - gap	nodes	CPU
25	4	0.00	0.139	0.77	0.40	708	1.68	4.09	43	245.07
50	5	0.00	1.282	0.07	10.95	10187	143.48	3.46	89	27597
50	10	0.00	21.136	0.14	66.71	2125737	57243	-	-	-
100	5	0.00	54.389	0.00	168.49	153266	7688	-	-	-
200	3	0.00	174.900	0.00	632.70	25349	3636.6	-	-	-

tabela 5.33. Poređenje sa ostalim metodama

EA je (videti [Stan06a]) dostigao sva poznata optimalna rešenja. Tabela 5.33. pokazuje da je vreme izvršavanja EA slično vremenima drugih heuristika, pri čemu je kvalitet rešenja nešto bolji. Sa druge strane, tačne metode imaju mnogo duže vreme izvršavanja. Kako njihovo vreme izvršavanja raste eksponencijalno sa porastom dimenzije problema, one se ne mogu izvršavati na najvećim instancama (n=200, p=20). Bilo bi interesantno da se na takvim instancama porede heuristički metodi za UMApHMP, ali u literaturi zasad nema podataka koji to omogućavaju.

Problem neograničene višestruke alokacije p-hab centra (UMApHCP)

U ovom poglavlju se obrađuje posebna varijanta problema uspostavljanja habova poznata pod imenom problem neograničene višestruke alokacije p-hab centra (eng. Uncapacitated Multiple Allocation p-hub Center Problem - UMApHCP).

Cilj je da se reorganizuje saobraćaj u mreži uspostavljanjem p habova (između n čvorova mreže) tako da najveće rastojanje između dva čvora u mreži (odnosno troškovi saobraćaja, odnosno vreme prenosa) budu što je moguće manji – naravno, podrazumeva se da saobraćaj u mreži ide isključivo preko habova.

Na primer, UMApHCP model može uspešno da se primeni pri dizajnu DHL poštanskog sistema, ili drugog kurirskog sistema u kom se mora garantovati isporuka pošiljke unutar prethodno određenog (obično dosta kratkog) intervala.

Matematička formulacija

Problem neograničene višestruke alokacije p-hab centra se matematički može formulisati na sledeći način:

Neka je $I = \{1, ..., n\}$ skup n različitih čvorova mreže, gde svaki čvor predstavlja izvor/odredište ili potencijalnu lokaciju uspostavljenog haba. Za svaki par čvorova i i j rastojanje među njima se označi sa C_{ij}, pri čemu važi $C_{ij} = C_{ji}$. Pretpostavlja se da za vrednosti rastojanja važi nejednakost trougla.

Tok (tj. saobraćaj u mreži) polazi od izvornog čvora i, usmerava se prema habu, pa može biti preusmeravan kroz jedan ili više habova pre nego što bude raspodeljen odredišnom čvoru j. Kod UMApHCP mora biti uspostavljeno tačno p habova. Binarne promenljive odluke H_k, X_j^{ik} i Y_{lj}^{i} uvode se na sledeći način:

- $H_k = 1$ ako je hab uspostavljen na čvoru k, inače je 0;
- $X_j^{ik} = 1$ ako je izvorni čvor i dodeljen habu k za konkretan par izvor-odredište (i, j), inače je 0;
- $Y_{lj}^{i} = 1$ ako je odredišni čvor j dodeljen habu l za konkretan par izvor-odredište (i, j), inače je 0.

Parametar α uzima vrednosti iz [0, 1], tako da 1-α označava faktor sniženje za transport između habova. Definišimo $C_{max} = max\ \{C_{ij} \mid i, j \in I\}$ i neka z predstavlja slobodnu promenljivu cilja. Sada problem glasi:

$$min\ z$$

pri čemu važe sledeći uslovi:

$$\sum_{k=1}^{n} H_k = p$$

$X_j^{ik} \le H_k$ za svako $i, j, k \in I$

$Y_{lj}^{i} \le H_l$ za svako $i, j, l \in I$

$$\sum_{k=1}^{n} X_j^{ik} = 1 \text{ za svako } i, j \in I$$

$$\sum_{l=1}^{n} Y_{lj}^{i} = 1 \text{ za svako } i, j \in I$$

$$z \ge \sum_{k=1}^{n} \left(C_{ik} + \alpha \cdot C_{kl}\right) \cdot X_j^{ik} + \sum_{m=1}^{n} C_{mj} \cdot Y_{mj}^{i} - \alpha \cdot (1 - Y_{lj}^{i}) \cdot C_{max} \text{ za svako } i, j, l \in I$$

$H_k, X_j^{ik}, Y_{lj}^{i} \in \{0, 1\}$ za svako $i, j, k, l \in I$

Dakle, cilj UMApHCP je minimizacija maksimuma ukupne cene koštanja saobraćaja između dva čvora u mreži. Pri tome, broj habova koji se uspostavljaju mora biti tačno p, svaki izvor/odredište čvor mora biti dodeljen habu, a data je i donja granica za promenljivu cilja z.

Metode rešavanja problema

Jedini rad u dosadašnjoj literaturi koji proučava UMApHCP je [Ernst04]. Autori tog rada su dali dve celobrojne formulacije problema. Oni su takođe razvili dva heuristička metoda za rešavanje varijacija ovog

problema sa shemom jednostruke alokacije i sa shemom višestruke alokacije. Njihova heuristika, označena sa EBnB, je zasnovana na metodu najkraćih puteva i na grananju i ograničavanju. Rezultati izvršavanja na standardnim ORLIB instancama CAB i AP, pokazuju da ova heuristika brzo dovodi do rešenja, ali da je kod nekih instanci odstupanje dobijenog rešenja od optimalnog i preko 10% (na primer, za AP instancu gde je n=25 i p=3). Najveća razmatrana instanca je n=100, p=10 i kod ove instance je EBnB dao rešenje posle više od tri dana izvršavanja algoritma. Za instance većih dimenzija nisu prikazani rezultati do kojih je došla heuristika.

Instance problema
Algoritmi su testirani na dva ranije opisana skupa ORLIB instanci CAB i AP, preuzeta iz [BeaslJ96] :

EA implementacija
EA implementacija za rešavanje UMApHCP dizajnirana je a sledeći način i podrobno opisana u radu [Kratic06]:

- U ovoj implementaciji se koristi binarno kodiranje jedinki. Svako rešenje je predstavljeno binarnom niskom dužine n. Jedinica u binarnoj nisci ukazuje da je hab uspostavljen, a nula da nije.
- I ova EA implementacija koristi FGTS kao selekcioni operator sa vrednošću parametra 5.4., Razlog za takvu odluku je isti kao i kod prethodnih EA implementacija: ovaj operator, sa vrednošću parametra 5.4 je tokom eksperimentisanja pokazao bolje rezultate u poređenju sa ostalim operatorima selekcije (pa i sa FGTS gde vrednost parametra nije bila 5.4).
- EA implementacija koristi specijalizovani operator ukrštanja , koji očuvava korektnost jedinki, a koji je detaljno opisan u [Kratic06]. I u ovoj EA implementaciji ukrštanje biva izvršavano sa nivoom 0.85.
- Operator mutacije je realizovan na potpuno isti način kao kod UMApHMP: određuju se zamrznuti geni i nivo mutacije za zamrznute gene ($1.0/n$) se postavlja ne vrednost dva i po puta veću od nivoa mutacije ostalih gena ($0.4/n$). Potom se ujednačava broj mutiranih jedinica i nula, čime operator mutacije očuvava broj uspostavljenih habova, tj. očuvava korektnost mutirane jedinke.
- Svaka jedinka inicijalne populacije se generiše korišćenjem iste strategije koja je korišćena pri rešavanju UMApHMP:svaki bit u genetskom genu se postavlja pseudoslučajno i dobija vrednost 1 sa verovatnoćom p/n , a potom se nekorektne jedinke koriguju dodavanjem (odnosno brisanjem) potrebnog broja jedinica sa kraja genetskog koda.
- Opet je veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Dve trećine populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se i ovde radi postavljanjem prilagođenosti duplikata na nulu, tako da ih sledeća primena selekcije automatski izbacuje.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. I ovde se, radi izbegavanja takvih situacija, ograničava (na 40) broj jedinki koje imaju istu prilagođenost, a različite genetske kodove.
- U ovoj implementaciji se izvršavanje EA zaustavlja posle 10000 generacija (ako se rešavaju veće instance) ili posle 1000 generacija (ako se rešavaju male instance), odnosno, ako se prilagođenost najbolje jedinke nije popravila tokom 4000 generacija (kod većih instanci) ili posle 400 generacija (kod manjih instanci).
- Performanse metoda su i ovde poboljšane keširanjem EA (videti [Kratic99], [Kratic06]) sa kešom veličine 5000.

Eksperimentalni rezultati pri poređenju algoritama za UMApHCP
U ovom poglavlju se prikazuju rezultati izvršavanja EA i porede sa postojećim algoritmima koji rešavaju UMApHCP. Sva izvršavanja EA su realizovana na računaru sa AMD K7 procesorom na 1.33GHz, koji ima 256MB operativne memorije.
Algoritam je izvršavan 20 puta na svakoj instanci (osim AP instanci gde je $n \geq 100$ - tu je izvršavan sam 10 puta).

Iz podataka dobijenih tokom eksperimenta se lako uočava (detaljno opisano u [Kratic06]) da na malim instancama EA dostiže optimalno rešenje za ne više od 20 sekundi. Na velikim AP instancama, koje nisu bile ranije rešene, EA dolazi do rešenja za manje od 223 sekunde.

U literaturi nije poznat neki drugi algoritam sposoban da reši instance tako velike dimenzije. Interesantno je da čak i za ove instance EA obezbeđuje rešenje u prihvatljivom vremenskom roku. Iako se, zbog same prirode EA, ne može dokazati optimalnost dobijenih rezultata, sva prethodna razmatranja i analize ukazuju da je i kod ove implementacije i na velikim instancama EA dobio veoma kvalitetno rešenje.

Metod EBnB, opisan u [Ernst04], daje optimalna rešenja i dokazuje njihovu optimalnost, ali on dolazi do rezultata samo kod instanci veličine do n=100, p=10. Heuristički metodi EH1 i EH2, opisani u istom radu, relativno brzo dovode do rešenja, ali u nekim slučajevima su ta rešenja nezadovoljavajuća, čak i za male dimenzije problema. Što se instanci sa velikim dimenzijama tiče, primena metoda EH1 i EH2 (pri čemu EH2 daje bolje rezultate) bi verovatno dala neke rezultate, ali ostaje pod znakom pitanja kvalitet tako dobijenih rešenja (indikativno je da autori rada, odnosno heurističkih metoda nisu izvestili o rezultatima takve primene).

U tabelama 5.34. i 5.35. koje slede, prikazani su, jedni pored drugih, rezultati koje su, na instancama problema manje dimenzije, postigli EA, EH2 i EBnB metodi.

Rezultati poslednja dva metoda, preuzeti iz [Ernst04], su dobijeni izvršavanjem na DEC Alpha 3000/700 računaru.

Kolone u tabelama imaju sledeća značenja:

- prva kolona sadrži parametre instance;
- sledeće kolona **jaz** i **CPU** sadrže odstupanje najboljeg rešenja koje je dobio EA od optimuma i prosečno vreme izvršavanja EA (na AMD K7/1.33GHz računaru);
- kolone **jaz** i **CPU** sadrže odstupanje EH2 rešenja optimuma i vreme izvršavanja EH2 (na DEC Alpha računaru)
- poslednja kolona **CPU** sadrži vreme izvršavanja EBnB (na DEC Alpha računaru).

