SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 503

Genetski algoritam s vjerojatnosnim modelom

Mislav Bobesić

Zagreb, veljača 2013.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Evolucijsko računanje područje je umjetne inteligencije koje pronalazi inspiraciju u biološkim mehanizmima poput evolucije te se koristi za rješavanje različitih kombinatoričkih problema. Najpoznatije grane evolucijskog računanja su evolucijski algoritmi, inteligencija roja, algoritmi umjetnog imunološkog sustava i klasifikatorski sustavi sa sposobnošću učenja. U ovom radu naglasak je na genetskim algoritmima s vjerojatnosnim modelom koji su podvrsta evolucijskih algoritama.

Genetski algoritam s vjerojatnosnim modelom (engl. *probabilistic model-building genetic algorithms, u daljnjem tekstu PMBGA*) razvijen je zbog nedostataka genetskih algoritama kod problema s raspršenim građevnim blokovima. Naime, dobra rješenja često su karakterizirana uzorcima koji se nazivaju građevnim blokovima za koja genetski algoritmi nemaju mehanizme očuvanja. U okviru diplomskog rada programski je ostvareno nekoliko vrsti PMBGA. Također, programski je ostvaren i jednostavni genetski algoritam kao referentni algoritam s čijim su rezultatima uspoređeni rezultati genetskih algoritama s vjerojatnosnim modelom. Algoritmi su ispitani na optimizacijskim funkcijama različitih karakteristika kako bi rezultati bili što potpuniji.

U sljedećem poglavlju ukratko su opisane najpoznatije grane evolucijskog računanja. Nakon toga slijedi opis PMBGA te kratki opis najpoznatijih algoritama. Poglavlje nakon toga detaljno su opisani programski ostvareni algoritmi: jednostavni genetski algoritam, postupno učenje zasnovano na populaciji, algoritam univarijantne marginalne distribucije, kompaktni genetski algoritam i algoritam bivarijantne marginalne distribucije. Nakon toga opisane su ispitne funkcije, provedeno je vrednovanje algoritma te su opisani dobiveni rezultati.

2. Evolucijsko računanje

2.1. Inteligencija roja

Inteligencija roja područje je evolucijskog računanja koje svoju inspiraciju vuče iz ponašanja jedinki koje u kolektivu pokazuju inteligenciju. Tipičan primjer inteligencije roja su mravi. Iako mrava samog po sebi ne smatramo inteligentnim bićem, ponašanje kolektiva mrava možemo okarakterizirati inteligentnim. Inteligenciju roja u prirodi možemo naći osim u već spomenutim kolonijama mrava i u jatu ptica, u stadu životinja, među bakterijama, u rojenju pčela i u jatu riba.

Iako postoje velike razlike između algoritama neke osobine se pojavljuju u svima. Algoritam sadrži veliki broj jedinki koje se ponašaju prema određenim pravilima na temelju interakcije s okolinom i drugim jedinkama. Zahvaljujući tim pravilima pojavljuje se inteligentno ponašanje sustava. Prisutna inteligencija naziva se izranjajuća inteligencija (engl. *Emerging intelligence*) te ju karakterizira inteligentno ponašanje sustava jedinki kojeg jedinke nisu svjesne.

Inteligencija roja sadrži veliki broj algoritama, a najpoznatiji algoritmi iz ove skupine su algoritam kolonije mrava (engl. *Ant Colony Optimization*) i algoritam roja čestica (engl. *Particle Swarm Optimization*). Osim njih, ovdje spadaju još i algoritam umjetne kolonije pčela (engl. *Artificial Bee Colony Algorithm*), stohastička difuzna pretraga (engl. *Stochastic Diffusion Search*), algoritam krijesnica (engl. *Firelfy Algorithm*) i još mnogi drugi.

Primjena inteligencije roja česta je u robotici pa tako primjerice američka vojska istražuje mogućnost primjene inteligencije roja u automatskom upravljanju vozilima. Također, inteligencija roja se koristi i u telekomunikacijama za rješavanje problema usmjeravanja (engl. *Routing*).

2.2. Algoritmi umjetnog imunološkog sustava

Algoritmi umjetnog imunološkog sustava (engl. *Artificial Immune System, AIS*) kao što im i ime govori inspiraciju pronalaze u imunološkom sustavu živih bića. Živa su bića neprestano izložena raznim virusima, bakterijama i općenito stranim tijelima koji mogu oštetiti organizam živog bića. Odgovor organizma jest imunološki sustav, složen sustav koji se raznim mehanizmima bori protiv stranih tijela. Posebno zanimljiv dio imunoloških sustava su sposobnost prilagodbe i sposobnost učenja koji mu omogućuju obranu od dosada nepoznatih stranih tijela.

Algoritme umjetnog imunološkog sustava možemo podijeliti u četiri podvrste koje se razlikuju po dijelovima imunološkog sustava na temelju kojeg je dobivena inspiracija za algoritam. Podvrste su:

- negativna selekcija,
- klonska selekcija,
- imunološke mreže i
- algoritam dendritskih stanica.

Primjena algoritma umjetnog imunološkog sustava je vrlo raznolika. Tako su algoritmi negativne selekcije često korišteni u klasifikaciji i raspoznavanju uzoraka, algoritmi klonske selekcije u problemima optimizacije i u raspoznavanju uzoraka. Imunološke mreže su korištene za rješavanje problema grupiranja, u vizualizaciji podataka i optimizaciji kontrolnih procesa. Potencijalna primjena algoritma dendritskih stanica je u računalnoj sigurnosti.

2.3. Klasifikatorski sustavi sa sposobnošću učenja

Klasifikatorski sustavi sa sposobnošću učenja (engl. Learning Classifier Systems, LCS) je tehnika strojnog učenja s uskim vezama s podržanim učenjem i genetskim algoritmima. LCS je prvi opisao John Holland kao sustav koji genetskim algoritmima uči pravila koja maksimiziraju nagradu dobivenu od okoliša. LCS se može podijeliti na tri podvrste: michiganski stil (engl. The Michigan Approach) koji je opisao Holland u radu u kojem je prvi put predstavljen LCS, pittsburghski stil (engl. The Pittsburgh Approach) te stil iterativnog učenja (engl. (The Iterative Rule Learning Approach).

2.4. Evolucijski algoritmi

Evolucijski algoritmi su stohastičke metode pretraživanja inspirirane biološkom evolucijom. Općenita ideja iza evolucijskih algoritama je: u populaciji jedinki, pritisak okoliša uzrokuje prirodnu selekciju što potiče preživljavanje najboljih jedinki i podizanje ukupne dobrote populacije.

Svaki evolucijski algoritam sastoji se od populacije jedinki, funkcije dobrote, operatora selekcije, mutacije i rekombinacije te se izvršava iterativno sve dok nije dobiveno zadovoljavajuće rješenje. Populacija jedinki predstavlja moguća rješenja. Selekcija se provodi na temelju funkcije dobrote te se njom odabiru jedinke koje preživljavaju iteraciju i nad kojima se provode rekombinacija i mutacija. Rekombinacija je miješanje dijelova jedinki odabranih selekcijom, dok je mutacija nasumična izmjena dijela jedinke. Mutacijom i rekombinacijom stvaraju se nove jedinke koje se natječu sa starima za mjesto u novom generaciji na temelju dobrote.

Evolucijski algoritmi dijele se u 4 podvrste:

- genetski algoritam,
- genetsko programiranje,
- evolucijsko programiranje i
- evolucijske strategije.

Genetski algoritam najčešće je korišteni evolucijski algoritam koji pomoću evolucijskih operatora selekcije, križanja i mutacije pretražuje prostor rješenja tražeći najbolje moguće rješenje. Genetsko programiranje je evolucijski algoritam koji optimizira populaciju računalnih programa uz pomoć evolucijskih operatora u cilju pronalaska programa koji najbolje izvršava računalni zadatak. Evolucijska strategija slična je genetskom algoritmu te ju odlikuje samoprilagođavanje. Evolucijsko programiranje slično je genetskom programiranju s time da se optimiraju samo numerički parametri dok je struktura programa koji predstavljaju jedinke populacije nepromijenjiva.

2.5. Genetski algoritam s vjerojatnosnim modelom

Genetski algoritmi s vjerojatnosnim modelom (engl. *probabilistic model-building genetic algorithms, u daljnjem tekstu PMBGA*), u literaturi prisutni i pod nazivom algoritmi s procjenom raspodjele (engl. *estimation of distribution algorithms*), metode su optimizacije koja je nastala zbog nedostataka genetskih algoritama, odnosno njihovih operatora. U optimizacijskim problemima dobra rješenja često sadrže iste uzorke,

odnosno građevne blokove. Međutim, ti građevni blokovi često su raštrkani te različitih veličina. Operatori evolucijskih algoritama nemaju svojstvo očuvanja građevnih blokova te zbog toga često imaju problema prilikom pronalaska rješnja problema sa raštrkanim građevnim blokovima.

Genetski algoritam s vjerojatnosnim modelom je razvijen s idejom da se građevni blokovi rješenja dobrih jedinka očuvaju iz generacije u generaciju. Algoritam stvara vjerojatnosni model koji pretpostavlja postojanje uzoraka u rješenju. Na temelju vjerojatnosnog modela stvaraju se nove jedinke. Algoritmi iz ove skupine detaljnije su opisani u sljedećem poglavlju.

3. Genetski algoritam s vjerojatnosnim modelom

Jedinka rješenja genetskog algoritma sastoji se od građevnih blokova, uzoraka koje pronalazimo kao dio dobrih rješenja. Operatori konvencionalnog genetskog algoritma, mutacija i križanje, ponekad nisu dovoljno učinkoviti za očuvanje tih građevih blokova. Tako je u [16] pokazano da genetski algoritmi rade dobro samo za probleme u kojima su građevni blokovi u uskim nakupinama. U istom radu pokazano je i da genetski algoritam radi vrlo loše za probleme s raštrkanim građevnim blokovima. Genetski algoritmi s vjerojatnosnim modelom (engl. *probabilistic model-building genetic algorithms, u daljnjem tekstu PMBGA*), u literaturi prisutni i pod nazivom algoritmi s procjenom raspodjele (engl. *estimation of distribution algorithms*), predloženi su kao rješenje tih nedostataka konvencionalnog evolucijskog algoritma. Osnovna ideja je spriječiti uništavanje tih građevnih blokova dajući im veliku vjerojatnost stvaranja. PMBGA to radi tako da prilikom stvaranja novih generacija eksplicitno daje vjerojatnosnu razdiobu preživljavanja građevnih blokova.

PMBGA, koji spada u grupu evolucijskih algoritama, stohastički je i optimizacijski algoritam zasnovan na populaciji rješenja koji usmjerava potragu za dobrim rješenjima na temelju vjerojatnosnog modela koji je stvoren iz određenog broja dobrih jedinki.