n	p	α	EA		EH2		EBnB
			jaz[%]	CPU[s]	jaz[%]	CPU[s]	CPU[s]
20	2	0.2	0.00	0.087	0.00	<0.01	0.01
20	2	0.4	0.00	0.087	0.00	0.01	0.01
20	2	0.6	0.00	0.087	0.00	0.01	0.01
20	2	0.8	0.00	0.083	0.00	<0.01	0.01
20	2	1.0	0.00	0.084	0.00	<0.01	0.01
20	3	0.2	0.00	0.110	0.00	0.01	0.02
20	3	0.4	0.00	0.104	0.00	0.01	0.02
20	3	0.6	0.00	0.111	0.00	0.01	0.03
20	3	0.8	0.00	0.111	0.00	0.01	0.03
20	3	1.0	0.00	0.110	0.00	0.01	0.01
20	4	0.2	0.00	0.148	0.00	0.02	0.04
20	4	0.4	0.00	0.158	7.76	0.02	0.04
20	4	0.6	0.00	0.149	2.29	0.02	0.14
20	4	0.8	0.00	0.155	3.13	0.02	0.04
20	4	1.0	0.00	0.190	0.00	0.02	0.02
25	2	0.2	0.00	0.097	0.00	0.01	0.01
25	2	0.4	0.00	0.095	0.00	0.01	0.01
25	2	0.6	0.00	0.098	0.00	0.01	0.01
25	2	0.8	0.00	0.096	0.00	0.01	0.02
25	2	1.0	0.00	0.098	0.00	0.01	0.01
25	3	0.2	0.00	0.144	0.17	0.02	0.05
25	3	0.4	0.00	0.166	1.28	0.02	0.04
25	3	0.6	0.00	0.146	0.00	0.03	0.04
25	3	0.8	0.00	0.146	0.00	0.03	0.04
25	3	1.0	0.00	0.149	0.00	0.02	0.03
25	4	0.2	0.00	0.233	5.94	0.05	0.10
25	4	0.4	0.00	0.255	9.46	0.05	0.09
25	4	0.6	0.00	0.292	0.00	0.05	0.10
25	4	0.8	0.00	0.220	0.37	0.06	0.12
25	4	1.0	0.00	0.304	0.00	0.04	0.06

tabela 5.34. Poređenje EA sa EH2 i sa EBnB na CAB instancama

Tabele 5.34. i 5.35. pokazuju da je vreme izvršavanja EA nešto duže nego kod EH2, ali EA dostiže optimalna rešenja a svim instancama, dok EH2 ima odstupanja na 8 od 30 CAB instanci i na 10 od 28 manjih AP instanci – pri čemu su na pojedinim instancama odstupanja izuzetno velika (na AP za n=10 i p=3; na CAB za n=25, p=4 i α=0.4). Metod EBnB, koji daje tačno rešenje, se izvršava veoma kratko za AP instance gde je $n < 50$ i/ili $p < 10$, ali vreme izvršavanja se brzo uvećava sa rastom n, a izuzetno brzo sa rastom p, pa za AP instancu gde je n=100 i p=10 ni posle trodnevnog izvršavanja algoritam nije stigao do završetka.

n	p	EA		EH2		EBnB
		jaz[%]	CPU[s]	jaz[%]	CPU[s]	CPU[s]
10	2	0.00	0.079	0.00	<0.01	<0.01
10	3	0.00	0.071	0.00	<0.01	<0.01
10	4	0.00	0.074	3.54	<0.01	0.02
10	5	0.00	0.077	0.00	<0.01	0.02
20	2	0.00	0.086	0.00	<0.01	0.01
20	3	0.00	0.108	6.09	0.01	0.02
20	4	0.00	0.149	0.99	0.02	0.04
20	5	0.00	0.182	0.00	0.03	0.04
20	10	0.00	0.726	0.00	0.12	0.15
25	2	0.00	0.097	0.00	0.01	0.01
25	3	0.00	0.154	11.10	0.02	0.03
25	4	0.00	0.208	0.00	0.04	0.06
25	5	0.00	0.335	0.00	0.07	0.10
25	10	0.00	1.270	0.00	0.31	0.40
40	2	0.00	0.149	0.00	0.05	0.08
40	3	0.00	0.412	1.75	0.14	0.25
40	4	0.00	0.810	0.00	0.27	0.55
40	5	0.00	0.732	0.00	0.42	1.09
40	10	0.00	3.479	0.00	2.18	139.6
50	2	0.00	0.219	0.00	0.12	0.18
50	3	0.00	0.695	0.00	0.30	0.60
50	4	0.00	1.241	2.42	0.60	1.72
50	5	0.00	1.571	4.33	0.97	5.26
50	10	0.00	5.558	0.00	5.38	14.587
100	2	0.00	1.218	0.00	1.55	2.53
100	3	0.00	4.575	4.00	4.45	14.47
100	5	0.00	77.079	0.76	16.51	536.8
100	10	0.00	174.825	not reported	not reported	>3 days

tabela 5.35. Poređenja EA sa EH2 i EBnB na AP instancama

Diskretno uređen problem medijane (DOMP)

Postoji veliki broj lokacijskih modela. Svi lokacijski modeli su vezani za primenu, pa njihova strukturna forma (ciljevi, zadaci, promenljive) biva određena konkretnim problemom koji se proučava. Dakle, ne postoji opšti lokacijski model koji odgovara svakom postojećem i mogućem diskretnom lokacijskom problemu. Veliki deo lokacijskih modeliranja se usmerava ka formulisanju novih i fleksibilnih lokacijskih modela koji će biti odgovarajući za različite primene i ka razvoju efikasnih tehnika za rešavanje novih, opštijih modela.

Diskretno uređen problem medijane (eng. Discrete Ordered Median Problem - DOMP) predstavlja uopštenje nekoliko poznatih lokacijskih problema, kao što su: p-medijana, p-centar, μ-cendiana i (k_1+k_2) odsečena sredina. U ovom lokacijskom problemu data je mreža sa lokacijama korisnika i potencijalnim lokacijama snabdevača. Svakog korisnika opslužuje tačno jedan, prethodno fiksiran snabdevač. Model pretpostavlja da nema ograničenja nad kapacitetom snabdevača. Daje se cena transporta za svaki par snabdevač-korisnik. Problem je da se locira unapred poznat broj snabdevača, kako bi se optimizovala funkcija cilja, koja je uopštenje ciljeva prethodno popularnih lokacijskih problema.

Matematička formulacija

DOMP se može matematički formulisati na sledeći način:

Uočiomo dati skup $I=\{1,..,n\}$ koji sadrži n različitih čvorova. Među tim čvorovima treba postaviti tačno p, $p \leq n$ snabdevača. Bez gubljenja opštosti, može se pretpostaviti da je broj kanidata za snabdevača jednak broju korisnika. Cena transporta među čvorovima je data nenegativnom matricom $C=[C_{ij}]$, $i,j=1,..,n$ i neka nema ograničenja vezanih za kapacitet snabdevača. Rešenja DOMP-a je skup X indeksa uspostavljenih

snabdevača: $X \subseteq I$, $|X| = p$. S obzirom da nema ograničanja kapaciteta snabdevača, svaki od korisnika *i* će se snabdevati od snabdevača *j* , takvog da je cena transporta do tog fiksiranog korisnika *i* minimalna – pa je $C_{ij} = C_i(X) \stackrel{def}{=} \min_{k \in X}\{C_{ik}\}$. Neka $\Lambda = (\lambda_1,..,\lambda_n),\ \lambda_i \geq 0,\ i=1,...,n$ označava vektor koeficijenata transportnih troškova za svaki od čvorova.

Za dato rešenje *X*, kvalitet rešenja se računa na sledeći način:

1. sortira se niz cena snabdevanja korisnika $C_1(X),...,C_n(X)$ u neopadajući poredak;
2. izabere se permutacija σ_X skupa *I*, takva da važe nejednakosti $C_{\sigma_X(1)}(X) \leq C_{\sigma_X(2)}(X) \leq ... \leq C_{\sigma_X(n)}(X)$;
3. izračuna se skalarni proizvod vektora $\Lambda = (\lambda_1,..,\lambda_n)$, sa vektorom $C_{\leq} = (C_{\sigma_X(1)}(X), C_{\sigma_X(2)}(X),..., C_{\sigma_X(n)}(X))$, čime je obezbeđeno da se koeficijenat λ_i primeni na *i*-tu najnižu cenu snabdevanja korisnika $C_{\sigma_X(i)}(X)$ za svako *i=1,..,n* .

Koristeći prethodno uvedene oznake DOMP se formuliše na sledeći način:

$$\min_{X \subseteq I, |X|=p} \sum_{i=1}^{n} \lambda_i C_{\sigma_X(i)}(X).$$

Očigledno, ra različite izbore vektora Λ dobijaju se razne vrste funkcije cilja, tj. različiti problemi.

Na primer, postavljanje $\Lambda = (1,1,..,1),\ \Lambda = (0,0,..,1)$ ili na $\Lambda = (\mu,\mu,..,\mu,1),\ 0 < \mu < 1,$ dovodi do poznatih problema p-medijana, p-centar, μ-cendiana. Pored ovih, i drugi značajni diskretni lokacijski problemi mogu biti izvedeni iz DOMP-a: k-centar, (k_1+ k_2) odsečena sredina, minimizacija sume cena k1 najjeftinijih i k2 najskupljih, minimizacija sume cena najjeftinijh i najskupljih (pomnožena sa 2), ignorisanje ostalih, itd. Naravno, iz ovog modela mogu se generisati i neke nove funkcije cilja, a samim tim i novi problemi.

Metode rešavanja problema

DOMP je uveden u radu [Nickel01]. U radu [Doming03a] s predložena dva heuristička pristupa za rešavanja DOMP: Pretraga promenljivih okolina (eng. Variable Neighborhood Search - VNS) i evolutivno programiranje – EP. Obe heuristike su testirane na standardnim ORLIB instancama p-medijane, sa brojem čvorova do *n*=900.

Metoda VNS je korišćena i za obezbeđivanje kvalitetne gornje granice za tačnu metodu grananja i ograničavanja BnB, koja je opisana u u radu [Boland03] i koja je primenjena na instancama do *n*=30 čvorova.

Instance problema

U eksperimentu će se vršiti izvršavanja EA nad istim instancama nad kojima su se izvršavali i prethodno opisani metodi za rešavanje DOMP-a. To su instance p-medijane (ukupno 40 njih) i one su preuzete iz ORLIB-a (videti [BeaslJ90], [BeaslJ96]).

EA implementacija HGA1

Osnovna shema za oba algoritma (tj. za obe EA implementacije – i za HGA1 i za HGA2) može biti predstavljena sledećim pseudokodom:

```
begin
 Input_Data();
 Population_Init();
 while not Finish() do
   for i:=1 to Npop do
     pi := Objective_Function(i);
   end
   if p1<>p1,prev then
     Improving_Heuristic (p1);
   end
   Fitness_Function();
   Selection();
   Crossover();
   Mutation();
 end
 Output_Data();
end
```

algoritam 5.1. – Struktura HGA1 i HGA2

U gornjem algoritmu, `Npop` predstavlja broj jedinki u populaciji, `pi` je vrednost *i*-te jedinke, a `p1,prev` je vrednost najbolje jedinke u prethodoj generaciji.

Pri dizajnu HGA1 evolutivnog algoritma, donesene su sledeće odluke (videti [Stan06]):

- HGA1 koristi binarnu reprezentaciju jedinki. Svaka jedinica u genetskom kodu označava da je u odgovarajućem čvoru uspostavljen snabdevač, dok nula označava da odgovarajući čvor mreže nije snabdevač, već korisnik. Korisnici se, s obzirom da nema ograničenja za kapacitet, usmeravaju prema najbližem snabdevaču.
- Da bi se dobile korektne jedinke prilikom oformljenja populacije, svaka jedinka inicijalne populacije se generiše korišćenjem iste strategije koja je korišćena pri rešavanju UMApHMP i UMApHCP :svaki bit u genetskom genu se postavlja pseudoslučajno i dobija vrednost 1 sa verovatnoćom p/n , a potom se nekorektne jednike koriguju dodavanjem (odnosno brisanjem) potrebnog broja jedinica sa kraja genetskog koda.
    1. Odluka o strukturi podataka, tj. rešenja je izuzetno značajna za uspešno kreiranje brze procedure evaluacije jedinke. . Pored matrice cena tansporta *C*, HGA1 implementacija koristi i niz indeksa snadevača i za svakog korisnika je taj niz sortiran u neopadajući poedak po ceni transporta od snabdevača do tog korisnika.
- Na osnovu istog razamtranja kao i u prethodnim implementacijama, odlučeno je da operator selekcije kod HGA1 implementacije bude FGTS, sa vrednošću parametra 5.4.
- HGA1 implementacija koristi specijalizovani operator ukrštanja , koji očuvava korektnost jedinki, a koji je detaljno opisan u opisu EA implementacije za rešavanje problema UMApHMP. I u ovoj EA implementaciji ukrštanje biva izvršavano sa nivoom 0.85.
- Operator mutacije je realizovan na potpuno isti način kao kod UMApHMP i kod UMApHCP: određuju se zamrznuti geni i nivo mutacije za zamrznute gene ($1.0/n$) se postavlja ne vrednost dva i po puta veću od novoa mutacije ostalih gena ($0.4/n$).Potom se ujednačava broja mutiranih jedinica i nula, čime operator mutacije očuvava broj uspostavljenih habova, tj. očuvava korektnost mutirane jedinke.
- Performanse metoda su i u ovoj implementaciji poboljšane keširanjem EA i GFI (videti [Kratic00], [Stan06]) sa kešom veličine 5000.
- Heuristika brze razmene, (eng. fast interchange - FI) se pokazala kao jako pogodna za hibridizaciju pri rešavanju problema p-centra i p-medijane (videti [Hansen97], [Resen03]). . Međutim, ta heuristika ne može biti primenjena na DOMP bez modifikacije. Ovde se predlaže uvođenje generalizovane heuristike brze razmene (GFI) kako bi hibridizacijom sa tom heuristikom bio popravljen kvalitet dobijenog rešenja.