Uspoređujući konvencionalne evolucijske algoritme i PMBGA glavna sličnost je inicijalno nasumična populacija rješenja koja se iterativno poboljšava. Dok konvencionalni algoritmi koriste operatore poput mutacije i križanja za implicitno određivanje razdiobe varijabli, PMBGA eksplicitno koristi vjerojatnosnu razdiobu. Također, PMBGA, kao i konvencionalni algoritmi koristi selekciju za odabir najboljih jedinki, međutim operatori križanja i mutacije se općenito ne koriste.

Općeniti izvorni tekst programa PMBGA algoritama dan je u algoritmu prikazanom na slici 3.1. Kao i svi populacijski algoritmi, PMBGA započinje stvaranjem početne populacije veličine N. Uobičajeni način stvaranja početne populacije je nasumično generiranje jedinki. Postoje alternativni načini stvaranja koji u generiranje

Stvori početnu populaciju veličine N
ponavljaj
    Odaberi M najboljih rješenja
    Izračunaj razdiobu zajedničke vjerojatnosti na temelju obećavajućih rješenja
    Stvori novu generaciju na temelju razdiobe i zamijeni roditelje
dok Kriterij zaustavljanja nije zadovoljen

Slika 3.1: Osnovni algoritam PMBGA

populacije uključuju znanje o problemu, no ti načini stvaranja početne populacije je rijetko korišten. Zatim slijede faze iteracije.

U svakoj iteraciji prvo se evaluira postojeća populacija na temelju funkcije evaluacije. Zatim se odabire M obećavajućih jedinki, jedinki na temelju kojih će se kasnije računati razdioba zajedničke vjerojatnosti, gdje je M neki broj za koji vrijedi $M << N$. Broj obećavajućih rješenja M se zadaje kao parametar algoritma (npr. algoritam univarijantne marginalne distribucije) ili je algoritmom određeno da se odabire samo najbolja jedinka, tj. $M = 1$ (npr. postupno učenje zasnovano na populaciji).

Nakon toga slijedi izračun razdiobe zajedničke vjerojatnosti i stvaranje nove generacije, dva koraka po kojima se razlikuju specifični algoritmi. Razdioba se računa na temelju prethodno odabranih obećavajućih rješenja. Ovisno o specifičnoj implementaciji algoritma, razdioba se ili računa u svakoj iteraciji (kao na primjer u algoritmu univarijantne marginalne distribucije) ili se razdioba pamti te ažurira iz iteracije u iteraciju (kao na primjer u postupnom učenju zasnovanom na populaciji).

Sljedeći korak je stvaranje nove generacije. Kod PMBGA algoritama uobičajeno je stvaranje cijele populacije u svakoj iteraciji. Nova generacija se stvara uzorkovanjem (engl. *sampling*) novih jedinki na temelju izračunatih vjerojatnosti. Izračunate vjerojatnosti nam govore kolika je vjerojatnost da neka varijabla rješenja poprima određenu vrijednost. Tako npr. u binarnom prikazu rješenja, vjerojatnost govori kolika je vjerojatnost da određeni bit poprima vrijednost 1. Algoritam za svaki bit postavi na 0 ili 1 s zadanom vjerojatnošću.

Kriteriji zaustavljanja PMBGA su isti kao i kod konvencionalnih algoritama. Najčešće korišteni su kriteriji određeni brojem iteracija, kvalitetom rješenja i nedostatkom poboljšanja rješenja u određenom broju zadnjih iteracija.

PMBGA se dijele u 3 podvrste koje su detaljnije opisane u nastavku.

– univarijantni algoritmi koji pretpostavljaju da ne postoji zavisnost između vari-

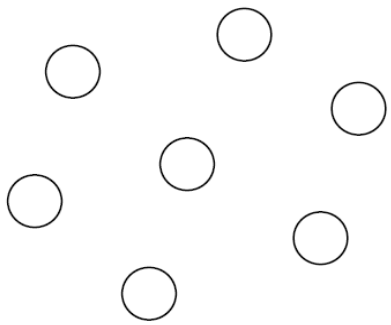
jabli rješenja,

- bivarijantni algoritmi koji pretpostavljaju da zavisnost postoji samo između dvije varijable,
- multivarijantni algoritmi koji pretpostavljaju da postoji zavisnost više od dvije varijable.

U nastavku poglavlja algoritmi su podijeljeni u podvrste te opisani. Programski ostvareni algoritmi su također detaljnije opisani u poglavlju 4 te se u ovom poglavlju navode samo osnovna svojstva tih algoritama.

3.1. Univarijatni algoritmi

Univarijantni algoritmi podvrsta su PMBGA koja pretpostavlja građevne blokove veličine 1, tj. da ne postoji nikakva međusobna zavisnost između varijabli rješenja. Očekivano, od svih podvrsta PMBGA, univarijantni imaju najmanju vremensku i prostornu složenost. Algoritmi koji pripadaju u ovu skupinu su postupno učenje bazirano na populaciji (engl. *Population Based Incremental Learning, u nastavku PBIL*), Algoritam univarijantne marginalne distribucije (engl. *Univariate Marginal Distribution Algorithm, UMDA*) i kompaktni genetski algoritam (engl. *Compact Genetic Algorithm, cGa*). Pregled algoritama dan je u nastavku.



Slika 3.2: Grafički prikaz univarijantne zavisnosti varijabli rješenja preuzet iz [14]

3.1.1. Postupno učenje zasnovano na populaciji

Postupno učenje zasnovano na populaciji (engl. *Population Based Incremental Learning, PBIL*) je jedan od prvih algoritama iz skupine PMBGA. Algoritam je predložen

u [1]. PBIL je nastao iz ideje kombiniranja genetskih algoritama i natjecateljskog učenja, tehnike često korištene u neuronskim mrežama. PBIL koristi binarni prikaz te uvodi vjerojatnosni vektor koji govori kolika je vjerojatnost da je vrijednost varijable rješenja 1.

Algoritam je programski ostvaren u okviru ovog rada te je detaljno opisan u poglavlju 4. Primjer izvođenja algoritma također je prisutan u navedenom poglavlju.

3.1.2. Kompaktni genetski algoritam

Kompaktni genetski algoritam (engl. *Compact Genetic Algorithm, cGa*) predložen je kao vjerojatnosna simulacija jednostavnog genetskog algoritma u [7]. Algoritam je također sličan PBIL-u. cGA održava vjerojatnosni vektor kao i PBIL, međutim uzorkuje samo dvije jedinke u jednoj iteraciju. Nakon usporedbe, ažurira vjerojatnosni vektor na temelju vrijednosti koje se razlikuju u pobjedničkom i gubitničkom rješenju. cGa je nastao iz pokušaja simulacije konvencionalnog genetskog algoritma sa probabilističkim algoritmom.

Algoritam je programski ostvaren u okviru ovog rada te je detaljno opisan u poglavlju 4. Primjer izvođenja algoritma također je prisutan u navedenom poglavlju.

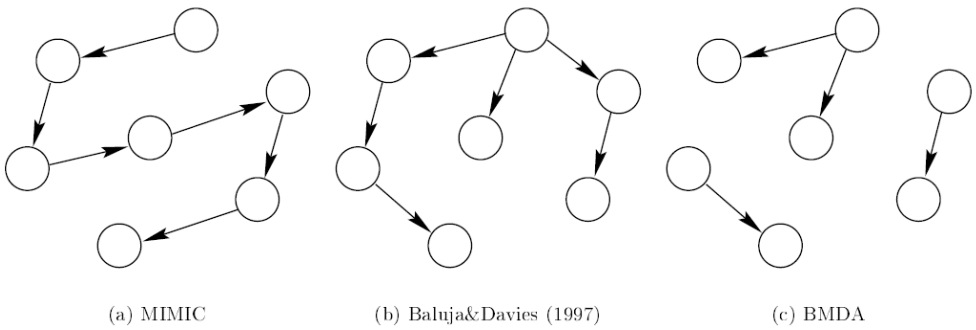
3.1.3. Algoritam univarijantne marginalne distribucije

Mühlenbein i Paass predložili su 1996 algoritam univarijantne marginalne distribucije (engl. *Univariate Marginal Distribution Algorithm, UMDA*). UMDA sličan je PBIL-u i jedina razlika je što nema memoriju. Dok PBIL ažurira svoj vjerojatnosni vektor, UMDA ga svaki put iznova računa. UMDA je jedan od prvih PMBGA koji je razvijen s idejom korištenja univarijantne marginalne distribucije za modeliranje raspodjele.

Algoritam je programski ostvaren u okviru ovog rada te je detaljno opisan u poglavlju 4. Primjer izvođenja algoritma također je prisutan u navedenom poglavlju.

3.2. Bivarijatni algoritmi

Bivarijantni algoritmi su podskupina PMBGA koja pretpostavlja građevne blokove veličine 2, tj. jedna varijabla rješenja ovisi najviše o jednoj drugoj varijabli rješenja . Bitno je napomenuti, moguće je i da više varijabli rješenja ovisi o jednoj, istoj varijabli rješenja. Moguće zavisnosti su prikazane na slici 3.3. Bivarijantni algoritmi bolje rade na algoritmima u kojima postoji zavisnost varijabli u parovima, ali s tim dolazi i veća složenost, kako vremenska, tako i prostorna. Algoritmi ove skupine su algoritam maksimizacije zajedničke informacije za grupiranje ulaznih podataka (engl. *Mutual Information Maximization for input clustering, MIMIC*), algoritam kombinirajućih optimizatora sa stablima zajedničke informacije (engl. *Combining Optimizers with Mutual Information Trees,COMIT*) i algoritam bivarijantne marginalne distribucije (engl. *Bivariate Marginal Distribution Algorithm, BMDA*).



Slika 3.3: Grafički prikaz bivarijantne zavisnosti varijabli rješenja preuzet iz [14]

3.2.1. Algoritam maksimizacije zajedničke informacije za grupiranje ulaznih podataka

U [3] je predložen algoritam maksimizacije zajedničke informacije za grupiranje ulaznih podataka (engl. *Mutual Information Maximization for input clustering, MIMIC*) koji koristi strukturu lanca za prikaz vjerojatnosne razdiobe. MIMIC se sastoji od dvije glavne komponente, nasumičnog optimizacijskog algoritma koji uzorkuje rješenja iz područja rješenja s velikom vjerojatnošću traženog rješenja i dijela koji vrši procjenu vjerojatnosne razdiobe.

Vjerojatnosna razdioba je lančane strukture i možemo ju zapisati kao:

$$P_{\Pi}(x) = p(x_{i_1}|x_{i_2})p(x_{i_2}|x_{i_3})...p(x_{i_{n-1}}|x_{i_n})$$

Uzorkuj populaciju sa uniformnom razdiobom.
Postavi granicu dobrote jediniki koje preživljavaju na medijan generirane populacije
ponavljaj
    Izbaci jedinke koje slabije od granice dobrote
    Generiraj vjerojatnosnu razdiobu
    Stvori novu generaciju na temelju razdiobe umjesto izbacenih jedinki
    Postroži granicu dobrote
dok Kriterij zaustavljanja nije zadovoljen

Slika 3.4: Algoritam MIMIC

gdje je $\Pi = i_1 i_2 ... i_n$ permutacija brojeva $1, 2...n$.