  GFI (detaljno opisana u [Stan06]) se primenjuje na najbolju jedinku u tekućoj generacji EA, ako je ona promenjena u poređenju sa najboljom jedinkom prethodne generacije. Heuristika pokušava da razmeni jednu otvorenu lokaciju (tj. snabdevača) i jednu zatvorenu (tj.korisnika). Proces razmene lokacija se izvršava kada se uvidi da je razmena popravila rešenje. Potom se taj korak ponavlja na novoj, popravljenoj jedinki sve dok ona ne ostane nepromenjena u dva uzastopna koraka.

- Opet je veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija. Dve trećine populacije direktno prelaze u novu generaciju (elitne jedinke) i za njih se prilagođenost ne treba ponovo izračunavati.
- U svakoj generaciji EA, duplikati se uklanjaju iz populacije. To se i ovde radi postavljanjem prilagođenosti duplikata na nulu, tako da ih sledeća primena selekcije automatski izbacuje.
- Jedinke koje imaju istu prilagođenost, ali različite genetske kodove u nekim slučajevima mogu dominirati populacijom. I ovde se, radi izbegavanja takvih situacija, ograničava (na 40) broj jedinki koje imaju istu prilagođenost, a različite genetske kodove.
- U ovoj implementaciji se izvršavanje EA zaustavlja posle 5000 generacija, odnosno, ako se prilagođenost najbolje jednike nije popravila tokom 2000 generacija.

EA implementacija HGA2

U literaturi se često nailazi na preporuku da kod DOMP, p-medijane, p-centra i sličnih lokacijskih problema binarno kodiranje ne daje najbolje rezultate (videti [Drezn02]). U tom radu autori sugerišu enkodiranje bazirano na niskama dužine p, gde brojevi (geni) odgovaraju indeksima uspostavljenih snabdevača.

Shodno stavovima ove preporuke, u radu [Stan06] je opisana implementacija još jednog EA metoda za rešavanje DOMP, nazvanog HGA2. HGA2 implementacija ima shemu rada opisanu algoritmom 5.1.(istu kao i HGA1). Ova nova implementacija ima sledeće osobine:

- Genetski kod svake jedinke se sastoji od p brojeva iz skupa $\{1, 2 ..., n\}$, koji predstavljaju redne brojeve uspostavljenih snabdevača.
  Dupliranje rednih brojeva snabdevača rešava se na sledeći način: ako se indeks snabdevača već pojavljuje u genetskom kodu, uzima se prvi veći broj koji se ne pojavljuje, a ako name većeg onda se uzima prvi manji (s obzirom da je $n>p$, mora se naći neki takav indeks – bilo veći bilo manji.
- Evaluacija jedinke se kod HGA2 vrši isto kao što se vršila kod HGA1, uz neke minimalne tehničke razlike koje proizilaze iz promenjene reprezentacije.
- I kod HGA2 je korišćena heuristika GFI, kao što je bio slučaj i kod HGA1. Naravno, implementacija je prilagođena novodefinsanoj genetskoj reprezentaciji i detaljno opisana u [Stan06].
- Kod HGA2 implementacije je FGTS operator selekcije, sa vrednošću parametra 5.4.
- Kao operator ukrštanja ovde se primenjuje jednopoziciono ukrštanje, sa verovatnoćom primene 0.85.
- HGA2 koristi prostu mutaciju sa zamrznutim genima, gde je nivo mutacije $0.4/n$ za nezamrznute, a $1.0/n$ za zamrznute gene.
- Opet je veličina populacije je 150 jedinki. Pri izvršavanju se koristi stacionarna verzija algoritma, kao i elitna strategija, kao što je bio slučaj kod HGA1 implementacije .
- I ovde se u svakoj generaciji EA, duplikati se uklanjaju iz populacije (postavljanjem prilagođenosti duplikata na nulu).
- Kod HGA2 implementacije se, takođe, , ograničava (na 40) broj jedinki koje imaju istu prilagođenost, a različite genetske kodove.
- U ovoj implementaciji se izvršavanje EA zaustavlja posle 5000 generacija, odnosno, ako se prilagođenost najbolje jednike nije popravila tokom 2000 generacija.
- Performanse metoda su i u ovoj implementaciji poboljšane keširanjem EA i GFI (videti [Kratic99], [Stan06]) sa kešom veličine 5000.

Eksperimentalni rezultati pri poređenju algoritama za DOMP

U ovom poglavlju se HGA1 i HGA2 i porede sa postojećim algoritmima koji rešavaju DOMP. Sva izvršavanja HGA1 i HGA2 su realizovana na računaru sa AMD K7 procesorom na 1.33GHz, koji ima 256MB operativne memorije.

Algoritam je, pri testiranju HGA1 i HGA2 izvršavan 20 puta na svakoj instanci (videti [Stan06]).

U tabelama 5.36. i 5.31. prikazani su, jedni pored drugih, rezultati koje su, na instancama problema p-medijane, rešavajući varijante T1 i T4 DOMP-a, postigli HGA1, HGA2, EP i VNS metodi.

Rezultati posledenja dva metoda, preuzeti iz [Doming03a], su dobijeni izvršavanjem na Intel Pentium III računaru brzine 800MHz.

Kolone u tabelama imaju sledeća značenja:

- prva kolona sadrži oznaku instance
- sledeće dve kolone sadrže parametre instance *n* i *p*;
- kolone **Naj. reš.** i **AMD 1.33 GHz** sadrže najbolje dobijeno rešenje i vreme izvršavanja HGA1;
- sledeće dve kolone **Naj. reš.** i **AMD 1.33 GHz** sadrže najbolje dobijeno rešenje i vreme izvršavanja HGA2;
- sledeće dve kolone **Naj. reš.** i **PIII 0.8 GHz** sadrže najbolje dobijeno rešenje i vreme izvršavanja EP;
- poslednje dve kolone **Naj. reš.** i **PIII 0.8 GHz** sadrže najbolje dobijeno rešenje i vreme izvršavanja VNS.

Najbolji rezultati po instancama (tj. situacije gde je rešenje koje je dobio konkretan algoritam bolje od svih ostalih alternativa) su podebljani.

Inst.	n	p	Opt.	HGA1		HGA2		EP		VNS	
				Naj. reš.	AMD 1.33 GHz	Naj. reš.	AMD 1.33 GHz	Naj. reš.	PIII/0.8G Hz	Naj. reš.	PIII/0.8G Hz
pmed1	100	5	5819	opt	2.919	opt	1.122	opt	25.42	opt	1.19
pmed2	100	10	4093	opt	3.379	opt	2.254	opt	37.55	opt	2.97
pmed3	100	10	4250	opt	3.384	opt	2.625	opt	37.88	opt	3.00
pmed4	100	20	3034	opt	4.715	opt	4.616	3046	61.48	3046	5.98
pmed5	100	33	1355	opt	5.722	opt	7.332	1361	93.22	1358	6.81
pmed6	200	5	7824	opt	8.615	opt	2.083	opt	36.25	opt	7.95
pmed7	200	10	5631	**opt**	15.590	5639	4.987	5645	55.39	5639	12.72
pmed8	200	20	4445	**opt**	11.605	4522	5.567	4465	91.81	4457	21.05
pmed9	200	40	2734	**2740**	14.432	2781	14.073	2762	170.25	2753	41.98
pmed10	200	67	1255	**1256**	21.763	1268	26.610	1277	290.53	1259	72.22
pmed11	300	5	7696	opt	15.221	opt	3.249	opt	47.98	opt	12.52
pmed12	300	10	6634	opt	22.852	opt	7.526	opt	75.63	opt	26.02
pmed13	300	30	4374	4381	22.393	opt	22.669	4432	193.22	**opt**	87.92
pmed14	300	60	2968	2969	34.192	2972	50.664	2997	359.58	2969	241.95
pmed15	300	100	1729	1735	56.106	**1734**	81.088	1749	580.98	1739	363.39
pmed16	400	5	8162	opt	28.152	opt	5.440	8183	56.89	opt	24.36
pmed17	400	10	6999	opt	47.575	opt	14.539	opt	95.08	opt	47.30
pmed18	400	40	4809	**opt**	44.661	4812	54.752	4880	320.38	4811	275.69
pmed19	400	80	2845	**2848**	79.621	2862	112.610	2891	604.36	2864	469.30
pmed20	400	133	1789	1794	138.629	1794	156.281	1832	963.44	**1790**	915.17
pmed21	500	5	9138	opt	34.118	opt	6.596	opt	70.14	opt	27.39
pmed22	500	10	8579	**opt**	64.166	8662	17.587	8669	116.59	8669	64.25
pmed23	500	50	4619	4624	75.542	opt	104.446	4651	486.08	opt	443.23
pmed24	500	100	2961	**2966**	192.806	2974	206.212	3009	924.66	2967	1382.84
pmed25	500	167	1828	1835	254.447	1835	274.420	1890	1484.13	1841	2297.25
pmed26	600	5	9917	opt	52.265	opt	13.212	9919	84.34	opt	48.45
pmed27	600	10	8307	opt	87.084	opt	28.942	8330	136.53	8310	127.63
pmed28	600	60	4498	**4500**	161.932	4505	172.558	4573	673.30	4508	965.48
pmed29	600	120	3033	3036	273.325	3040	347.686	3099	1268.89	3036	2758.56
pmed30	600	200	1989	**2002**	651.279	2012	668.467	2036	2403.33	2009	3002.34
pmed31	700	5	10086	opt	60.908	opt	14.603	opt	92.67	opt	56.02
pmed32	700	10	9297	opt	122.546	opt	28.240	9319	156.50	9301	165.27
pmed33	700	70	4700	4719	206.433	4713	376.866	4781	894.19	**4705**	2311.03
pmed34	700	140	3013	3023	485.249	**3022**	819.587	3100	1762.69	3024	5384.19
pmed35	800	5	10400	opt	74.310	opt	21.280	opt	109.86	opt	88.50
pmed36	800	10	9934	9951	133.443	opt	50.990	9947	182.06	opt	200.97
pmed37	800	80	5057	**5063**	242.446	5070	182.110	5126	1190.25	5066	2830.30
pmed38	900	5	11060	opt	117.790	opt	26.950	opt	120.14	opt	150.53
pmed39	900	10	9423	opt	164.910	opt	66.972	opt	207.75	opt	200.73
pmed40	900	90	5128	5134	766.518	**5132**	876.836	5188	1492.59	5141	4774.38

tabela 5.36. Poređenje rezultata na T1 (p-medijana)

Iz podataka u prethodnoj tabeli se primećuje da su (od 40 instanci problema) HGA1 i HGA2 implementacije 21 put dosegle optimalno rešenje, da je EP kod 12 instanci stigao do optimalnog rešenja, a kod VNS je pri rešavanju 17 instanci problema dobijeno rešenje jednako optimalnom.

Ako se posmatraju broj instanci na kojima je jedna implementacija nadmašila ostale (ti elementi su podebljani), vidimo da je HGA1 to postigao 11 puta, HGA2 i VNS 3 puta, a EP nije ni jednom

Inst.	n	p	Best known	HGA1		HGA2		EP		VNS	
				Naj. reš.	AMD 1.33 GHz	Naj. reš.	AMD 1.33 GHz	Naj. reš.	PIII/0.8GHz	Naj. reš.	PIII/0.8GHz
pmed1	100	5	4523	best	2.870	best	1.364	best	25.20	best	1.27
pmed2	100	10	2987	best	3.385	best	2.383	2993	36.98	best	3.80
pmed3	100	10	3067	best	3.351	best	2.304	best	36.91	3074	2.80
pmed4	100	20	2137	best	5.128	best	4.740	2153	60.80	2142	6.98
pmed5	100	33	818	best	4.837	best	6.393	839	92.08	best	8.22
pmed6	200	5	6064	best	8.754	best	2.354	best	35.52	6078	7.88
pmed7	200	10	4206	best	11.766	best	4.006	4225	54.17	best	13.41
pmed8	200	20	3182	best	12.202	3184	5.845	3248	91.95	best	28.30
pmed9	200	40	1807	**best**	18.481	1827	11.737	1831	167.61	1816	66.39
pmed10	200	67	818	**best**	27.905	823	26.556	849	274.09	829	75.91
pmed11	300	5	5979	best	15.444	best	3.261	best	47.75	best	13.30
pmed12	300	10	5021	best	21.057	best	6.856	best	73.83	best	25.86
pmed13	300	30	3133	best	22.712	best	21.066	3175	183.25	best	97.80
pmed14	300	60	1949	**best**	50.271	1951	50.553	2027	346.42	1957	303.64
pmed15	300	100	1133	1134	83.866	1137	82.541	1181	549.67	**best**	415.80
pmed16	400	5	6341	best	24.237	best	4.465	best	56.06	best	24.13
pmed17	400	10	5381	best	37.933	best	12.865	5440	89.30	5413	43.83
pmed18	400	40	3437	best	54.822	best	43.746	3463	309.50	3443	261.86
pmed19	400	80	1922	1926	63.999	**best**	128.841	1973	618.88	1933	779.77
pmed20	400	133	1146	**best**	201.458	1154	175.917	1191	1000.41	1152	1108.48
pmed21	500	5	7245	best	33.997	best	6.264	best	71.69	best	24.22
pmed22	500	10	6685	best	63.664	best	18.153	6749	117.88	6722	58.58
pmed23	500	50	3306	3307	82.941	3307	117.958	3379	461.50	**best**	639.95
pmed24	500	100	2004	2005	131.277	**best**	311.142	2068	888.27	2005	1455.81
pmed25	500	167	1149	**best**	537.011	1155	369.609	1198	1524.86	1151	2552.02
pmed26	600	5	7787	best	46.058	best	9.659	7789	87.30	best	48.11
pmed27	600	10	6444	best	85.349	best	21.993	6481	141.97	best	141.70
pmed28	600	60	3202	**best**	146.599	3206	182.158	3304	687.42	3210	1113.89
pmed29	600	120	2006	best	205.499	best	336.494	2087	1249.78	best	3178.69
pmed30	600	200	1295	**best**	759.281	1297	654.791	1359	1976.77	1308	4942.75
pmed31	700	5	8046	best	71.425	best	17.389	8047	90.81	best	66.16
pmed32	700	10	7278	best	111.564	best	28.208	7318	148.77	7280	162.97
pmed33	700	70	3405	**best**	216.540	3407	236.516	3463	857.47	3413	2377.72
pmed34	700	140	2023	2033	681.274	2036	642.381	2083	1624.61	**best**	5657.56
pmed35	800	5	8191	best	83.567	best	22.164	best	102.58	best	72.58
pmed36	800	10	7796	**best**	198.654	7820	57.702	7840	170.38	7820	201.64
pmed37	800	80	3600	**best**	358.066	3611	181.476	3684	1086.13	3604	3170.70
pmed38	900	5	8720	best	102.448	8766	25.919	8768	111.61	best	140.84
pmed39	900	10	7360	best	170.158	best	69.272	7398	189.19	best	313.03
pmed40	900	90	3710	best	693.154	best	612.117	3768	1372.98	3718	5422.73

tabela 5.37. Poređenje rezultata na T4 (k_1+ k_2) odsečena sredina

Analizirajući podatke iz tabele 5.37. uočava se da je (od 40 instanci problema) HGA1 35 puta dostigla najbolje poznato rešenje, HGA2 implementacija 25 puta, EP 8 puta, a VNS je pri rešavanju 20 instanci problema dobio najbolje rešenje.

Ako se posmatraju broj instanci na kojima je jedna implementacija nadmašila ostale (ti elementi su podebljani), vidimo da je HGA1 to postigao 10 puta, HGA2 dva puta, VNS 3 puta, a EP nije ni jednom.

6. ZAKLJUČAK

U radu su predložena poboljšanja klasičnog operatora turnirske selekcije. Za predložene operatore izvode se odgovarajuće teorijske procene i empirijska poređenja. Novi operatori, FGTS i DFGTS, imaju sve dobre osobine klasične turnirske selekcije, s tim da se može postaviti finiji tj. precizniji odnos između eksploatacije i istraživanja.
Rad sadrži pet velikih celina. U prvoj, uvodnom poglavlju se definišu osnovni pojmovi, opisuju evolutivni algoritmi (istorijat, oblasti primene, struktura, osobine i modifikacije), a potom se uvode i osnovni pojmovi vezani za paralelno procesiranje, evolutivni algoritmi i postojeći softverski sistemi za implementciju evolutivnih algoritama (kako sekvencijalnih, tako i paralelnih).
Druga celina sadrži nekoliko metodologija teorijske analize evolutivnih algoritama (sekvencijalnih i paralelnih) i pojedinih njegovih aspekata. Preciznije, opisane su metodologija teorijske analize koja se oslanja na sheme i na gradivne blokove, te metodologija koja izvođenje tvrđenja teorije evolutivnih algoritama zasniva na Markovljevim lancima. Tu su takođe opisani i sledeći aspekti analize evolutivnih algoritama: obmanjivački problemi, preuranjena konvergencija, kolateralna konvergencija i pejzaži prilagođenosti,
U trećoj celini su opisane veličine koje kvantifikuju kvalitet operatora selekcije. Tu su detaljno opisani najvažniji operatori selekcije i migracije, te predložene nove alternative i poboljšanja ovih operatora. Operatori (i klasični i novi) su analizirani u svetlu prethodno definisanih metodologija. Za novodefinisane operatore su izvedeni izrazi za sve prethodno opisane mere kvaliteta operatora selekcije.
Četvrta celina opisuje EA sisteme koje je razvijao autor, kako one dominantno posvećene rešavanju NP teških problema, tako i novorazvijeni EA web servis i aplikacije koje ga koriste. Opisana je motivacija, karakteristike, domen korišćenja i arhitektura ovih EA sistema.
Peta celina sadrži metodologije empirijskog poređenja raznih evolutivnih algoritama, kao i rezultate poređenja kod tako razvijene metodologije i kod konkretnih realnih problema. Pri tome je posebna pažnja posvećena dizajnu evolutivnih algoritama za rešavanje NP teških problema koji po svojim karakteristikama (u trenutku publikovanja) prevazilaze druge pristupe rešavanju tih problema.
Postojeće rezultate je moguće proširiti i unaprediti u nekoliko pravaca:

- Dizajnirati znatno složeniji operator turnirske selekcije kod kojeg, pored željene srednje vrednosti veličine turnira, selekcioni metod zavisi i od prostora prilagođenosti u kojem se jedinke nalaze.
- Hibridizacija EA sa drugim metaheuristikama i sa egzaktnim metodama rešavanja.
- Izvršavanje EA implementacije na LAN/WAN mrežama sa velikim brojem računara.

6.1. Naučni doprinos rada

Neki od najvažnijih novih rezultata prikazanih u ovom radu su:

- Programski kod sekvencijalne i paralelne EA implementacije (EA Web Servis).
- EA implementacija korišćenjem web servisa je osmišljena tako da omogući efikasno izvršavanje algoritma u Intetnet okruženju.
- Dizajn i teoretska procena operatora selekcije FGTS i DFGTS sa stanovišta teoreme o shemama i Markovljevih lanaca.
- Empirijsko poređenje novodizajniranih operatora sa drugim operatorima selekcije na De Jongovom skupu test-funkcija.
- Teorijska procena i empirjsko poređenje raznovrsnih politika migracije jedinki između potpopulacija.
- Efikasna implementacija novodizajniranih selekcionih metoda i njihovo uklapanje u GANP PGANP.
- Uspešna primena novodizajniranih operatora pri rešavanju nekoliko realnih NP-teških problema. Pri rešavanju standardnih instanci razmatranih problema, EA koji koristi novodizajnirane operatore daje bolje rezultate u odnosu na druge metode rešavanja problema.

Osim naučnog doprinosa, u radu su vrlo važni i stručni aspekti, tj. primena modernih programskih paradigmi:

- Direktno korišćenje XML standarda u okviru programske implementacije.
- Softver je implementiran u programskom jeziku C#, korišćenjem .NET okvira.
- Arhitektura softvera, kao što je to slučaj u modernim softverskim rešenjima informacionih sistema velikih kompanija, počiva na objektno-orjentisanom pristupu i na šablonima dizajna.
- Softver uključuje i predefinisane komponente, tzv. aplikativne blokove.

Posle prikaza svih osobina operatora selekcije i migracije, kao i rezultata sekvencijalne i paralelne EA implementacije, može se izvesti zaključak da one predstavljaju moćno sredstvo za rešavanje složenih problema optimizacije.

Naučno istraživanje prikazano u ovom radu predstavlja doprinos oblastima kombinatorne optimizacije, evolutivnih algoritama, paralelnih algoritama i web programiranja.

Deo dobijenih rezultata je već objavljen u inostranim i domaćim časopisima, a ostali delovi su u pripremi za objavljivanje.

LITERATURA

[Abdinn98] Abdinnour-Helm S, Venkataramanan M. A.: Solution Approaches to Hub Location Problems, *Annals of Operations Research*, Vol. 78, pp. 31-50, 1998.

[Abdinn98a] Abdinnour-Helm S.: A hybrid heuristic for the uncapacitated hub location problem, . *European Journal of Operational Research*, Vol. 106, pp. 489-499, Elsevier, 1998.

[Aizawa93] Aizawa A. N, Wah B. W.: Dynamic Control of Genetic Algorithms in a Noisy Environment, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 48-55, Morgan Kaufmann, San Mateo, CA, 1993.

[Akl89] Akl S.: *Design and Analysis of Parallel Algorithms*, Prentice-Hall International, Englewoods Cliffs, NJ, 1989.

[Alande95] 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*

[Alba99] Alba E, Troya J.M.: A Survey of Parallel Distributed Genetic Algorithms, *Complexity*, Vol. 4, pp. 31-52, 1999.

[Alba02] Alba E, Cotta C, Chicano F, Nebro A.J.: Parallel Evolutionary Algorithms in Telecommunications: Two Case Studies, *Proceedings of the CACIC02*, 2002.

[Alba04] Alba E, Chicano F.: Solving the Error Correcting Code Problem with Parallel Hybrid Heuristics, *Proceedings of the 2004 ACM Symposium on Applied Computing*, pp. 985-989, Nicosia, Cyprus, 14-17 mart 2004.

[Alba04a] Alba E, Luque G.: Growth Curves and Takeover Time in Distributed Evolutionary Algorithms, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, WA, USA, 26-30.06.2004, *Lecture Notes in Computer Science*, Vol. 3102, pp. 864-876, 2004.

[Alves92] Alves M.L, Almeida M.T.: Simulated Annealing Algorithm for the Simple Plant Location Problem: A Computational Study, *Revista Investigação*, Vol. 12, 1992.

[Ander02] Anderson R, Francis B, Homer A.: *Proffesional ASP .NET 1.0*, Wrox Press, Birmingham, 2002.

[Antoni89] Antonisse J.: A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 86-91, Morgan Kaufmann, San Mateo, CA, 1989.

[Archer01] Archer T.: *Inside C#*, Microsoft Press, Redmond, Washington, 2001.

[Archer02] Archer T, Chapet W.: *Inside C# 2*, Microsoft Press, Redmond, Washington, 2002.

[Aykin95] Aykin T.: Networking Policies for Hub-and-spoke Systems with Application to the Air Transportation System, *Transportation Science 29*, pp. 201-221, 1995.

[Baarts00] Baartse M, Blair R, Bolognese L, Dalvi D, Hahn S, Haines C, Homer A.: *Proffesional ASP XML*, Wrox Press, Birmingham, 2000.

[Balakr89] 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.

[Baluja93] Baluja S.: Structure and Performance of Fine-Grain Parallelism in Genetic Search, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 155-162, Morgan Kaufmann, San Mateo, CA, 1993.

[Battle93] Battle D, Vose M. D. Isomorphisms of genetic algorithms, *Artificial Intelligence 60*, pp. 155-165, 1993.

[Bauer84] Bauer R. J, Jr.: *Genetic Algorithms and Investment Strategies*, John Wiley & Sons, Inc, New York, 1984.

[Bayer91] Bayer S.E, Wang L.: A genetic algorithm programming environment: Splicer, *Proceedings of the 1991 IEEE International Conference on Tools with Artificial Intelligence - TAI'91*, pp. 138-144, IEEE Computer Society Press, Los Alamitos, 1991.

[Bäck91] Bäck T, Hoffmeister F, Schwefel H-P.: A Survey of Evolution Strategies, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*,pp. 2-9, Morgan Kaufmann, San Mateo, CA, 1991.

[Bäck91a] Bäck T, Hoffmeister F.: Extended Selection Mechanisms in Genetic Algorithms, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91,* pp. 92-99, Morgan Kaufmann, San Mateo, CA, 1991.

[Bäck92] Bäck T.: Self-adaptation in Genetic Algorithms, *Proceedings of the First European Conference on Artificial Life*, MIT Press, 1992.

[Bäck92a] Bäck T.: Optimization by Means of Genetic Algorithms, *Internal report,* Department of Computer Science, University of Dortmund, 1992.

[Bäck92b] Bäck T: A User's Guide to GENEsYs 1.0, *Internal report,* Department of Computer Science, University of Dortmund, 1992.

[Bäck93] Bäck T.:Optimal Mutation Rates in Genetic Search, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 2-8,. Morgan Kaufmann, San Mateo, CA, 1993

[Bäck94] Bäck T.: Selective Pressure in Evolutionary Algorithms: A Caracterisation of Selection

Mechanisms, *Proceedings of the First IEEE Conference on Evolutionary Computation - ICEC '94*, pp. 57-62, 1994.

[Bäck95] Bäck T.: Generalized Convergence Models for Tournament and (μ,λ)-Selection, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 2-8, *Morgan Kaufmann,* San Mateo, CA, 1995.

[Bäck96] Bäck T.: *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[Bäck00] Bäck T, Fogel D.B, Michalewicz Z.: *Evolutionary Computation 1: Basic Algorithms and Operato*rs, Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[Bäck00a] Bäck T, Fogel D.B, Michalewicz Z.: *Evolutionary Computation 1: Advanced Algorithms and Operators*, Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[BeaslJ90] Beasley J.E.: OR Library: distributing test problems by electronic mail, *Journal of the Operational Research Society*, Vol. 41, pp. 1069-1072, 1990.