MIMIC je zapravo algoritam uzastopne aproksimacije. Algoritam prvo uzorkuje cijelu populaciju te pronalazi medijan rješenja. Na temelju rješenja čija je dobrota bolja od medijana, algoritam radi procjenu razdiobe. Zatim izbacuje rješenja koja su slabija od neke granice (obično medijana rješenja u prvoj iteraciji) te na temelju generirane procjene razdiobe stvara nove uzorke. Zatim se ponovo stvara procjena razdiobe, ali tako da se postrožuju kriteriji, tj. manji broj jedinki preživljava.

MIMIC je najjednostavniji bivarijantni algoritam te s idejom prikazivanja zavisnost varijabli lančanim grafom. Iz takvog prikaza strukture slijedi da svaka varijabla ovisi o najviše jednoj drugoj varijabli i da o svakoj varijabli ovisi najviše jedna druga varijabla.

Pseudokod algoritma MIMIC dan je na slici 3.4.

3.2.2. Algoritam kombinirajućih optimizatora sa stablima zajedničke informacije

Algoritam kombinirajućih optimizatora sa stablima zajedničke informacije (engl. *Combining Optimizers with Mutual Information Trees, COMIT*) je predložen u radu [2] te je bivarijantni PMBGA koji pretpostavlja stablastu vjerojatnosnu razdiobu. Algoritam je razvijen kao nadogradnja na MIMIC proširenjem mogućih razdioba. Stablastom strukturom koju koristi COMIT, mogu se prikazati i lančane strukture koje koristi MIMIC, ali obrnuto ne vrijedi. Iz takvog prikaza strukture slijedi da svaka varijabla ovisi o najviše jednoj drugoj varijabli dok o jednoj varijabli može ovisiti više drugih varijabli.

Algoritam stvara potpuni graf u kojem je svaka varijabla vrh, a svaki brid je zajednička vjerojatnost vrhova koje spaja. Nakon toga se pomoću algoritma maksimalnog

Pretpostavi da je zajednička vjerojatnost svake dvije varijable neka inicijalna vrijednost

ponavljaj

Stvori novo stablo koje predstavlja vjerojatnosnu razdiobu

Uzorkuj novu generaciju na temelju stabla

Smanji zajedničku vjerojatnost svih varijabli na temelju predefinirane konstante

Povećaj zajedničku vjerojatnost onih parova varijabli koje se nalaze u određenom broju rješenja

dok Kriterij zaustavljanja nije zadovoljen

Slika 3.5: Algoritam COMIT

razapinjujućeg stabla odredi optimalna stablasta struktura grafa. Dobiveni graf se koristi za modeliranje vjerojatnosti raspodjele.

Nadalje, raspodjela se koristi na sljedeći način:

$$P(x) = \prod_{x_i} P(x_i, x_{roditelj})$$

prilikom čega se gleda uvjeta vjerojatnost svake varijable ovisno o njezinom roditelju u prethodno dobivenom grafu.

Pseudokod algoritma COMIT dan je na slici 3.5.

3.2.3. Algoritam bivarijantne marginalne distribucije

BMDA predložen u [12] je nadogradnja na UMDA koja pretpostavlja građevne blokove veličine 2. BMDA je generalizacija prethodna dva algoritma s obzirom da je struktura vjerojatnosti šuma koja je zapravo skup stabala. Sa šumom kao strukturom se može prikazati i stablasta struktura, a samim time i lančana.

Algoritam koristi Pearsonovu chi-square statisktiku za mjerenje ovisnosti između varijabli. Varijable su 95% nezavisne ako je vrijednost Pearsonove chi-square statisktike manja od 3.84. Algoritam sve varijable koje su 95% nezavisne smatra nezavisnima, dok su ostale varijable zavisne.

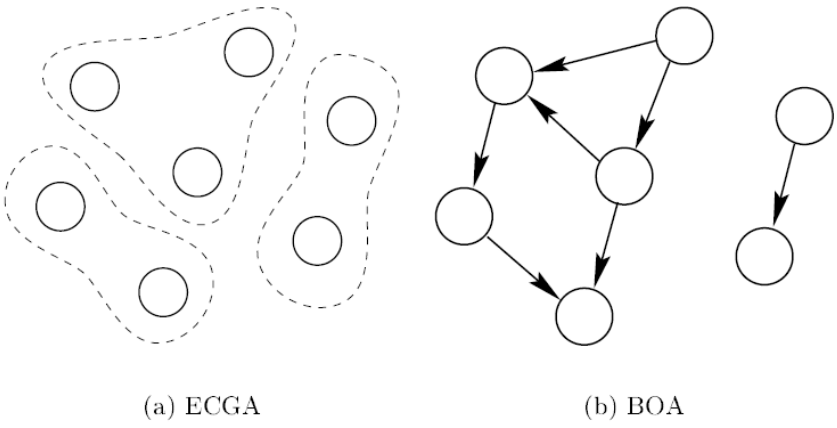
Za varijable $i \neq j$ nezavisnost se definira na sljedeći način:

$$X^2_{i,j} = \sum_{x_i, x_j} (N * p(x_i, x_j) - N * p(x_i) * p(x_j))$$

Algoritam je programski ostvaren u okviru ovog rada te je detaljno opisan u poglavlju 4. Primjer izvođenja algoritma također je prisutan u navedenom poglavlju.

3.3. Multivarijantni algoritmi

Multivarijantnim algoritmima smatraju se svi algoritmi koji pretpostavljaju građevne blokove veće od 2 varijable, odnosno zavisnost između 3 ili više varijabli. Zbog velikog porasta složenosti, u ovim algoritmima drastično raste memorijska i vremenska složenost. Čest pristup rješavanju tih problema je korištenje pohlepnih algoritama za stvaranje modela vjerojatnosti. Nedostatak tog pristupa je što ne garantira optimalnost dobivenog rješenja. Algoritmi iz ove skupine su prošireni kompaktni genetski algoritam (Extended Compact Genetic Algorithm, ECGA), algoritam faktorizirane raspodjele (Factorised Distribution Algorithm, FDA), Bayesov optimizacijski algoritam (Bayesian Optimization Algorithm, BOA), algoritam faktorizirane raspodjele sa sposobnošću učenja (Learning Factorised Distribution Algorithm, LFDA) i algoritam procjene Bayesove mreže (Estimation of Bayesian Network, EBNA).



Slika 3.6: Grafički prikaz multivarijantne zavisnosti varijabli rješenja preuzet iz [14]

3.3.1. Prošireni kompaktni genetski algoritam

Jedna od osobina efikasnog genetskog algoritma je i očuvanje građevnih blokova. Prepoznavanje tih blokova naziva se problem povezanosti. U [15] predloženo je rješenje koje je bazirano na tvrdnji da su rješenje problema povezanosti i odabir dobre vjerojatnostne razdiobe ekvivalentni problemi.

Uzorkuj populaciju sa uniformnom razdiobom
ponavljaj
    Odaberi jedinke turnirskom selekcijom
    Pohlepnim algoritmom modeliraj MPM
    Generiraj novu populaciju s dobivenim modelom
dok Kriterij zaustavljanja nije zadovoljen

Slika 3.7: Algoritam ECGA

Prošireni kompaktni genetski algoritam (engl. *Extended Compact Genetic Algorithm, ECGA*) je nadogradnja na cGA te koristi vjerojatnosni model koji se naziva model marginalnog produkta (engl. *Marginal Product Model, MPM*). Razlika između MPM i modela koje koriste prethodno opisani algoritmi je što MPM ne koristi zavisnost, nego samo univarijantnu marginalnu distribuciju i multivarijantnu marginalnu distribuciju. U praksi to znači da se ne prepostavlja zavisnost, već da se računa marginalna vjerojatnost skupa varijabli. Zbog velike složenosti, za izgradnju takvog modela koriste se pohlepni algoritmi.

Pseudokod algoritma ECGA dan je na slici 3.7.

Algoritam je vrlo jednostavan te je potrebno detaljnije opisati samo pohlepni algoritam. Algoritam započinje s pretpostavkom da su građevni blokovi veličine 1. Nakon toga gradijentnim spustom pokušavaju se spojiti građevne blokove kako bi se stvorili veći blokovi. Ukoliko se zbroj složenosti modela i složenosti prikaza rješenja smanjio spajanjem, blokovi se spajaju. Algoritam se izvršava dok više nema blokova koji bi se mogli spojiti.

3.3.2. Algoritam faktorizirane raspodjele

Proširenje prethodno navedenog UMDA koje pretpostavlja građevne blokove bilo koje veličine je Boltzmannov algoritam procjene distribucije (engl. *Boltzmann Estimated Distribution Algorithm, BEDA*) koji koristi Boltzmannovu selekciju za računanje Boltzmannove razdiobe. Detaljniji opis Boltzmannove selekcije i razdiobe može se pronaći u [9]. Iako je BEDA davao dobre rezultate, on nije iskoristiv u praksi zbog toga što je potrebno za izračun razdiobe zbrojiti eksponencijalno mnogo parametara. Algoritam faktorizirane raspodjele (engl. *Factorised Distribution Algorithm, FDA*) je nadogradnja BEDA-e s teoremom faktorizacije koji omogućuje efikasno računanje Boltzman-

Uzorkuj populaciju sa uniformnom razdiobom
ponavljaj
    Odaberi jedinke s Boltzmannovom selekcijom
    Procijeni uvjetne vjerojatnosti varijabli
    Generiraj novu populaciju na temelju uvjetnih vjerojatnosti
dok Kriterij zaustavljanja nije zadovoljen

Slika 3.8: Algoritam FDA

nove razdiobe.

Trenutno postoji više verzija FDA, međutim svima im je zajednička Boltzmannova razdioba. Za FDA nije nužno da koristi Boltzmannovu selekciju, moguće je i korištenje bilo koje druge metode selekcije. Međutim, konvergencija rješenja dokazana je samo u slučaju u kojem se koristi i Boltzmannova selekcija i Boltzmannova razdioba.

Pseudokod algoritma dan je na slici 3.8.

FDA je vrlo efikasan algoritam ako je struktura problema poznata unaprijed u obliku dekompozicije funkcije. Međutim, u stvarnim problemima struktura problema možda nije poznata ili nije dostupna. Kao rješenje tog nedostatka razvijen je LFDA.

3.3.3. Algoritam faktorizirane raspodjele sa sposobnošću učenja

Algoritam faktorizirane raspodjele sa sposobnošću učenja (Learning Factorised Distribution Algorithm, LFDA) je nadogradnja FDA koja ne treba poznavati strukturu problema unaprijed. Koristi se Bayesovska mreža za izračun modela vjerojatnosne razdiobe. Pristup je vrlo sličan pristupu u BOA, s jedinom razlikom u načinu ocjenjivanja kvalitete Bayesove mreže. BOA je detaljno opisan u sljedećem poglavlju.

3.3.4. Bayesov optimizacijski algoritam

Ideja zapisivanja modela zajedničke vjerojatnosti u Bayesovske mreže koristi se u Bayesovom optimizacijskom algoritmu (Bayesian Optimization Algorithm, BOA) predložena je u [13]. Bayesovska mreža često se koristi u PMBGA, osim u Bayesovskom optimizacijskom algoritmu, koristi se i u LFDA i u algoritmu procjene Bayesove mreže.

Vrlo bitan za rad algoritma je kriterij ocjene kvalitete Bayesove mreže. BOA uobičajeno koristi Bayes-Dirichlet metriku za ocjenu kvalitete mreže, ali u radovima su

Uzorkuj populaciju sa uniformnom razdiobom
ponavljaj
    Odaberi obećavajuće jedinke
    Konstruiraj Bayesovsku mrežu koristeći odabranu metriku
    Stvori K novih jedinki na temelju izgrađene mreže
    Zamijeni jedinike iz populacije sa jedinkama stvorenim u ovoj generaciji
dok Kriterij zaustavljanja nije zadovoljen

Slika 3.9: Algoritam BOA

korišteni i druge metrike poput minimalne dužine opisa. Kao i ostali multivarijantni algoritmi, za optimizaciju mreže BOA koristi pohlepni algoritam.

Pseudokod algoritma BOA dan je na slici 3.9.

4. Programski ostvareni algoritmi

U okviru ovog diplomskog rada programski su ostvareni sljedeći genetski algoritmi s vjerojatnosnim modelom:

- postupno učenje zasnovano na populaciji (PBIL),
- kompaktni genetski algoritam (cGa),
- algoritam univarijantne marginalne distribucije (UMDA) i
- algoritam bivarijantne mariginalne distribucije (BMDA).

Također je programski ostvaren jednostavni genetski algoritam s kojim uspoređujemo ostale programski ostvarene algoritme kako bi došli do zaključka o kvaliteti predloženih algoritama.

Algoritmi su testirani na optimizacijski teškim funkcijama s različitim karakteristikama. U ovom poglavlju, algoritmi su detaljnije opisani te su opisani implementacijski detalji. Također, opisu algoritma je pridodan primjer izvođenja dvije iteracije algoritma.

4.1. Jednostavni genetski algoritam

Genetski algoritam je najčešće korišteni evolucijski algoritam. Algoritam je stohastička metoda optimizacije te je prvi put opisan u [8]. Kao i ostalim evolucijskim algoritmima, inspiracija genetskog algoritma je darwinova teorija o postanku vrsta. Tako algoritam evoluira skup rješenja, populaciju, pomoću evolucijskih operatora te vrši selekcijski pritisak kojim otpadaju loše jedinke, a preživljavaju dobre.

Evolucijski operatori korišteni u genetskom algoritmu su selekcija, križanje i mutacija. Selekcija je proces odabira jedinki iz populacije rješenja nad kojima će se dalje vršiti ostali evolucijski operatori te čiji će genentski materijal prenijeti u sljedeću generaciju. Selekcija je stohastički postupak koji ne osigurava odabir najbolje jedinke, ali joj daje veću vjerojatnost da će biti odabrana. Križanje je proces u kojem dva roditelja

Nasumično stvori početnu populaciju
Evaluiraj početnu populaciju
ponavljaj
    Odaberi jedinke iz populacije
    Primjeni genetske operatore nad odabranim jedinkama
dok Kriterij zaustavljanja nije zadovoljen

Slika 4.1: Genetski algoritam

izmijenjuju genski materijal te stvaraju potomke koji sadrži osobine oba roditelja. Mutacija je manja slučajna izmjena kojoj su podvrgnute jedinke koja služi izbjegavanju lokalnih optimuma i očuvanju genetske raznovrsnosti.

Pseudokod jednostavnog genetskog algoritma dan je na slici 4.1.

Implementirani genetski algoritam koristi sljedeće evolucijske operatore. Operator selekcije je troturnirska selekcija uz jednostavni eliminacijski odabir. Selekcija odabire tri jedinke te odbacuje najlošiju od te tri. Na mjesto odbačene jedinke dodaje se nova jedinka dobivena križanjem dviju jedinki koje su pobijedile u turniru. Prednost turnirske selekcije je što je implicitno algoritmu daje i svojstvo elitizma, svojstvo koje osigurava preživljavanje najbolje jedinke. Korišteno križanje je jednoliko ili uniformno križanje kod kojeg gen dijeteta preslikava iz jednog od roditelja koji se nasumično odabire. Oba roditelja imaju jednaku vjerojatnost da će biti odabrana. Algoritam koristi i jednostavnu mutaciju kod koje algoritam mijenja vrijednost jednog bita s malom vjerojatnošću.

4.2. Postupno učenje zasnovano na populaciji

Kao što je već navedeno u prethodnom poglavlju, postupno učenje zasnovano na populaciji (engl. *Population Based Incremental Learning, PBIL*) je metoda rješavanja problema optimizacije kombiniranjem genetskog algoritma i natjecateljskog učenja. Genetski algoritam detaljno je opisan u prethodnom poglavlju.

Natjecateljsko učenje je nenadzirana metoda učenja često korištena za grupiranje podataka na temelju sličnosti između podataka. Metoda nema nikakvo znanje o skupu koji obrađuje poput bitnijih karakteristika ili broja skupina sličnih podataka, nego se očekuje od nje da sama odredi grupe podataka. Najpoznatiji primjeri natjecateljskog učenja su samorganizirajuće ili Kohonenove mape i kvantizacijski vektori.

Pseudokod algoritma PBIL prikazan je na slici 4.2. Kao što se vidi u pseudokodu, algoritam na početku inicijalizira vjerojatnosni vektor. Vjerojatnosni vektor nam govori kolika je vjerojatnost da je bit na odgovarajućoj poziciji postavljen na 1. Nakon svake iteracije vjerojatnosni vektor se ažurira na temelju vrijednosti najbolje jedinke i stope učenja. Vjerojatnosni vektor se postavlja na temelju sljedećeg pravila:

$$p_i = p_i * (1 - LR) + s_i * LR \tag{4.1}$$

gdje je p_i trenutna vrijednost vjerojatnosnog vektora na poziciji i, s_i vrijednost najbolje jedinke na poziciji i, a LR vrijednost stope učenja.

PBIL također koristi i mutaciju vjerojatnosnog vektora s istim ciljem kao i u genetskom algoritmu, da bi spriječio prebrzu konvergenciju i očuvao gensku raznolikost. Mutacija vjerojatnosnog vektora se izvršava po sljedećem pravilu:

$$p_i = p_i * (1 - MS) + RV * MS \tag{4.2}$$

gdje je MS korak mutacije, a RV nasumična vrijednost koja može poprimiti vrijednosti 0 i 1 s jednakom vjerojatnošću.

Postoji još nekoliko podvrsti algoritma PBIL koje se razlikuju po detaljima poput nekorištenja operatora mutacije, ažuriranja vjerojatnosnog vektora na temelju određenog broja najboljih jedinki ili ažuriranja vjerojatnosnog vektora na temelju i najboljih i najgorih jedinki. U ovom radu je implementirana prethodno opisana inačica.

4.2.1. Primjer rada algoritma

Rad algoritma postupnog učenja zasnovanog na populaciji prikazan je na primjeru s populacijom veličine 4 i jedinkama veličine 5 bitova. Radi jednostavnosti, koristi

Inicijaliziraj vjerojatnosni vektor na 0.5 za svaku varijablu
Uzorkuj populaciju na temelju vjerojatnosnog vektora
ponavljaj
    Odaberi najbolju jedinku
    Ažuriraj vjerojatnosni vektor
    Stvori novu generaciju na temelju razdiobe i zamijeni roditelje
dok Kriterij zaustavljanja nije zadovoljen

Slika 4.2: Algoritam PBIL

se binarni prikaz broja te će algoritam pokušavati minimizirati funkciju identiteta, tj. $f(x) = x$. Algoritam koristi stopu učenja od 0.1 te stopu mutacije od 0.01.

Algoritam započinje inicijalizacijom vjerojatnosnog vektora na 0.5 na svakoj poziciji.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0.5	0.5

Slika 4.3: Početni vjerojatnosni vektor

Nakon toga slijedi uzorkovanje jedinki i evaluacija funkcije. Jedinke se uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1. U sljedećoj tablici su prikazane izgenerirane jedinke. s_j označava j-ti bit jedinke.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	1	0	1	0	1	21
2	1	1	0	0	0	24
3	1	0	1	1	1	23
4	0	1	0	0	1	9

Slika 4.4: Nasumično uzorkovane jedinke

Nakon toga slijedi ažuriranje vjerojatnosnog vektora na temelju najbolje jedinke. Vidljivo je da je četvrta jedinka najbolja. Na temelju te jedinke i izraza (4.1) ažurira se

vjerojatnosni vektor. Stopa učenja LR je 0.1.

$$p_0 = p_0 * (1 - LR) + s_0 * LR = 0.5 * (1 - 0.1) + 0 * 0.1 = 0.45$$

$$p_1 = p_1 * (1 - LR) + s_1 * LR = 0.5 * (1 - 0.1) + 1 * 0.1 = 0.55$$

$$p_2 = p_2 * (1 - LR) + s_2 * LR = 0.5 * (1 - 0.1) + 0 * 0.1 = 0.45$$

$$p_3 = p_3 * (1 - LR) + s_3 * LR = 0.5 * (1 - 0.1) + 0 * 0.1 = 0.45$$

$$p_4 = p_4 * (1 - LR) + s_4 * LR = 0.5 * (1 - 0.1) + 1 * 0.1 = 0.55$$

Zadnji korak u jednoj iteraciji je mutiranje vjerojatnosnog vektora na temelju (4.2). RV je nasumična vrijednost odabrana za svaku varijablu posebno, a stopa mutacije MR je 0.01.

$$p_0 = p_0 * (1 - MR) + RV * MR = 0.45 * (1 - 0.01) + 0 * 0.01 = 0.4455$$

$$p_1 = p_1 * (1 - MR) + RV * MR = 0.55 * (1 - 0.01) + 1 * 0.01 = 0.5995$$

$$p_2 = p_2 * (1 - MR) + RV * MR = 0.45 * (1 - 0.01) + 1 * 0.01 = 0.4905$$

$$p_3 = p_3 * (1 - MR) + RV * MR = 0.45 * (1 - 0.01) + 0 * 0.01 = 0.4455$$

$$p_4 = p_4 * (1 - MR) + RV * MR = 0.55 * (1 - 0.01) + 1 * 0.01 = 0.5995$$

Dobiveni vjerojatnosni vektor je prikazan na slici 4.5.

p_0	p_1	p_2	p_3	p_4
0.4455	0.5995	0.4905	0.4455	0.5995

Slika 4.5: Vjerojatnosni vektor korišten u drugoj iteraciji

Sljedi druga iteracija. Prvi korak uzorkovanje nove generacije jedinki. Jednike se ponovo uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1. Dobivene jedinke su dane u tablici 4.6.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	0	0	1	0	0	4
2	0	1	1	0	1	13
3	1	1	0	0	1	25
4	2	1	1	0	0	28