[BeaslJ96] 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
http://www.brunel.ac.uk/depts/ma/research/jeb/orlib

[Beldin95] Belding T. C.: The Distributed Genetic Algorithm Revisited, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 114-121, Morgan Kaufmann, San Mateo, CA, 1995.

[Bell04] Bell A.: Deth by UML Feaver, *ACM Queue,* Vol. 2 No. 1 5,. 2004.

[Berton93] Bertoni A, Doringo M.: Implicit parallelism in genetic algorithms, *Artificial Intelligence 61*, pp. 307-314, 1993.

[Bertse89] Bertsekas D.P, Tsitsiklis J.N.: *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall International, 1989.

[Blickl95] Blickle T, Thiele L.: A Mathematical Analysis of Tournament Selection, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 9-16, Morgan Kaufmann, San Mateo, CA, 1995.

[Blickl95a] Blickle T, Thiele L.: A Comparison of Selection Schemes Used in Genetic Algorithms, *Internal report (2. edition)*, Swiss Federal Institute of Technology, 1995.

[Blickl97] Blickle T, Thiele L.: A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation, 4(4)*, pp. 361–394, 1997.

[Boland03] Boland N, Domínguez-Marin P, Nickel S, Puerto J.: Exact Procedures for Solving the Discrete Ordered Median Problem, *ITWM Bericht*, Vol. 47, Fraunhofer Institut für Tecno-und Wirtschaftsmathematik (ITWM), Kaiserslautern, Germany, 2003.

[Boland04] Boland N, Krishnamoorty M, Ernst A. T, Ebery, J.: Preprocessing and Cutting for Multiple Allocation Hub Location Problems. *European Journal of Operational Research*, Vol. 155, pp. 638-653, 2004

[Bologn02] Bolognese L.: *Designing Data Tier Components and Passing Data Through Tiers*, Microsoft Patterns&Practices, Microsoft press 2002.

[Booch99] Booch, G, Rumbaugh J, Jacobson I.: *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, Massachusetts, 1999.

[Bott02] Bott, G: *Operating .NET Framework Based Applications*, Microsoft Patterns&Practices, Microsoft press, 2002.

[Branke04] Branke J, Kamper A, Schmeck H.: Distribution of Evolutionary Algorithms in Heterogeneous Networks, *Genetic and Evolutionary Computation Conference – GECCO 2004*, Seattle, WA, USA, 26-30.06.2004, *Lecture Notes in Computer Science*, Vol. 3102, pp. 923-934, 2004.

[Bui95] Bui T. N, Moon B. R.: On Multi-Dimensional Encoding/Crossover, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 49-56, Morgan Kaufmann, San Mateo, CA, 1995.

[Bull94] Bull L, Fogarty T. C.: An Evolution Strategy and Genetic Algorithms Hybrid: an Initial Implementation and First Results, *Evolutionary Computing*, pp. 95-102, AISB Workshop Leeds, 1994.

[Burke95] Burke E. K, Elliman D. G, Weare R. F.: A Hybrid Genetic Algorithm for Higly Constrained Timetabling Problems, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 605-610, Morgan Kaufmann, San Mateo, CA, 1995.

[Buschm96] Buschmann F.: *Pattern-Oriented Software Architecture, Vol 1*, Wiley & Sons, 1996.

[Butler94] 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

[Bücker04] Bücker H.M, Rasch A, Wolf A.: A Class of OpenMP Applications Involving Nested Parallelism, *Proceedings of the 2004 ACM Symposium on Applied Computing*, pp. 220-224, Nicosia, Cyprus, 14-17 mart 2004.

[Canov04] Canovas L, Garcıa S, Marın A.: Solving the Uncapacitated Multiple Allocation Hub Location Problem by Means of a Dualascent Technique, *Working paper no 1*, Departamento de Estadısticae Investigacion Operativa, University of Murcia, Spain, 2004.
http://www.optimization-online.org/
DB FILE/2004/01/812.pdf

[Cantu94] Cantu-Paz E.: Distributed GENESIS User's Guide - Version 1.0 *Internal report*, Instituto Tecnologico Autonomo de Mexico, 1994.

[Cantu97] Cantu-Paz E.: A Survey of Parallel Genetic Algorithms, *IlliEAL Report No 97003*, *Revised version of IlliEAL Report No 95007*, Illinois Genetic Algorithms Laboratory, 1997.

[Cantu00] Cantu-Paz E.: *Efficient and Accurate Parallel Genetic Alghorithms*, Kluwer Academic Publishers, Boston, 2000.

[Caprar95] 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.

[Caprar95a] 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*

[Caruana91] Caruana R. A, Eshelman L. J, Schaffer D. J.: Representation and Hidden Bias II: Eliminating Defining Lenght Bias in Genetic Search via Shuffle Crossover, *Machine Learning*, pp. 750-755, 1991.

[Chamb95] Chambers D.L.*: Practical Handbook of Genetic Algorithms, Volume 2: New Frontiers*, CRC Press, Boca Raton, 1995.

[Chamb99] Chambers D.L.: *Practical Handbook of Genetic Algorithms, Volume 3: Complex Coding Systems*, CRC Press, Boca Raton, 1999.

[Chamb01] Chambers D.L.: *Practical Handbook of Genetic Algorithms, Volume 1: Applications*, Second edition, Chapman & Hall/CRC, Boca Raton, 2001.

[Chandra01] Chandra R, Dagum L, Kohr D, Maydan D, McDonald J, Menon R.: *Parallel Programming in OpenMP*, Morgan Kaufmann, San Francisco, CA 2001.

[Cheng05] Cheng F-H.: A hybrid heuristic for the uncapacitated single allocation hub location problem, *OMEGA - The International Journal of Management Science*, Elsevier, , u štampi.

[ChenH98] Chen H, Flann N.S, Watson D.W.: Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 2, pp. 126-136, 1998.

[CHI91] *CHIMP Concepts*, Edinburgh Parallel Computing Centre, University of Edinburgh, UK, 1991.

[CHI92] *CHIMP Version 1.0 Interface*, Edinburgh Parallel Computing Centre, University of Edinburgh, UK, 1992.

[Chong98] Chong F. S.: A Java based Distributed Approach to Genetic Programming on the Internet, *MS disertation*, University of Birmingham, School of Computer Science, UK, 1998.

[Chong99] Chong F. S.: Java based Distributed Genetic Programming on the Internet, *Technical report CS-RP-99-7*, University of Birmingham, School of Computer Science, UK, 1999.

[Chung03] Chung.-J-Y, Lin K-J, Mathieu R.G.: Web Services Computing: Advancing Software Interoerability, *Computer Volume 36 Number 10*, pp. 35-37, IEEE Computer Society, October 2003.

[Cohoon91] Cohoon J. P, Martin W. N, Richards D. S.: A Multi-population Genetic Algorithm for Solving K-Partition Problem on Hyper-cubes, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 244-248, Morgan Kaufmann, San Mateo, CA, 1991.

[Cohoon91a] Cohoon J, Martin W, Richards D.: Genetic algorithms and punctuated equilibria in VLSI, *Parallel Problem Solving from Nature - PPSN,* Schwefel H.P. and Manner R. (eds.), pp. 134-144 , Springer-Verlag, Berlin, 1991.

[Darvin85] Darvin Č.: *Postanak vrsta*, Nolit, Beograd, 1985.

[Davis91] Davis L.: *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[Davis91a] Davis L.: Bit-Climbing, Representational Bias, and Test Suite Design, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 18-23, Morgan Kaufmann, San Mateo, CA, 1991.

[DeFalc96] De Falco I, Del Balio R, Della Cioppa A, Tarantino E.: A Comparative Analysis of Evolutionary Algorithms for Function Optimisation, *Internal report*, Research Institute on Parallel Information Systems, National Research Council of Italy, 1996.

[DeJong75] De Jong K.E.: An analysis of the behavior of a class of genetic adaptive systems, *PhD thesis*, University of Michigan, 1975.

[DeJong89] De Jong K. A, Spears W. M.: Using Genetic Algorithms to solve NP-Complete Problems, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 124-132, Morgan Kaufmann, San Mateo, CA, 1989.

[DeJong92] De Jong K. A.: Genetic Algorithms are NOT Function Optimizers, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 5-19, Morgan Kaufmann, San Mateo, CA, 1992.

[DeJong95] De Jong K. A, Sarma J.: On Decentralizing Selection Algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 17-25, Morgan Kaufmann, San Mateo, CA, 1995.

[DeJong95a] De Jong K. A, Spears W.: A Formal Analysis of the Role of Multi-Point Crossover in Genetic Algorithms, *Internal report*, George Mason University Fairfax, VA, 1995.

[DeLaMa93] De la Maza M, Tidor B.: An Analysis of Selection Procedures with Particular Attention Paid to Proportional and Boltzmann Sellection, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 124-131, Morgan Kaufmann, San Mateo, CA, 1993.

[Doming03] Domínguez E, Muñoz J, Mérida E.: A recurrent neural network model for the *p*-hub problem, *Lecture Notes in Computer Science* 2687, pp. 734-741, 2003.

[Doming03a] Domínguez-Marin E, Hansen P, Mlademović N, Nickel S.: Heuristic Procedures for Solving the Discrete Ordered Median Problem, *ITWM Bericht,* Vol. 46, Fraunhofer Institut für Tecno-und Wirtschaftsmathematik (ITWM), Kaiserslautern, Germany, 2003.

[Dongar93] 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.

[Dongar03] Dongarra J, Foster I, Fox G, Gropp W, Kennnedy K, Torczon L, White A.: *Sourcebook of Parallel Computing*, Morgan Kaufmann, San Francisco, CA 2003.

[Drezn02] Drezner Z, Hamacher H.: *Facility Theory: Applications and Theory*, Springer-Verlag, Berlin-Heidelberg, 2002.

[Erlen78] Erlenkotter D.: A dual-based procedure for uncapacitated facility location", *Operations Research*, Vol. 26, pp. 992-1009, 1978.

[Ernst98] Ernst A.T, Krishnamoorthy M.: An exact solution approach based on shortest-paths for p-hub median problems*, INFORMS Journal on Computing 10*, pp. 149-162, 1998.

[Ernst98a] Ernst A.T, Krishnamoorthy M.: Exact and Heuristic Algorithms for the Uncapacitated Multiple Allocation p-hub Median Problem, *European Journal of Operational Research 104*, pp. 100-112, 1998.

[Ernst04] Ernst, A.T, Hamacher H, Jiang H, Krishnamoorthy M, Woeginger G.: Uncapacitated Single and Multiple Allocation p-Hub Center Problems, *Operations Research*, 2004.

[Ernst05] Ernst A.T, Hamacher H, Jiang H, Krishnamoorthy M, Woeginger G.: Uncapacitated single and multiple allocation p-hub center problems, *Operations Research*, 2005.

[Eshelm93] Eshelman L. J, Schaffer D. J.: Crossover's Niche, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 9-14, Morgan Kaufmann, San Mateo, CA, 1993.

[Fergu02] Ferguson J, Patterson B, Beres J, Boutquin P, Gupta M.: *C# Bible,* Wiley Publishing, Inc., Indianapolis, Indiana, 2002.

[Filho94] Filho J. R, Treleaven P C, Alippi C.: Genetic-Algorithm Programming Enviroments, *Computer*, pp. 28-43, june 1994.

[Filipo96] Filipović V, Tošić D.: Comparasion of Selection Operators in Genetic Algorithms for the Function Optimization, *Abstracts of XI Conference on Applied Mathematics - PRIM '96*, pp. 94, Budva, 1996.

[Filipo96a] Filipović V, Kratica J, Radojević S, Vugdelija M.: Uticaj binarnog kodiranja na genetske algoritme za nalaženje ekstremnih vrednosti, *Zbornik radova sa X međunarodne konferencije o industrijskim sistemima*, pp. 193-198, Novi Sad, 1996.

[Filipo97] Filipović V, Tošić D, Urošević D, Kratica J.: General parallel algorithm to the solution of the geophisical inversion problem applied to the transputer system, *Proceedings of the VII Conference on Logic and Computer Science LIRA '97*, pp. A3-A8, Novi Sad, Yugoslavia, 1997.

[Filipo97a] Filipović V.: Određivanje performansi evolutivnih algoritama u teoriji i praksi, *Prolećna škola o programskim jezicima '97 - Tekstovi predavanja*, st. 131-141, Novi Sad, 1997.

[Filipo98] Filipović V.: Predlog poboljšanja operatora turnirske selekcije kod evolutivnih algoritama, *Magistarski rad*, Matematički fakultet, Beograd, 1998.

[Filipo00] Filipović V, Kratica J, Tošić D, Ljubić I.: Fine Grained Tournament Selection for the Simple Plant Location Problem, *Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications - WSC5*, pp. 152-158, September 2000,.

[Filipo01] Filipović V, Tošić D, Kratica J.: Experimental Results in Applying of Fine Grained Tournament Selection, *Proceedings of the 10th Congress of Yugoslav Mathematicians*, pp. 331-336, Belgrade, 2001.