Slika 4.6: Jedinke uzorkovane u drugoj iteraciji

Nakon toga slijedi ažuriranje vjerojatnosnog vektora na temelju najbolje jedinke. Vidljivo je da je prva jedinka najbolja. Na temelju te jedinke i izraza (4.1) ažurira se vjerojatnosni vektor. Kao što je prethodno navedeno, stopa učenja LR je 0.1

$$p_0 = p_0 * (1 - LR) + s_0 * LR = 0.445 * (1 - 0.1) + 0 * 0.1 = 0.400$$

$$p_1 = p_1 * (1 - LR) + s_1 * LR = 0.599 * (1 - 0.1) + 0 * 0.1 = 0.544$$

$$p_2 = p_2 * (1 - LR) + s_2 * LR = 0.490 * (1 - 0.1) + 1 * 0.1 = 0.541$$

$$p_3 = p_3 * (1 - LR) + s_3 * LR = 0.445 * (1 - 0.1) + 0 * 0.1 = 0.400$$

$$p_4 = p_4 * (1 - LR) + s_4 * LR = 0.599 * (1 - 0.1) + 0 * 0.1 = 0.544$$

Zadnji korak u jednoj iteraciji je mutiranje vjerojatnosnog vektora na temelju (4.2). RV je nasumična vrijednost odabrana za svaku varijablu posebno, a stopa mutacije MR je 0.01.

$$p_0 = p_0 * (1 - MR) + RV * MR = 0.400 * (1 - 0.01) + 0 * 0.01 = 0.396$$

$$p_1 = p_1 * (1 - MR) + RV * MR = 0.544 * (1 - 0.01) + 1 * 0.01 = 0.548$$

$$p_2 = p_2 * (1 - MR) + RV * MR = 0.4541 * (1 - 0.01) + 0 * 0.01 = 0.449$$

$$p_3 = p_3 * (1 - MR) + RV * MR = 0.400 * (1 - 0.01) + 0 * 0.01 = 0.396$$

$$p_4 = p_4 * (1 - MR) + RV * MR = 0.544 * (1 - 0.01) + 1 * 0.01 = 0.548$$

Dobiveni vjerojatnosni vektor koji se koristi u sljedećoj iteraciji dan je na slici 4.7.

p_0	p_1	p_2	p_3	p_4
0.4455	0.5995	0.4905	0.4455	0.5995

Slika 4.7: Vjerojatnosni vektor u drugoj iteraciji

Algoritam se nastavlja izvoditi sve dok nije ispunjen uvjet zaustavljanja.

4.3. Kompaktni genetski algoritam

Kompaktni genetski algoritam (engl. *Compact Genetic Algorithm, cGa*) razvijen je na temelju modela nasumične šetnje (engl. *Random Walk Model*) predloženog u [7]. Modelom nasumične šetnje analiziran je skup problema sa svojstvom da su građeni od nezavisnih građevnih blokova. Na temelju tog uvjeta, autori su pretpostavili da se svaki dio rješenja može rješiti nezavisno. Koristeći model nasumične šetnje, autori su uspjeli predvidjeti konvergenciju genetskog algoritma. Dobiveni rezultati sugeriraju da je moguće simulirati njegovo ponašanje za probleme s nezavisnim građevnim blokovima na temelju čega je razvijen cGa.

Ideja iza cGa je bilježenje populacije kao vjerojatnosnog vektora čime se smanjuje memorijska potrošnja te manipuliranje tim vjerojatnosnim vektorom na način da se imitira rad jednostavnog genetskog algoritma. Naravno, zbog pretpostavke da se svaki dio rješenja može rješiti nezavisno, cGa imitira samo rad algoritma nad nezavisnim građevnim blokovima. Križanje je imitirano uzorkovanjem jedinki na temelju vjerojatnosne distribucije. Naime, višestrukim korištenjem operatora križanja zapravo se smanjuje korelacija između gena jedinke, samim time možemo uzorkovanje iz vjerojatnosnog vektora vidjeti kao prečac k učinku križanja.

Psuedokod cGa je dan na slici 4.8. Algoritam generira dvije jedinke na temelju vjerojatnosnog vektora. Nakon toga uspoređuju se dobrote dobivenih jedinki, te se temelju dobrote određuju pobjednik i gubitnik. Zatim se ažurira vjerojatnosni vektor na temelju sljedećeg pravila:

$$p_i = p_i + LR(W_i - L_i) \tag{4.3}$$

gdje je W_i gen pobjedničke jedinke, a L_i gen gubitničke jedinke na poziciji i. Stopa učenja je označena sa LR i obično se uzima kao $1/n$ gdje je n veličina populacije. Iako populacija u cGa zapravo ne postoji, uzima se veličina populacije jednostavnog genetskog algoritma koju cGa želi simulirati.

cGa simulira rad genetskog algoritma s velikom prednošću u smanjenoj potrošnji memorije. Nedostatak u odnosu na konvencionalni genetski algoritam je nemogućnost prilagodbe algoritma u slučaju specifičnog problema. Također, cGa nije prilagođen za korištenje na probleme gdje postoji zavisnost između dijelova rješenja što ga čini pogodnim samo sa specifičan skup problema.

Inicijaliziraj vjerojatnosni vektor na 0.5 za svaku varijablu
ponavljaj
    Uzorkuj 2 rješenja
    Na temelju dobrote rješenja, odredi gubitnika i pobjednika
    Ažuriraj vjerojatnosni vektor
dok Kriterij zaustavljanja nije zadovoljen

Slika 4.8: Algoritam cGA

4.3.1. Primjer rada algoritma

Rad kompaktnog genetskog algoritma prikazan je na primjeru s veličinom pretpostavljene populacije $n = 10$ i jedinkama veličine 5 bitova. Vrijednost parametra LR je $1/n = 1/10 = 0.1$. Radi jednostavnosti, koristi se binarni prikaz broja te će algoritam pokušavati minizirati funkciju identiteta, tj. $f(x) = x$.

Algoritam započinje inicijalizacijom vjerojatnosnog vektora na 0.5 na svakoj poziciji.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0.5	0.5

Slika 4.9: Početni vjerojatnosni vektor

Nakon toga se na temelju vjerojatnosnog vektora uzorkuje populacija. Kao i kod algoritma PBIL, jedinke se uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	1	0	1	0	1	21
2	0	1	0	0	1	9

Slika 4.10: Nasumično uzorkovane jedinke

Zatim se na temelju (4.3) ažurira vjerojatnosni vektor. Stopa učenja LR je 0.1.

$$p_0 = p_0 + LR(s_{2,0} - s_{1,0}) = 0.5 + 0.1 * (0 - 1) = 0.4$$
$$p_1 = p_1 + LR(s_{2,1} - s_{1,1}) = 0.5 + 0.1 * (1 - 0) = 0.6$$

$$p_2 = p_2 + LR(s_{2,2} - s_{1,2}) = 0.5 + 0.1 * (0 - 1) = 0.4$$
$$p_3 = p_3 + LR(s_{2,3} - s_{1,3}) = 0.5 + 0.1 * (0 - 0) = 0.5$$
$$p_4 = p_4 + LR(s_{2,4} - s_{1,4}) = 0.5 + 0.1 * (1 - 1) = 0.5$$

Dobiveni vjerojatnosni vektor je dan na slici 4.11.

p_0	p_1	p_2	p_3	p_4
0.4	0.6	0.4	0.5	0.5

Slika 4.11: Vjerojatnosni vektor na početku druge iteracije cGa

Druga iteracija započinje uzorkovanjem dviju jedinki. Jednike se ponovo uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	0	0	1	0	1	5
2	0	1	0	1	1	11

Slika 4.12: Nasumično uzorkovane jedinke

Zatim se na temelju (4.3) ažurira vjerojatnosni vektor. Stopa učenja LR je 0.1.

$$p_0 = p_0 + LR(s_{2,0} - s_{1,0}) = 0.4 + 0.1 * (1 - 1) = 0.4$$
$$p_1 = p_1 + LR(s_{2,1} - s_{1,1}) = 0.6 + 0.1 * (0 - 1) = 0.5$$
$$p_2 = p_2 + LR(s_{2,2} - s_{1,2}) = 0.4 + 0.1 * (1 - 0) = 0.5$$
$$p_3 = p_3 + LR(s_{2,3} - s_{1,3}) = 0.5 + 0.1 * (0 - 1) = 0.4$$
$$p_4 = p_4 + LR(s_{2,4} - s_{1,4}) = 0.5 + 0.1 * (1 - 1) = 0.5$$

Dobiveni vjerojatnosni vektor je dan na slici 4.13.

p_0	p_1	p_2	p_3	p_4
0.4	0.5	0.5	0.4	0.5

Slika 4.13: Vjerojatnosni vektor nakon druge iteracije cGa

Algoritam se nastavlja s uzorkovanjem jedinki u trećoj iteraciji te se izvodi sve dok nije ispunjen uvjet zaustavljanja.

4.4. Algoritam univarijantne marginalne distribucije

Algoritam univarijantne marginalne distribucije (engl. *Univariate Marginal Distribution Algorithm, UMDA*) predložen je 1996. u [11]. UMDA je vrlo jednostavan algoritam sličan PBIL-u s glavnom razlikom u tome što PBIL pamti svoj vjerojatnosni vektor i ažurira ga, dok UMDA računa vjerojatnosni vektor iznova u svakoj iteraciji. UMDA, kao i PBIL, pretpostavlja nezavisnost između svih varijabli rješenja. Postoje brojna proširenja algoritma koja pretpostavljaju zavisnost varijabli, neki od njih poput BMDA, FDA i LFDA su opisani u okviru ovog rada.

Kao što je već navedeno, UMDA koristi vjerojatnosni vektor za prikaz rješenja. Kao i kod PBIL algoritma, vektor govori kolika je vjerojatnost da je bit odgovarajućoj poziciji postavljen na 1. Prilikom inicijalizacije algoritma, vektor je postavljen tako da svi bitovi imaju vjerojatnost 0.5 za poprimanje vrijednosti 1. Zatim kreće iteriranje algoritma. Nakon uzorkovanja jedinki na temelju vjerojatnosnog vektora, odabire se određeni broj najboljih jedinki koji je u nastavku označen s M. Na temelju tih jedinki stvara se novi vjerojatnosni vektor. Vjerojatnosni vektor stvara se po sljedećem izrazu:

$$p_i = 1/M * \sum_{j=1}^{M} s_{j,i} \tag{4.4}$$

gdje je p_i vjerojatnosni vektor na poziciji i, M broj jedinki na temelju kojih računamo vjerojatnosni vektor, a $s_{j,i}$ vrijednost $j - te$ jedinke na odgovarajućoj poziciji.

Kao i kod prethodno navedenih algoritama, postoje različite implementacije UMDA algoritma. Neke od implementacija koriste mutaciju kako bi održale raznolikost populacije, dok neke koriste memoriju kako bi poboljšale kreiranje vjerojatnosnog vektora. Implementacija koju obrađujemo u ovom radu ne sadrži ni mutaciju ni memoriju, međutim eksplicitno je dodan elitizam koji održava najbolju jedinku na životu nakon svake iteracije.

Inicijaliziraj vjerojatnosni vektor na 0.5 za svaku varijablu
Uzorkuj populaciju na temelju vjerojatnosnog vektora
ponavljaj
    Odaberi M najboljih rješenja
    Ažuriraj razdiobu univarijantne vjerojatnosti
    Stvori novu generaciju na temelju razdiobe i zamijeni roditelje
dok Kriterij zaustavljanja nije zadovoljen

Slika 4.14: Algoritam UMDA

4.4.1. Primjer rada algoritma

Rad algoritma univarijantne marginalne distribucije prikazan je na primjeru s populacijom veličine 4 i jedinkama veličine 5 bitova. Radi jednostavnosti, koristi se binarni prikaz broja te će algoritam pokušavati minizirati funkciju identiteta, tj. $f(x) = x$. Obećavajućim rješenjima smatraju se dva najbolja rješenja.

Algoritam započinje inicijalizacijom vjerojatnosnog vektora na 0.5 na svakoj poziciji.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0.5	0.5

Slika 4.15: Početni vjerojatnosni vektor

Nakon toga slijedi uzorkovanje jedinki i evaluacija funkcije. Kao i kod algoriatma PBIL i cGa, jedinke se uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1. U sljedećoj tablici su prikazane izgenerirane jedinke. s_j označava j-ti bit jedinke.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	1	0	1	0	1	21
2	1	1	0	0	0	24
3	1	0	1	1	1	23
4	0	1	0	0	1	9

Slika 4.16: Nasumično uzorkovane jedinke

Nakon toga slijedi ažuriranje vjerojatnosnog vektora na temelju dviju najboljih jedinke. Vidljivo je da su najbolje prva i četvrta jedinka. Na temelju te jedinke i izraza (4.4) računa se vjerojatnosni vektor.

$$p_0 = (s_{1,0} + s_{2,0})/2 = (1 + 0)/2 = 0.5$$

$$p_1 = (s_{1,1} + s_{2,1})/2 = (0 + 1)/2 = 0.5$$

$$p_2 = (s_{1,2} + s_{2,2})/2 = (1 + 0)/2 = 0.5$$

$$p_3 = (s_{1,3} + s_{2,3})/2 = (0 + 0)/2 = 0$$

$$p_4 = (s_{1,4} + s_{2,4})/2 = (1 + 1)/2 = 1$$

Dobiveni vjerojatnosni vektor je dan na slici 4.17.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0	1

Slika 4.17: Vjerojatnosni vektor korišten u drugoj iteraciji

Druga iteracija započinje s uzorkovanjem jedinki na temelju vjerojatnosnog vektora. Ponovo se jedinke uzorkuju na temelju vjerojatnosnog vektora koji nam govori kolika je vjerojatnost da je bit na poziciji i jednak 1.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	0	0	0	0	1	1
2	1	0	1	0	1	21
3	0	1	1	0	1	13
4	1	1	1	0	1	29

Slika 4.18: Nasumično uzorkovane jedinke u drugoj iteraciji

Nakon toga slijedi ažuriranje vjerojatnosnog vektora na temelju dviju najboljih jedinke. Vidljivo je da su najbolje prva i treća jedinka. Na temelju te jedinke i izraza (4.4) računa se vjerojatnosni vektor.

$$p_0 = (s_{1,0} + s_{2,0})/2 = (0 + 0)/2 = 0$$

$$p_1 = (s_{1,1} + s_{2,1})/2 = (0 + 1)/2 = 0.5$$

$$p_2 = (s_{1,2} + s_{2,2})/2 = (0 + 1)/2 = 0.5$$

$$p_3 = (s_{1,3} + s_{2,3})/2 = (0 + 0)/2 = 0$$

$$p_4 = (s_{1,4} + s_{2,4})/2 = (1 + 1)/2 = 1$$

Dobiveni vjerojatnosni vektor je dan na slici 4.19.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0	1

Slika 4.19: Vjerojatnosni vektor izračunat u drugoj iteraciji

Algoritam se nastavlja s uzorkovanjem jedinki u trećoj iteraciji te se izvodi sve dok nije ispunjen uvjet zaustavljanja.

4.5. Algoritam bivarijantne marginalne distribucije

Algoritam bivarijantne marginalne distribucije (engl. *Bivariate Marginal Distribution Algorithm, BMDA*) nadogradnja je prethodno opisanog algoritma UMDA. Za razliku od njega, BMDA pretpostavlja zavisnost između dvije varijable, odnosno građevne blokove veličine 2. Algoritam se sastoji od dvije bitne cjeline: stvaranja grafa zavisnosti i uzorkovanja jedinki na temelju tog grafa zavisnosti. Za izračun zavisnosti koristi se Pearsonova χ^2 statistika i Bayesov teorem. Detaljan opis Pearsonove statistike dan je u [6].

Slično kao i u UMDA-i, algoritam sadrži vjerojatnosni vektor koji govori kolika je vjerojatnost da je bit odgovarajućoj poziciji postavljen na 1. Međutim, BMDA sadrži i matricu vjerojatnosti koja govori kolika je vjerojatnost da su dva bita istovremeno postavljena na 1. Pomoću tih dviju vjerojatnosti i Bayesovog teorema možemo izračunati uvjetnu vjerojatnost:

$$p(x_i|x_j) = \frac{p(x_i, x_j)}{p(x_j)} \tag{4.5}$$

Zatim algoritam korist Pearsonvu χ^2 statistiku kako bi izračunao zavisnost između varijabli $p(x_i)$ i $p(x_j)$. Statistika se računa po:

$$\chi^2 = \sum_{i,j} \frac{(D - O)^2}{O}$$

gdje je D dobivena zajednička vjerojatnost, a O očekivana zajednička vjerojatnost.

U slučaju nezavisnosti varijabli, jednostavno je izračunati očekivanu zajedničku vjerojatnost kao $p(x_i) * p(x_j)$. Dobivenu zajednička vjerojatnost je vjerojatnost dobivena iz populacije najboljih rješenja prethodno izračunata i spremljena u matricu vjerojatnosti. Izraz za izračun pojedine vrijednosti je:

$$\chi^2_{i,j} = \frac{M(p(x_i, x_j) - p(x_i) * p(x_j))^2}{p(x_i) * p(x_j)} \tag{4.6}$$

gdje je M broj odabranih obećavajućih jedinki.

Pearsonova χ^2 statistika kaže da su dvije varijable 95% nezavisne ako vrijedi:

$$\chi^2_{i,j} < 3.84$$

Sljedeći korak je stvaranje grafa zavisnosti. Ideja je vrlo jednostavna, stvori graf koji prikazuje zavisne varijable kao povezane komponente grafa, a nezavisne kao nepovezane. Varijable rješenja prikazujemo kao vrhove grafa, a zavisnost između varijabli kao bridove grafa. Dobiveni graf će sigurno biti stablo, tj. nepovezani graf bez ciklusa.

Uzorkuj populaciju sa uniformnom razdiobom.
ponavljaj
Odaberi određeni broj jedinki koje predstavljaju obećavajuća rješenja
Izračunaj vjerojatnosni vektor($p(x_i)$) na temelju obećavajućih rjepenja
Izračunaj vjerojatnosnu matricu($p(x_i, x_j)$) na temelju obećavajućih rješenja
Generiraj vjerojatnosnu razdiobu pomoću Pearsonove chi-square statisktike.
Izbaci K jedinki te na njihovo mjesto ubaci jedinke generirane na temelju generirane razdiobe
dok Kriterij zaustavljanja nije zadovoljen

Slika 4.20: Algoritam BMDA

Graf se stvara pomoću tri skupa; skup koji sadrži sve vrhove, skup koji sadrži sve bridove i skup koji sadrži po jedan vrh iz svake nezavisne komponente stabla. Algoritam je sljedeći: uzmi jedan vrh iz skupa vrhova, dodaj ga u skup nezavisnih komponenti i pronađi sve vrhove zavisne o njemu. Zatim dodaj bridove između tih vrhova u skup bridova. Ponavljaj dok skup vrhova ne bude prazan.

Stvoreni graf zavisnosti se koristi za uzorkovanje novih jedinki. Nove jedinke stvaraju se na sljedeći način: nasumično odaberi jednu varijablu rješenja koja još nije postavljena, postavi ju na 1 s vjerojatnošću zapisanom u vjerojatnosnom vektoru, te postavi sve varijable rješenja koje su zavisne o odabranoj varijabli. Zavisne varijable postavi na vjerojatnost izračunatom pomoću Bayesovog teorema. Algoritam se ponavlja dok sve jedinke nisu uzorkovane.

4.5.1. Primjer rada algoritma

Rad algoritma bivarijantne marginalne distribucije prikazan je na primjeru s populacijom veličine 6 i jedinkama veličine 5 bitova. Radi jednostavnosti, koristi se binarni prikaz broja te će algoritam pokušavati minizirati funkciju identiteta, tj. $f(x) = x$. Vjerojatnosni vektor ažurirati će se na temelju cijele populacije, kako bi pokazali slučaj u kojem postoji zavisnost između varijabli, iako ažuriranje na temelju cijele populacije nije uobičajeno ni u BMDA ni općenito u PMBGA.

Algoritam započinje inicijalizacijom vjerojatnosnog vektora na 0.5 na svakoj poziciji.

p_0	p_1	p_2	p_3	p_4
0.5	0.5	0.5	0.5	0.5

Slika 4.21: Početni vjerojatnosni vektor

Nakon toga slijedi uzorkovanje jedinki i evaluacija funkcije. U sljedećoj tablici su prikazane izgenerirane jedinke. s_j označava j-ti bit jedinke.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	1	0	0	0	1	17
2	1	1	0	0	0	24
3	1	0	1	1	1	23
4	0	1	0	0	1	9
5	1	1	0	0	1	25
6	0	1	0	0	0	8

Slika 4.22: Nasumično uzorkovane jedinke

Sljedeći korak je izračun vjerojatnosnog vektora koji se računa po (4.4) te se koristi u izračunu matrice vjerojatnosti.

Vjerojatnosni vektor

p_0	p_1	p_2	p_3	p_4
0.66	0.66	0.16	0.16	0.66

Nakon toga slijedi izračun matrice vjerojatnosti. Matrica vjerojatnosti se računa na isti način kao i vjerojatnosni vektor jedino što provjerava da su dva bita istodobno postavljena na 1.