[Filipo03] Filipović V.: Fine-grained Tournament Selection Operator in Genetic Algorithms, *Computers and Informatics (formerly Computers and Artificial Intelligence)*, Vol. 22, No. 2, pp. 143-162, 2003.

[Filipo03a] Filipović V, Tošić D, Kratica J.: Paralell Evolutionary Algoritm Web Service, *Proceedings of the XXX SYM-OP-IS 2003*, st. 292-295, Herceg Novi, 2003.

[Flynn66] Flynn M.J.: Very High-Speed Computing Systems, *Proc. IEEE*, Vol. 54, pp. 1901-1909, 1966.

[Flynn72] Flynn M.J.: Some Computer Organizations and Their Effectiveness, *IEEE Transactions on Computers*, Vol. 21, pp. 948-960, 1972.

[Fogart89] Fogarty T. C.: Varying the probability of Mutation in the Genetic Algorithmics, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 104-109, Morgan Kaufmann, San Mateo, CA, 1989.

[Fogart91] Fogarty T.C, Huang R.: Implementing the genetic algorithm on transputer based parallel processing systems, *Parallel Problem Solving from Nature - PPSN,* Schwefel H.P. and Manner R. (eds.), pp. 145-149, Springer-Verlag, Berlin, 1991.

[Fogart95] Fogarty T. C, Vavak F, Cheng P.: Use of the Genetic Algorithm for Load Balancing of Shugar Beet Presses, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 617-624, Morgan Kaufmann, San Mateo, CA, 1995.

[Forest92] Forest S, Mitchel M.: Relative Building-Block Fitness and the Building-Block Hypothesis, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 109-126, Morgan Kaufmann, San Mateo, CA, 1992.

[Fowler02] Fowler M.: *Refactoring – Improving the Design of Existing Code*, Addison - Wesley, Boston, 2002.

[Fowler03] Fowler M.: *Patterns of Enterprise Application Architecture*, Addison-Wesley, Reading, 2003.

[Gamma95] Gamma E, Helm R., Johnson R., Vlissides, J.: *Design Patterns:Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, 1995.

[Geist94] 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

[Goldbe89] Goldberg D. E.: *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company Inc, Reading, MA, 1989.

[Goldbe90] Goldberg D.E.: A Note on Boltzmann Tounament Selection for Genetic Algorithms and Population-oriented Simulated Annealing, *Complex Systems,* pp. 445-460, 1990.

[Goldbe91] Goldberg D. E, Deb K, Korb B.: Don't Worry, Be Messy, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 24-30, Morgan Kaufmann, San Mateo, CA, 1991.

[Goldbe91a] Goldberg D. E, Deb K.: A Comparative Analysis of Selection Schemes Used in Genetic Algorithm, *Foundations of Genetic Algorithms - FOGA*, pp. 69-93, edited by Rawlins B, Morgan Kaufmann, San Mateo, CA, 1991.

[Goldbe92] Goldberg D. E, Deb K, Clark J. H.: Genetic algorithms, Noise and Sizing of Population, *Complex Systems 6*, pp. 333-362, 1992.

[Goldbe94] Goldberg D. E.: Genetic and Evolutionary Algorithms Come of Age, *Communications os ACM 37(3)*, pp. 113-119, 1994..

[Gordon93] Gordon S. V, Whitley D.: Serial and Parallel Genetic Algorithms as Function Optimizers, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 177-183, Morgan Kaufmann, San Mateo, CA, 1993.

[Gorges89] Gorges-Schleuter M.: ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 422-427, Morgan Kaufmann, San Mateo, CA, 1989.

[Gorges91] Gorges-Schleuter M.: Explicit parallelism of genetic algorithms through population structures, *Parallel Problem Solving from Nature - PPSN ,* Schwefel H.P. and Manner R. (eds.), pp. 150-159, Springer-Verlag, Berlin, 1991.

[Grefen84] Grefenstette J.J.: Genesis: A system for Using Genetic Search Procedures, *Proceedings of the Conference on Intelligent Systems and Machines,* pp. 161-165, 1984.

[Grefen86] Grefenstette J. J.: Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man and Cybernetics, Vol 16*, pp. 122-128, 1986.

[Grefen89] Grefenstette J. J, Baker J. E.: How Genetic Algorithms Work: A Critical Look at Implicit Parallelism, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 20-27, Morgan Kaufmann, San Mateo, CA, 1989.

[Grefen92] Grefenstette J. J.: Deception Considered Harmfull, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 75-93, Morgan Kaufmann, San Mateo, CA, 1992.

[Grefen95] Grefenstette J.: Virtual Genetic Algorithms: First Results, *Internal report,* NAVY Center for Applied Research in Artificial Inteligence (NCARAI), 1995.

[Grefen96] Grefenstette J.: Competition-based Learning, *Internal report,* NAVY Center for Applied Research in Artificial Inteligence (NCARAI), 1996.

[Guigna88] Guignard M.: A Lagrangean dual ascent algorithm for simple plant location problems, *European Journal of Operational* Research, Vol. 35, pp. 193-200, 1988.

[Hamil02] Hamilton-Wright A, Stacey D.: Fault-Tolerant Network Computation of Individuals in Genetic Algorithms, *Proceedings of the 2002 IEEE World on Computational Intelligence – WCCI 2002*, *Congress on Evolutionary Computation – CEC 2002*, pp. 1721-1726, Honolulu, Hawaii, 12-17. 05. 2002.

[Hancoc94] Hancock P. J.: B. An empirical comparasion of selection methods in evolutionary algorithms, *Evolutionary Computing*, pp. 80-94, AISB Workshop Leeds, april 1994.

[Hansen97] Hansen P, Mladenović M.: Variable Neighborhood Search for the *p*-median. *Location Science 5*, pp. 207-226, 1997.

[Harik97] Harik G. R, Lobo F. G, Goldberg D. E.: The Compact Genetic Algorithm, *IlliEAL Report No 97006*, University of Illinois, august 1997.

[Hecker96] Heckerdorn R. B, Whitley D, Rana S.: Nonlinearity, Walsh Coefficient, Hyperplane Ranking and the Simple Genetic Algorithm, *Internal report*, Department of Computer Science, Colorado State University, february 1996.

[Heitko93] Heitkoetter J, Beasley D.: The Hitch-Hiker's Guide to Evolutionary Computation, *FAQ* in *comp.ai.genetic*, 1993.

[Hejlsb03] Hejlsberg A.: *Exception Management in .NET*, Microsoft Patterns&Practices, Microsoft press, 2003.

[Herber96] Herbert D.: A Markov Chain Analysis of Genetic Algorithms with a State Dependent Fitness Function, *Internal report*, Department of Operations Research and Systems Theory, Vienna University of Technology, 1996.

[Hinter95] Hinterding R, Gielewski H, Peachey T. C.: The Nature of Mutation in Genetic Algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 65-72, Morgan Kaufmann, San Mateo, CA, 1995.

[Hollan75] Holland J. H.: *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[Hollan92] Holland J. H.: *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, Massachusetts, 1992.

[Holmbe91] 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, Linkoping Institute of Technology, Sweden, 1991.

[Holmbe95] 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.

[Holmbe97] 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*

[Hollmbe97a] 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*

[Horn93] Horn J.: Finite Markov Chain Analysis of Genetic Algorithms with Niching, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 110-117, Morgan Kaufmann, San Mateo, CA, 1993.

[Hou94] Hou E. S. H, Ansari N.: A Genetic Algorithm for Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems, vol.5 no.2*, pp. 113-120, february 1994.

[Hughes89] Hughes M.: Genetic Algorithm Workbench Documentation, *Cambridge Consultants*, Cambridge, UK, 1989.

[Hutter02] Hutter M.: Fitness Uniform Selection to Preserve Genetic Diversity, *Proceedings of the 2002 IEEE World on Computational Intelligence – WCCI 2002, Congress on Evolutionary Computation – CEC 2002*, pp. 783-788, Honolulu, Hawaii, 12-17. 05. 2002.

[Ivkov89] Ivković Z.: *Teorija verovatnoća sa matematičkom statistikom*, Naučna knjiga, Beograd, 1989.

[Janiko91] Janikow C. Z, Michalewicz Z.: An Experimental Comparasion of Binary and Floating Point Representations in Genetic Algorims, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 37-44, Morgan Kaufmann, San Mateo, CA, 1991.

[Janson93] Janson D, Frenzel J.: Training Product Unit Neural Networks with Genetic Algorithms, *IEEE Expert*, october 1993.

[Jezier02] Jezierski E.: *Application Architecture for .NET - Designing Application and Services*, Microsoft Patterns&Practices, Microsoft Press, 2002.

[Jones95] Jones T, Forrest S.: Fitness Distance Corelation as a Measure of Problem Difficullty for Genetic Algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 184-192, Morgan Kaufmann, San Mateo, CA, 1995.

[Joshi02] Joshi B, Dickinson P.: *Proffesional ADO .NET*, Wrox Press, Birmingham, 2002.

[Juels94] Juels A, Wattenberg M.: Stohastic hillclimbing as a Baseline Method for Evaluating Genetic Algorithms, *Internal report*, Groupe de BioInformatique, Ecole Normale Superieure, 1994.

[Kalyan92] Kalyanmoy D, Goldberg D. E.: Analyzing Deception in Trap Functions, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 93-108, Morgan Kaufmann, San Mateo, CA, 1992.

[Kapsal94] Kapsalis A, Smith G.D, Rayward-Smith V.J.: A Unified Paradigm for Parallel Genetic Algorithms, *Evolutionary Computing*, pp. 131-149, AISB Workshop Leeds, april 1994.

[Karr99] Karr C.L, Freeman L.M.*: Industrial Applications of Genetic Algorithms*, CRC Press, Boca Raton, Florida, 1999.

[Kelly92] Kelly J. D. Jr, Davis L.: A Hybrid Genetic Algorithm for Classification, *Learning and Knowlege Acquisition*, pp. 645-650, 1992.

[Kahng95] Kahng A. B, Moon B. R.: Toward More Powerfull Recombinations, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 96-103, Morgan Kaufmann, San Mateo, CA, 1995.

[Keogh03] Keogh D.: *Logging Application Block*, Microsoft Patterns&Practices, Microsoft press, 2003.

[Khuri93] Khuri S.: Walsh and Haar Funcions in Genetic Algorithms, *Internal report*, San Jose State University, 1993.

[Khuri95] Khuri S, Heitkoetter J, Shutz M.: Evolutionary Heuristics for the Bin Packing Problem, *Proceedings of International Conferatio on Artificial NN and EAs - ICANNEA*, 1995.

[Klince91] Klincewicz J.G: Heuristics for the p-hub location problem, *European Journal of Operational Research 53* pp. 25-37, 1991.

[Klince96] Klincewicz J. G.: A Dual Algorithm for the Uncapacitated Hub Location Problem.*Location Science*, Vol. 4, No. 3, pp. 173-184, 1996.

[Kojima02] Kojima K, Matsuo H, Ishigame M.: Reduction of Communication Quantity for Network Based Parallel GA, *Proceedings of the 2002 IEEE World on Computational Intelligence – WCCI 2002, Congress on Evolutionary Computation – CEC 2002*, pp. 1715-1720, Honolulu, Hawaii, 12-17. 05. 2002.

[Koncha98] Konchady M.: Parallel Computing Using LINUX, *LINUX Journal*, pp. 78-81, 1998.

[Koza91] Koza J. R.: Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 37-44, Morgan Kaufmann, San Mateo, CA, 1991.

[Koza93] Koza J. R.: Hierarchial Genetic Algoritms Operating on Populations of Computer Programs, *Machine Learning*, pp. 768-774, 1993.

[Koza95] Koza J. R, Andre D.: Parallel Genetic Programming on Network of Transputers, *Technical report STAN-CS-TR-95-1542*, Stanford University, 1995.

[Körk89] Körkel M.: On the exact solution of large-scale simple plant location problems, *European Journal of Operational* Research, Vol. 39, pp. 157-173, 1989.

[Kratic96] Kratica J, Filipović V, Šešum V, Tošić D.: Solving of the uncapacitated warehouse location problem using a simple genetic algorithm, *Proceedings of the XIV International Conference on Material handling and warehousing*, pp. 3.33-3.37, Belgrade, 1996.

[Kratic97] Kratica J, Radojević S, Šešum V.: Jedna metoda poboljšanja vremena izvršavanja prostog genetskog algoritma, *Zbornik radova sa 23. JUPITER konferencije*, st. 4.55-4.59, Beograd, 1997.

[Kratic97a] Kratica J.: Napredne tehnike evolutivnih algoritama i njihova implementacija, *Prolećna škola o programskim jezicima '97 - Tekstovi predavanja*, st. 123-130, Novi Sad, 1997.

[Kratic98] Kratica J, Ljubić I, Šešum V, Filipović V.: Neke metode za rešavanje problema trgovačkog putnika pomoću evolutivnih algoritama, *Zbornik radova sa drugog međunarodnog simpozijuma iz industrijskog inženjerstva SIE '98*, st. 281-284, Mašinski fakultet, Beograd, 1998.