Matrica vjerojatnosti

Bit	s_0	s_1	s_2	s_3	s_4
s_0	0.66	0.33	0.16	0.16	0.5
s_1	0.33	0.66	0	0	0.33
s_2	0.16	0	0.16	0.16	0.16
s_3	0.16	0	0.16	0.16	0.16
s_4	0.5	0.33	0.16	0.16	1

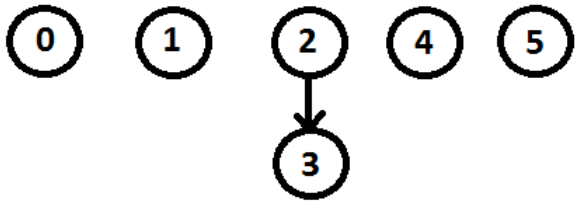
Nakon što smo dobili matricu vjerojatnosti, računamo χ^2 statistiku na temelju (4.6) za svake dvije varijable. Radi preglednosti dan je prikaz izračuna jednog nezavisnog para varijabli i jednog (ujedno i jedinog) zavisnog para varijabli.

$$\chi^2_{0,1} = \frac{M(p(x_0, x_1) - p(x_0) * p(x_1))^2}{p(x_0) * p(x_1)} = 6 * (0.33 - 0.44)^2/0.44 = 0.165$$

$$\chi^2_{2,3} = \frac{M(p(x_0, x_1) - p(x_0) * p(x_1))^2}{p(x_0) * p(x_1)} = 6 * (0.166 - 0.027)^2/0.027 \approx 4.29$$

Na temelju Pearsonove χ^2 statistike zaključujemo da su bitovi s_0 i s_1 nezavisni, dok su s_2 i s_3 zavisni. Izračunom preostale χ^2 statistike dolazi se do zaključka da su samo s_2 i s_3 zavisni.

Sljedeći korak je izgradnja grafa zavisnosti. Nasumično odabiremo jednu varijablu, npr. varijablu s_1. Varijabla s_1 je nezavisna te će u grafu biti prikazan kao nepovezana komponenta. Zatim uzimamo varijablu s_2. Varijabla s_2 je zavisna o varijabli s_3, te ih spajamo u grafu zavisnosti. Više ni jedna varijabla nije zavisna o varijablama s_2 i s_3 te odabiremo neku od preostalih varijabli. Odabiremo varijablu s_0 te je ona kao i varijabla s_1 nezavisna i prikazuje se kao nepovezana komponenta. Zatim uzimamo jedinu preostalu varijablu, s_4 koja je ponovo nezavisna i prikazuje se na isti način. Dobiveni graf je prikazan na slici 4.23.



Slika 4.23: Graf zavisnosti

Sljedeća iteracija započinje stvaranjem nove populacije na temelju grafa zavisnosti, vektora vjerojatnosti, matrice vjerojatnosti i Bayesovog teorema. Nasumično se odabire jedna varijabla, npr. varijabla s_2. Varijablu s_2 postavljamo na temelju vjerojatnosti iz vjerojatnosnog vektora. Zatim odabiremo jednu od varijabli koje je zavisna od varijable s_2. Jedina varijabla zavisna o varijabli s_2 je varijabla s_3. Varijablu s_3 postavljamo na temelju Bayesovog teorema(4.6) koristeći vjerojatnosti iz vjerojatnosnog vektora i vjerojatnosne matrice. Kako više nema varijabli zavisnih o varijablama s_2 ili s_3, odabire se nova varijabla. Nasumično je odabrana varijabla s_1 koju postavljamo

na temelju vjerojatnosnog vektora. Kako ni jedna varijabla nije ovisna o varijabli s_1, ponovo se nasumično odabire nova varijabla i uzorkuje se na temelju vjerojatnosnog vektora. Uzorkovanje jedinke se nastavlja sve dok cijela jedinka nije uzorkovana.

S obzirom da su sve varijable osim varijable s_3 nezavisne, njih uzorkujemo pomoću vjerojatnosnog vektora. Kao i kod prethodno opisanih algoritama, bit na odgovarajućoj poziciji postavlja se na 1 s vjerojatnošću zapisanom na odgovarajućem mjestu u vjerojatnosnom vektoru. Varijabla s_3 je zavisna o varijabli s_2 te nju računamo pomoću Bayesovog teorema:

$$p(x_i|x_j) = \frac{p(x_i, x_j)}{p(x_j)} = \frac{0.16}{0.16} = 1$$

ako je vrijednost varijable s_2 1, te

$$p(x_i|x_j) = \frac{p(x_i, x_j)}{p(x_j)} = \frac{0.16}{0.84} = 0.19$$

ako je vrijednost varijable s_2 0.

Uz pomoć tih vjerojatnostni uzorkujemo nove jedinke. Uzorkovane jedinke su prikazane na slici 4.24.

Jedinka	s_0	s_1	s_2	s_3	s_4	f(x)
1	0	1	0	0	0	8
2	1	0	0	1	0	18
3	1	1	0	0	1	25
4	1	0	0	1	1	19
5	0	0	0	0	1	1
6	1	1	1	0	1	29

Slika 4.24: Populacija uzorkovana u drugoj iteraciji

Slijedi računanje vjerojatnosnog vektora i vjerojatnosne matrice.

p_0	p_1	p_2	p_3	p_4
0.66	0.5	0.16	0.33	0.33

Slika 4.25: Vjerojatnosni vektor u drugoj iteraciji

Matrica vjerojatnosti

Bit	s_0	s_1	s_2	s_3	s_4
s_0	0.66	0.33	0.16	0.33	0.16
s_1	0.33	0.5	0.16	0	0.16
s_2	0.16	0.16	0.16	0	0.16
s_3	0.33	0	0	0.16	0
s_4	0.16	0.16	0.16	0	0.33

Nakon izračuna vjerojatnosnog vektora ponovo slijedi stvaranje grafa zavisnosti i uzorkovanje novih jedinki na temelju razdiobe opisane vjerojatnosnim vektorom, vjerojatnosnom matricom i grafom zavisnosti. Algoritam se nastavlja izvoditi sve dok nisu zadovoljeni uvjeti zaustavljanja.

5. Vrednovanje algoritama

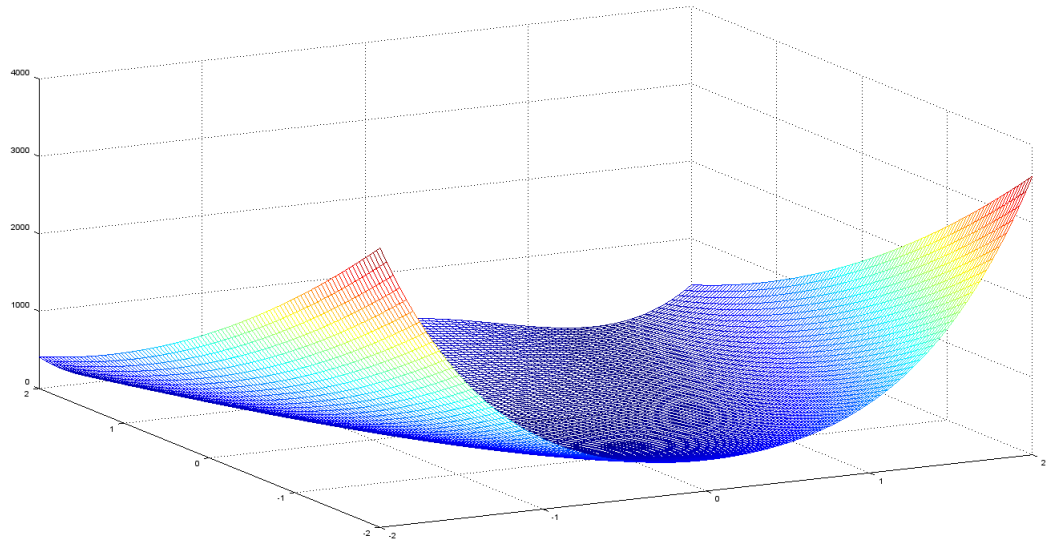
5.1. Ispitne funkcije

Algoritmi su testirani na četiri optimizacijske funkcije koje su odabrane zbog težine pronalaska optimalnog rješenja. Odabrane funkcije često su korištene u literature te su odabrane zbog različitih karakteristika.

Odabrane funkcije su:

- Rosenbrockova dolina,
- Rastriginova funkcija,
- Schewefelova funkcija i
- decepcijska funkcija trećeg reda.

5.1.1. Rosenbrockova dolina



Slika 5.1: Rosenbrockova dolina

Rosenbrockova dolina, poznata kao i "banana funkcija", klasični je problem u optimizaciji. Funkcija je unimodalna i područje u kojem se nalazi globalni optimum je vrlo lako pronać. Unatoč tome, funkcija je optimizacijski zahtjevna jer globalni optimum leži u dugačkoj, uskoj ravnini oblika parabole.

Rosenbrockova dolina opisana je s:

$$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 + x_i)^2]$$

Globalni minimum funkcije je u točki:

$$\overrightarrow{x} = [1, 1, 1..1]$$

Vrijednost funkcije u minumumu je:

$$f(\overrightarrow{x}) = 0$$

Pretraživano područje rješenja je:

$$-10 < x < 10$$

s preciznošću od 0.01.

5.1.2. Rastriginova funkcija

Rastriginova funkcija također je često korištena prilikom evaulacije optimizacijskih algoritama. Dobivena je tako da se unimodalnoj funkciji dodala modulacija kosinusa kako bi se stvorili česti lokalni optimumi. Iako je funkcija izrazito višemodalna, optimumi su pravilno raspoređeni u prostoru.

Rastriginova funkcija je:

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10cos(2\pi x_i)]$$

Globalni minimum funkcije je u točki:

$$\overrightarrow{x} = [0, 0, 0..0]$$

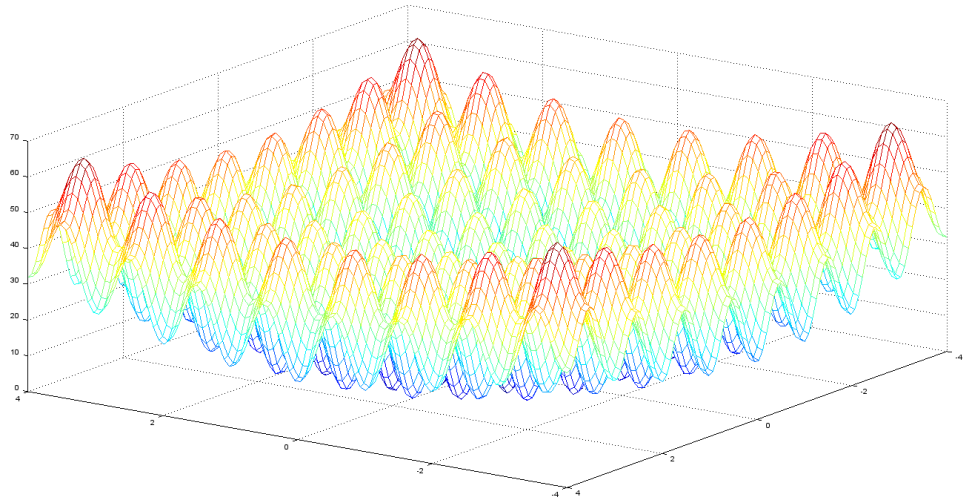
Vrijednost funkcije u minumumu je:

$$f(\overrightarrow{x}) = 0$$

Pretraživano područje rješenja je:

$$-10 < x < 10$$

s preciznošću od 0.01.