[Kratic98a] Kratica J, Filipović V, Tošić D.: Solving the Uncapacitated Warehouse Location Problem by SGA with Add-Heuristic, *Proceedings of the XV ECPD International conference on material handling and warehousing*, pp. 2.28-2.32, Faculty of Mechanical Engineering, Belgrade, 1998.

[Kratic99] Kratica J.: Improving Performances of the Genetic Algorithm by Caching, *Computers and Artificial Intelligence*, Vol. 18, No. 3, pp. 271-283, 1999.

[Kratic99a] Kratica J.: Improvement of Simple Genetic Algorithm for Solving the Uncapacitated Warehouse Location Problem, *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 390-402, Springer-Verlag London Ltd., 1999.

[Kratic00] Kratica J.: Paralelizacija evolutivnih algoritama za rešavanje nekih NP-kompletnih problema, *Doktorska disertacija*, Matematički fakultet, Beograd, 2000.

[Kratic00a] Kratica J, Tošić D, Filipović V, Ljubić I.: Genetic Algorithm for Designing a Spread-Spectrum Radar Polyphase Code, *Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications - WSC5*, pp. 191-197, september 2000.

[Kratic01] Kratica J, Filipović V, Tošić D, Ljubić I.: Solving the Simple Plant Location Problem by Genetic Algorithm, *RAIRO Operations Research, Vol. 35, No. 1*, pp. 127-142, 2001.

[Kratic01a] Kratica J, Tošić D.: Introduction to Genetic Algorithms and Some Applications, *Proceedings of a Workshop on Computational Intelligence - Theory and Applications*, pp. 57-68, Niš, Yugoslavia, february 2001.

[Kratic01b] Kratica J, Tošić D, Filipović V, Ljubić I.: Comparing Performances of Several Algorithms for Solving Simple Plant Location problem, *Proceedings of the 10th Congress of Yugoslav Mathematicians*, pp. 337-341, Belgrade, 2001.

[Kratic01c] Kratica J, Tošić D, Filipović V, Ljubić I.: A Genetic Algorithm for the Uncapacitated Network Design Problem, *6th Online World Conference on Soft Computing - WSC6*, september 2001.

[Kratic02] Kratica J, Tošić D, Filipović V, Ljubić I.: A Genetic Algorithm for the Uncapacitated Network Design Problem, *Soft Computing in Industry - Recent Applications*, Springer Verlag, pp. 329-338., 2002.

[Kratic03] Kratica J, Ljubić I, Tošić D.:A genetic algorithm for the index selection problem, *Springer Lecture Notes in Computer Science*, Vol. *2611*, pp. 281-291, 2003.

[Kratic05] Kratica J, Stanimirović Z, Tošić D, Filipović V.: Genetic Algorithm for Solving Uncapacitated Multiple Allocation Hub Location Problem, *Computing and Informatics – CAI*, Vol.24 No 4, 2005

[Kratic06] Kratica J, Stanimirović Z.: Solving the Uncapacitated Multiple Allocation p-hub Center Problem by Genetic Algorithm, *Asia-Pacific Journal of Operational Research – APJOR*, u štampi

[Kratic06a] Kratica J, Stanimirović Z, Tošić D, Filipović V.: Two Genetic Algorithms for Solving the Uncapacitated Single Allocation p-Hub Median Problem, *European Journal of Operational Research – EJOR*, na recenziji

[Labbe04] Labbe M, Yaman H.: Projecting the Flow Variables for Hub Location Problems, *Internal Report*, Universite Libre de Bruxelles, 2004.

[Legg04] Legg S, Hutter M, Kumar A.: Tournament versus Fitness Uniform Selection, *Technical Report,* IDSIA-04-04, 20 March 2004.

[Levine93] Levine D.: *A Parallel Genetic Algorithm for the Set Partitioning Problem*, PhD thesis, Argonne National Laboratory, ANL-94/23, Illinois Institute of Technology, 1993.

[Levine93a] Levine D.: A genetic algorithm for the set partitioning problem, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93,* pp. 65-69 , Morgan Kaufmann, San Mateo, California, 1993.

[Levine95] Levine D.: *PGAPack Parallel Genetic Algorithm Library*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ilinoiss, 1995.

[Levine95a] 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, Ilinoiss, pp. 77, 1995.

[Lewis92] Lewis T, El-Rewini H.: *Introduction to Parallel Computing*, Prentice-Hall International, Englewood Cliffs, NJ, 1992.

[Li02] Li X, Kirley M.: The Effects of Varying Population Density in a Fine-grained Parallel Genetic Algorithm, *Proceedings of the 2002 IEEE World on Computational Intelligence – WCCI 2002, Congress on Evolutionary Computation – CEC 2002*, pp. 1709-1714, Honolulu, Hawaii, 12-17. 05. 2002.

[Lippma02] Lippman S. B.: *C#Primer, A Practical Approach*, Addison - Wesley, Boston, 2002.

[Ljubić98] Ljubić I, Kratica J, Filipović V.: Primena evolutivnih algoritama u nalaženju minimalnog Steinerovog stabla, *Zbornik radova sa drugog međunarodnog simpozijuma iz industrijskog inženjerstva SIE '98*, st. 277-280, Mašinski fakultet, Beograd, 1998.

[Ljubić00] Ljubić I, Kratica J.: A Genetic Algorithm for the Biconnectivity Augmentation Problem, *Proceedings of the Conference on Evolutionary Computation - CEC 2000*, pp. 89-96, San Diego, CA, july 2000.

[Ljubić00a] Ljubić I, Raidl G.R, Kratica J.: A Hybrid EA for the Edge-Biconnectivity Augmentation Problem, *Springer Lecture Notes in Computer Science, Vol. 1917*, pp. 641-650, september 2000.

[Ljubić00b] Trmčić (Ljubić) I.: Primena genetskih algoritama na probleme povezanosti grafova, *Magistarski rad,* Matematički fakultet u Beogradu, 2000.

[Ljubić01] Ljubić I., Raidl G.R.: An Evolutionary Algorithm with Hill-Climbing for the Edge-Bicconectivity Augmentation Problem, *Springer Lecture Notes in Computer Science 2037*, pp. 20-29, 2001.

[Ljubić03] Ljubić I., Raidl G. R.: A memetic algorithm for minimum-cost vertex-biconnectivity augmentation of graphs, *Journal of Heuristics, Vol. 9*, No. 5, pp. 401-428, 2003.

[Ljubić04] Ljubic I.: Exact and Memetic Algorithms for Two Network Design Problems, *PhD thesis*, Faculty of Computer Science, Vienna University of Technology, November 2004.

[Ljubić06]. Ljubic I, Weiskircher R, Pferschy U, Klau G, Mutzel P, Fischetti M, An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem, *Mathematical Programming, Series B*, prihvaćeno za štampu, 2005.

[Luke00] Luke S, Hamahashi S, Kyoda K, Ueda H.: Biology: See It Again—For The First Time, *Internal report*, Sony Computer Science Laboratory, 2000.

[Magna86] Magnanti T.L, Mireault P, Wong R.T.: Tailoring Benders Decomposition for Uncapacitated Network Design, *Mathematical Programming Study*, Vol. 29, pp. 464-484, 1986.

[Mander89] Manderick B, Spiessens P.: Fine-Grained Parallel Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 428-433, Morgan Kaufmann, San Mateo, CA, 1989.

[Manela93] Manela M, Thornhill N, Campbell J. A.: Fitting Spline Functions to Noisy Data Using a Genetic Algorithm, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 549-556, Morgan Kaufmann, San Mateo, CA, 1993.

[Mason93] Mason A.: Crossover Non Linearity Ratios and the Genetic Algorithm: Escaping the Blinkers of Schema Processing and Itrinsic Parallelism, *Report No 535b*, School of Engineering, University of Auckland, New Zealand

[Mayer02] Mayer G, Wagner B.: HubLocator: An Exact Solution Method for the Multiple Allocation Hub Location Problem, *Computers and Operations Research*, Vol. 29, pp. 715-739, 2002.

[McLaug00] McLaughlin B.: *Java and XML*, O'Reilly, Cambridge, 2000.

[Meier02] Meier J.D, Mackman A, Vasireddy S, Dunner M.: *Building secure ASP .NET Applications*, Microsoft Pattrens&Practices, Microsoft press, 2002.

[Merelo94] 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*

[Merelo94a] 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*

[MGA92] MicroGA*, Genetic Algorithm Digest*, Vol. 6, No. 35, *Emergent Behavior,* Palo Alto, CA, 1992.

[Michal95] Michalewicz Z.: Genetic Algorithms, Numerical Optimizations and Constraints, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 151-158, Morgan Kaufmann, San Mateo, CA, 1995.

[Michal96] Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, third edition, Springer, Berlin, 1995.

[Miller97] Miller B.: Noise, Sampling and Efficient Genetic Algorithm, *IlligaL Report No 97001*, Illinois Genetic Algorithms Laboratory, may 1997.

[Mitch99] Mitchell M.: *An Introduction to Genetic Algorithms*, A Bradford Book – The MIT Press, Massachusetts, 1999.

[Миренк89] Миренков Н. Н.: *Паралелное Програмирование дла многомодулних вичислителних систем*, Радио и Сваза, Москва, 1989.

[Mock89] Mock J.: *Processes, Channels, and Semaphores, Transputer Toolset*, Inmos Corporation, 1989.

[MPI94] MPI Forum, The Message Passing Interface, *Interantional Journal of Supercomputing Applications*, Vol. 8, No. 3-4, pp. 165-414, 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, pp. 239, 1995. http://www.netlib.org/mpi/mpi-1.1-report.ps

[Muneto93] Munetomo M, Takai Y, Sato Y.: An Efficient Migration Scheme for Subpopulation-Based Asynchronously Parallel Genetic Algorithms, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 649, Morgan Kaufmann, San Mateo, CA, 1993.

[Mühlen89] Mühlenbein H.: Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, *Proceedings of the Third International*

Conference on Genetic Algorithms - ICGA '89, pp. 416-421, Morgan Kaufmann, San Mateo, CA, 1989.

[Mühlen92] Mühlenbein H.: Parallel Genetic Algorithm in Combinatorial Optimization, *Computer Science and Operations Research*, Balci O., Sharda R., Zenios S. (eds), pp. 441-456, Pergamon Press, 1992.

[Mühlen97] Mühlenbein H.: Genetic algorithms, *Local Search in Combinatorial Optimization,* eds. Aarts E.H.L., Lenstra J.K., pp. 137-172, John Wiley & Sons Ltd., 1997.

[Neri95] Neri F, Saitta L:: Analysis of Genetic Algorithms Evolution under Pure Selection, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 32-41, Morgan Kaufmann, San Mateo, CA, 1995.

[Neri95a] Neri F, Giordana A.: A Parallel Genetic Algorithm for Concept Learning, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 436-443, Morgan Kaufmann, San Mateo, CA, 1995.

[Ng95] Ng W. K, Ravishankar C. V.: A Preliminary Study of Genetic Data Compresson, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 566-573, Morgan Kaufmann, San Mateo, CA, 1995.

[Nichol96] Nichols B, Buttlar D, Farrell J.P.: *Pthreads Programming*, O'Reilly & Associates, Sebastopol, California, 1996.

[Nickel01] Nickel S.: Discrete Ordered Weber problems, Operations Research Proceedings, pp. 71–76, Springer Verlag, 2001.

[Nillso98] Nillson N. J.: *Artificial Inteligence: A New Synthesis*, Morgan Kaufman, San Francisko, CA, 1998.

[Ognjan01] Ognjanović Z., Kratica J., Milovanović M.: A Genetic Algorithm for Satisfiability Problem in a Probabilistic Logic: A First Report, *Springer Lecture Notes in Artificial Intelligence, Vol. 2143*, pp. 805-816, 2001.

[Ognjan04] Ognjanović Z., Midić U., Kratica J.: A Genetic Algorithm for Probabilistic SAT Problem, *Springer Lecture Notes in Artificial Intelligence, Vol. 3070*, pp. 462-467, 2004.

[Olive03] Oliver R I.: *Designing Enterprise Applications with Microsoft Visual Basic .NET*, Microsoft press, Redmond, Washington, 2003.

[Park95] Park K.: A Comparative Study of Genetic Search, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 512-519, Morgan Kaufmann, San Mateo, CA, 1995.

[Perez04] Pérez Pérez M, Almeida Rodríguez F, Moreno Vega J.M.: On the use of the path relinking for the p-hub median problem, *Lecture Notes in Computer Science 3004*, pp. 155-164, 2004.

[Peltz03] Peltz C.: Web Services Orchestration and Choreography, *Computer Volume 36 Number 10*, pp. 46-52, IEEE Computer Society, October 2003.

[Petey89] Petey C. C, Leuze M.R.: A theorthical Investigation of a Parallel Genetic Algorithm, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 398-405, Morgan Kaufmann, San Mateo, CA, 1989.

[Puch04] Puchinger J, Raidl G.R, Koller G.: Solving a Real-World Glass Cutting Problem, *Springer Lecture Notes in Computer Science 3004*, pp. 162-173, 2004.

[Radcli92] Radcliffe N. J.: Genetic Set Recombination, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 203-221, Morgan Kaufmann, San Mateo, CA, 1992.

[Raidl02] Raidl G.R, Ljubić I.: Evolutionary local search for the edge-biconnectivity augmentation problem, *Information Processing Letters Journal*, Vol. 82, No. 1, pp. 39-45, 2002.

[Reevs94] Reevs C. R. Genetic Algorithms and Neighbourhood Search, *Evolutionary Computing*, pp. 115-130, AISB Workshop Leeds, april 1994.

[Reevs95] Reevs C. R.: *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, London, 1995.

[Reevs95a] Reevs C. R, Wright C. C.: Epistasis in Genetic Algorithms: An Experimental Design Perspective, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 217-224, Morgan Kaufmann, San Mateo, CA, 1995.

[Resen03] Resende M.G.C, Werneck R.F.: On the Implementation of a Swap-Based Local Search Procedure for the p-median Problem, *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments - ALENEX'2003*, pp. 119-127, SIAM, Philadelphia, 2003.

[Reynol96] Reynolds D Gomatam J.: Stohastic modelling of Genetic Algorithms, *Artificial Inteligence 82*, pp. 303-330, 1996.

[ReyHae02] Reynolds-Haertle R. A.: *OOP with Microsoft VB .NET and Microsoft C# .NET*, Microsoft press, Redmond, Washington, 2002.

[Ribar86] Ribarić S., *Arhitektura računala pete generacije,* Tehnička knjiga, Zagreb, 1986.

[Ribeir94] Ribeiro-Filho J.L., Treleavean P.C., Alippi C.: Genetic-Algorithm Programming Environments, *IEEE Computer,* pp. 28-43, June 1994.

[Ritche02] Ritcher J.: *Applied Microsoft .NET Programming Framework*, Microsoft press, Redmond, Washington, 2002.

[Robbin03] Robbins J.: *Debugging Applications for Microsoft .NET and Microsoft Windows*, Microsoft Press, Redmond, Washington, 2003.

[Robins02] Robinson S.: *C#2nd Edition*, Wrox Press, Birmingham, 2003.

[Rudolp95] Rudolph G.: Convergence Analysis of Canonical Genetic Algorithms, *Internal report*, Department of Computer Science, University of Dortmund, 1995.

[Rumba98] Rumbaugh, J, Jacobson, I, Booch, G.: *The Unified Modeling Language Reference Manual,* Addison-Wesley, Boston, MA, 1998.

[Schaff89] Schaffer D. J, Caruana R. A, Eshelman L. J, Rajarshi D.: A Study Of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, *Proceedings of*

the Third International Conference on Genetic Algorithms - ICGA '89, pp. 51-60, Morgan Kaufmann, San Mateo, CA, 1989.

[Schne01] Schneider G, Winters J. P.: *Applying Use Cases*, Addison-Wesley, Upper Saddle River, NJ 07458, 2001.

[Schrau92] Schraudolph N. N, Belew R. K.: Dynamic Parameter Encoding for Genetic Algorithms, *UCSD Tehnical Report CS90-175*, University of CA, San Diego, july 1992.

[Schwef95] Schwefel H.-P.: *Evolution and Optimum Seeking*, John Wiley & Sons, New York, 1995.

[Schweh96] Schwehm M.: Parallel Population Models for Genetic Algorithms, *Internal report*, University of Erlargen-Nurnberg, 1996.

[Shallo03] Shalloway A, Trott J.R.: Design Patterns Explained – a New Perspective on Object Oriented Design, NetObjectives, 2003.

[Shonkw93] Shonkwiler R.: Parallel Genetic Algorithm, *Proceedings of the Fifth International Conference on Genetic Algorithms - ICGA '93*, pp. 199-205, Morgan Kaufmann, San Mateo, CA, 1993.

[Simao89] 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.

[Skjell92] Skjellum A, Smith S, Still C, Leung A, Morari M.: The Zipcode message passing system, *Technical Report*, Lawrence Livermore National Laboratory, 1992. http://www.netlib.org/mpi/zipcode.uue

[Skorin94] Skorin-Kapov D, Skorin-Kapov J.: On tabu search for the location of interacting hub facilities, *European Journal of Operational Research 73*, pp. 502-509, 1994.

[Skorin96] Skorin-Kapov D, Skorin-Kapov J, O'Kelly M.: Tight Linear Programming Relaxations of Uncapacitated p-hub Median Problems, *European Journal of Operational Research 94*, pp. 582-593, 1996.

[Sohn98] Sohn J, Park S.: Efficient Solution Procedure and Reduced Size Formulations for p-hubLocation Problems, *European Journal of Operational Research 108*, pp. 118-126, 1998.

[Spears91] Spears W. M.: Adapting Crossover in Evolutionary Algorithms, *Evolutionary Programming IV*, pp. 367-384, 1991.

[Spears95] Spears W. M.: A NN Algorithm for Boolean Satisfability Problems, *Internal report*, NAVY Center for Applied Research in Artificial Inteligence (NCARAI), 1995.

[Spears95a] Spears W. M.: Recombination parameters, *Internal report*, NAVY Center for Applied Research in Artificial Inteligence (NCARAI), 1995.

[Spears00] Spears W. M.: The Equilibrium and Transient Behavior of Mutation and Recombination, *Internal report*, NAVY Center for Applied Research in Artificial Inteligence (NCARAI), 2000.

[Spiess91] Spiessens. P, Manderick B.: A Massively Parallel Genetic Algorithm – Implementationa and First Analysis, *Proceedings of the Fourth International Conference on Genetic Algorithms - ICGA '91*, pp. 271-278, Morgan Kaufmann, San Mateo, CA, 1991.

[Sriniv94] Srinivas M, Patnaik L. M.: Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms, *IEEE Transactions on Systems, Man, And Cybernetics, vol.25, no. 4*, pp. 656-667, april 1994.

[Stan04] Stanimirović Z.: Rešavanje nekih diskretnih lokacijskih problema primenom genetskih algoritama, *Magistarski rad*, Matematički fakultet u Beogradu, 2004.

[Stan06] Stanimirović Z, Kratica J, Dugošija Đ.: Genetic Algorithms for Solving the Discrete Ordered Median Problem, *European Journal of Operational Research – EJOR*, na recenziji

[Stan06a] Stanimirović Z.: An Efficient Genetic Algorithm for the Uncapacitated Multiple Allocation p-hub Median Problem, *Control and Cybernetics*, na recenziji

[Stanle95] Stanley T. J, Mudge T.: A Parallel Genetic Algorithm for Multiobjective Mikroprocessor Design, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 597-604, Morgan Kaufmann, San Mateo, CA, 1995.

[Starkw91] Starkweather T, Whitley D, Mathias K.: Optimization Using Distributed Genetic Algorithms, *Parallel Problem Solving from Nature - PPSN ,* Schwefel H.P. and Manner R. (eds.), pp. 176-185, Springer-Verlag, Berlin, 1991.

[Sumida90] Sumida B.H, Houston A.I, McNamara J.M, Hamilton W.D.: Genetic Algorithms and Evolution, *Journal of Theoretical Biology 147*, pp. 59-84, 1990.

[Suzuki95] Suzuki J.: A Markov Chain Analysis on Simple Genetic Algorithms, *IEEE Transactions on Systems, Man, And Cybernetics, vol.25, no. 4*, pp. 655-659, april 1995.

[Suzuki97] Suzuki J.: Further Results on the Markov Chain Model of GA and their Application to SA-like Strategy, *Internal report*, Information Systems Labaratory, Stanford University, 1997.

[Syswer89] Syswerda G.: Uniform Crossover in Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 2-9, Morgan Kaufmann, San Mateo, CA, 1989.

[Tanese87] Tanese R.: Parallel Genetic Algorithms for a Hypercube, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, pp. 177-183, 1987.

[Tanese89] Tanese R.: Distributed Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 434-439, Morgan Kaufmann, San Mateo, CA, 1989.

[Tcha88] 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.

[Teale03] Teale P, Etz C, Kiel M, Zeitz C.: *Data Patterns*, Microsoft Patterns&Practices, Microsoft press 2003.

[Teilh04] Teilhet S, Hilyard J.: *C# Cookbook*, O'Reilly & Associates, Inc., Sebastopol, CA, 2004

[Thier94] Thierens D, Goldberg D.: Convergence Models of Genetic Algorithm Selection Schemes, *Parallel Problem Solving from Nature - PPSN III*, pp. 119-129, *Lecture Notes in Computer Science 866*, Springer-Verlag, Berlin, 1994.

[Thomp96] Thompson R. K, Wright A. H.: Additively Decomposable Fitness Functions, *Internal report*, Computer Science Department, University of Montana, 1996.

[Topcu05] Topcuoglu H, Court F, Ermis M, Yilmaz G.: Solving the uncapacitated hub location problem using genetic algorithms, *Computers & Operations Research,* Vol. 32, pp. 967–984, Elsevier, 2005.

[Tošić97] Tošić D.: Pregled razvoja i opis osnovnih karakteristika evolutivnih (genetskih) algoritama, *Prolećna škola o programskim jezicima '97 - Tekstovi predavanja*, st. 115-122, Novi Sad, 1997.

[Trowbr03] Trowbridge D, Mancini D, Quick D, Hohpe G, Newkirk J, Lavigne D.: *Enterprise Solution Patterns Using Microsoft .NET*, Microsoft Patterns&Practices, Microsoft press, 2003.

[Turner03] Turner.-M, Budgen D, Brereton P.: Turning Software into a Service, *Computer Volume 36 Number 10*, pp. 38-44, IEEE Computer Society, October 2003.

[VanRo83] Van Roy T. J.: Cross Decomposition for Mixed Integer Programming, *Mathematical Programming*, Vol. 25, pp. 46-63, 1983.

[Vinc02] Vincent J.: A Comparison of Reproductive Success and the Effect of Mating Restrictions in Coarse-Grained Parallel Genetic Algorithms, *Proceedings of the 2002 IEEE World on Computational Intelligence – WCCI 2002, Congress on Evolutionary Computation – CEC 2002*, pp. 1697-1702, Honolulu, Hawaii, 12-17. 05. 2002.

[Voigt91] Voigt H.M, Born J, Treptow J.: *The Evolution Machine Manual - V 2.1*, Institute for Informatics and Computing Techniques, Berlin, 1991.

[Vose91] Vose M. D.: Generalizing the notion of schema in genetic algorithms, *Artificial Intelligence 50*, pp. 385-396, 1991.

[Vose92] Vose M. D.: Modeling Simple Genetic Algorithms, *Foundation of Genetic Algorithms 2 - FOGA 2*, pp. 63-73, Morgan Kaufmann, San Mateo, CA, 1992.

[Vugdel96] Vugdelija M, Kratica J, Filipović V, Radojević S.: Mogućnosti Evolutivnih algoritama u mašinskom učenju, *Zbornik radova sa 22. JUPITER konferencije*, st. 4.55-4.59, Beograd, 1996.

[W3C] World Wide Web Consortium . *http://www.w3.org*

[W3CXML] World Wide Web Consortium .XML *http://www.w3.org/xml*

[Weisf00] Weisfeld M.: *The Obiect-Orientied Thought Process*, SAMS Publishing, Indiana, USA 2000.

[Whigha96] Whigham P. A.: A Generalised Schema Theorem for Fixed and Variable-Lenght Structures, *Internal report*, Department of Computer Science, University College, University of New South Wales, february 1996.

[Whitle89] Whitley D.: The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best, *Proceedings of the Third International Conference on Genetic Algorithms - ICGA '89*, pp. 116-123, Morgan Kaufmann, San Mateo, CA, 1989.

[Whitle90] Whitley D, Starkweather T.: Genitor-II: A Distributed Genetic Algorithm, *Journal of Experimental Theoretical Artificial Intelligence*, Vol. 2, pp. 189-214, 1990.

[Whitle95] Whitley D, Mathias K, Rana S, Dzubera J.: Building Better Test Functions, *Proceedings of the Sixth International Conference on Genetic Algorithms - ICGA '95*, pp. 239-247, Morgan Kaufmann, San Mateo, CA, 1995.

[Whitle96] Whitley D, Rana S, Dzubera J, Mathias K. E.: Evaluating evolutionary algorithms, *Artificial Intelligence 85*, pp. 245-276, 1996.

[Wilki98] Wilkinson B, Allen M.: *Parallel Programming: Techniques and Application Using Networked Workstations and Parallel Computers*, Prentice-Hall, Upper Saddle River, NJ, 1998.

[Wilson02] Wilson J.: *Microsoft .NET Server Solutions for the Enterprise*, Microsoft press, Redmond, Washington, 2002.

[Wolper95] Wolpert D.H, Macready W.G. No Free Lunch Theorems for Search, *Internal report*, Santa Fe Instiute, 1995

INDEKS