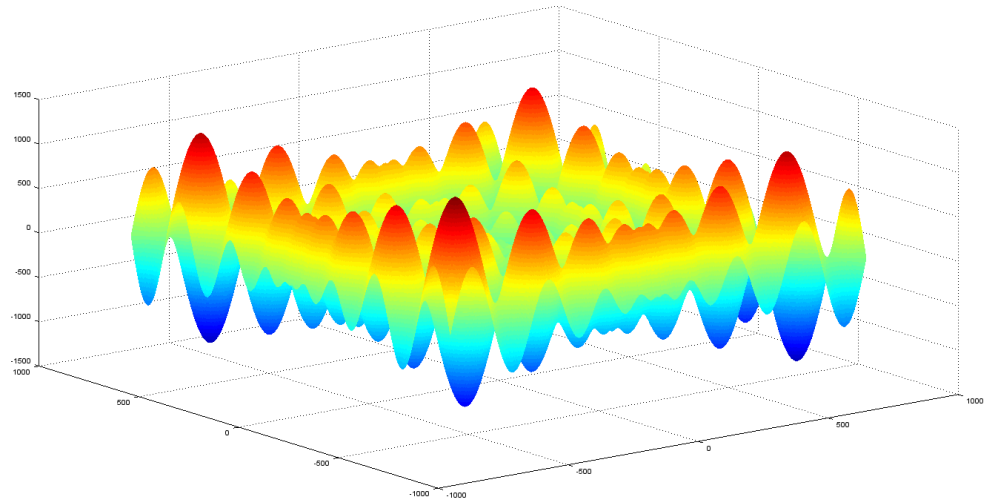


Slika 5.2: Rastriginova funkcija

5.1.3. Schwefelova funkcija

Kao i prethodne funkcije, Schwefelova je također često korištena prilikom evaluacije optimizacijskih algoritama. Funkcija je, kao i Rastriginova, multimodalna, ali razlika je u tome što Schwefelova funkcija ima globalni optimum udaljen u prostoru od lokalnih optimuma što ju čini težom za optimizaciju.

$$f(x) = -\sum_{i=1}^{n}[-x_i sin(\sqrt{(|x_i|)}]$$



Slika 5.3: Schwefelova funkcija

Globalni minimum funkcije je u točki:

$$\overrightarrow{x} = [420.96, 420.96, 420.96..420.96]$$

Vrijednost funkcije u minumumu je:

$$f(\overrightarrow{x}) = 0$$

Pretraživano područje rješenja je:

$$400 < x < 450$$

s preciznošću od 0.01.

5.1.4. Decepcijska funkcija trećeg reda

Decepcijski problemi su oni problemi koji "zavaravaju" optimizacijske algoritme. Uobičajeno takve funkcije imaju područje lokalnih optimuma značajno veće nego područje globalnog optimuma. Prethodno navedena Schwefelova funkcija je primjer decepcijske funkcije.

Decepcijska funkcija na kojoj smo izvršili evaluaciju je:

$$f(n) = \begin{cases} -x/\alpha_i & \text{if } 0 \leq x_i \leq 4/5\alpha_i \\ 5x/\alpha_i - 4 & \text{if } 4/5\alpha_i \leq x_i \leq \alpha_i \\ 5(x-\alpha_i)/(\alpha_i - 1) + 1 & \text{if } \alpha_i \leq x_i \leq (1+4\alpha_i)/5\alpha_i \\ (x-1)/(1-\alpha_i) + 4/5 & \text{if } (1+4\alpha_i)/5 \leq x_i \leq 1 \end{cases}$$

Globalni minimum funkcije je u točki:

$$\overrightarrow{x} = [\alpha_1, \alpha_2, \alpha_3 ... \alpha_n]$$

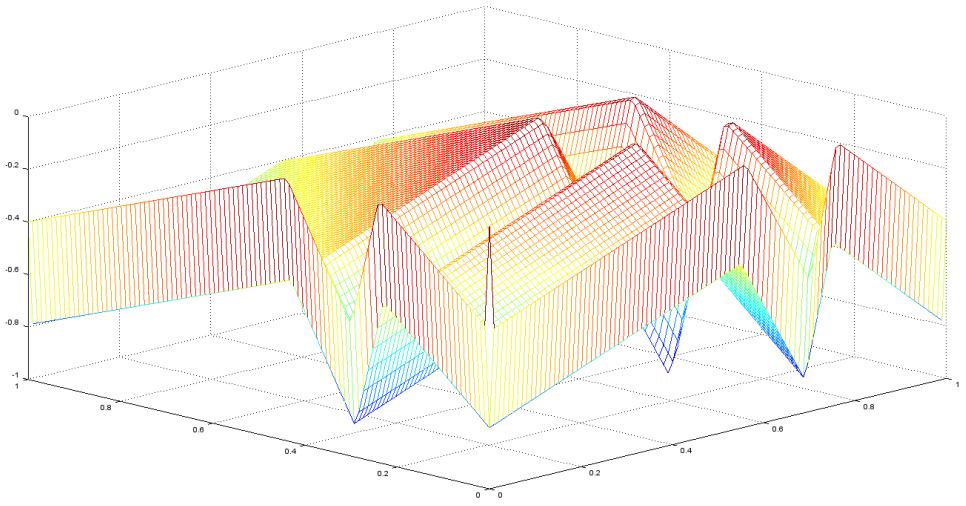
Vrijednost funkcije u minumumu je:

$$f(\overrightarrow{x}) = -1$$

Pretraživano područje rješenja je:

$$0 < x < 1$$

s preciznošću od 0.01.



Slika 5.4: Decepcijska funkcija tipa 3 s $\alpha_1 = 0.7$ i $\alpha_2 = 0.3$

5.2. Prikaz rješenja

Za prikaz rješenja optimizacijskih funkcija koristi se Grayev kod. Grayev kod je binarni sustav prikaza brojeva kod kojeg se dva susjedna broja uvijek razlikuju u točno jednom bitu. Zbog tog svojstva Grayev kod izbjegava Hammingove litice, slučaj u kojem je prikaz dva susjedna broja komplementaran u binarnom prikazu, što ga čini pogodnim prikazom rješenja u optimizacijskim problemima. U tablici 5.1 dana je usporedba binarnog i Grayevog koda nad brojevima koji se mogu prikazati sa 4 bita.

5.3. Parametri algoritama

Svi opisani algoritmi sadrže neke parametre koji određuju ponašanje algoritma. Parametri su odabrani na temelju iskustva, nije eksperimentalno pokazano da algoritmi s ovim paramterima daju bolje rješenje od ostalih, međutim iz iskustva se može zaključiti da daju dovoljno dobra rješenja. Dva parametra sadrže svi algoritmi u testiranju te su u okviru ovog rada isti za sve algoritme. To je kriterij zaustavljanja i veličina populacije. Kriterij zaustavljanja je određen brojem iteracija, točnije svaki algoritam se izvršavao točno 1000 iteracija. Veličina populacije u svim algoritmima je 50.

Parametri genetskog algoritma su vjerojatnost mutacije i vjerojatnost križanja. U [17] navedeno je da su najčešće korištene vjerojatnosti između 60% i 90% za križanje i između 1% i 5% za mutaciju. U ovom radu je ispitan genetski algoritam s 75% vjerojatnosti križanja i 3% vjerojatnosti mutacije.

Parametri specifični za PBIL su stopa učenja i stopa mutacije. U [1] predložene su

Decimalni broj	Binarni kod	Grayev kod
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Tablica 5.1: Usporedba prikaza broja binarnim i Grayevim kodom

sljedeće vrijednosti za stopu učenja: između 0.05 i 0.3 te između 0.01 i 0.1 za stopu mutacije. Programski ostvareni algoritam koristi 0.3 za stopu učenja i 0.01 za stopu mutacije.

Kompaktni genetski algoritam koristi samo parametar stope učenja. Vrijednost parametra je postavljena na $1/P$ gdje je P veličina populacije genetskog algoritma koji cGA simulira. Veličina populacije je kao što je prethodno navedeno 50. Bitno je napomenuti da cGA ne održava cijelu populaciju u memoriji nego samo vjerojatnosni vektor. Veličina populacije koristi se samo za određivanje stope učenja.

UMDA i BMDA algoritmi koriste samo parametar koji određuje na temelju koliko najboljih rješenja se računa vjerojatnosni vektor. Oba algoritma računaju na temelju petine populacije, odnosno u ovom slučaju na temelju najboljih 10 jedinki u populaciji.

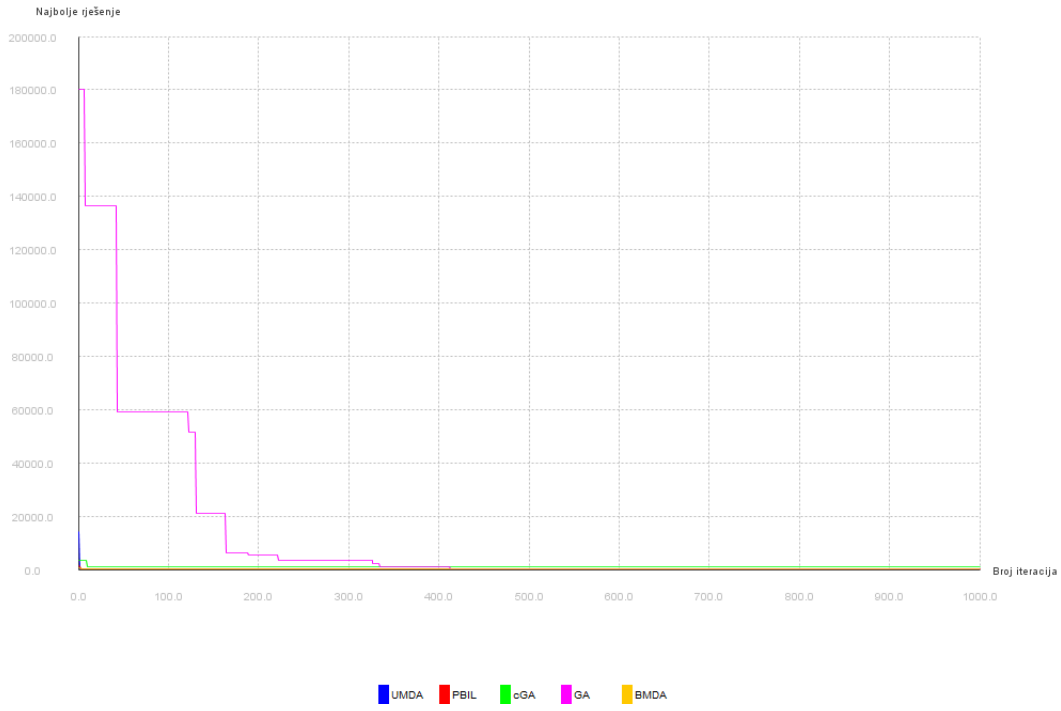
5.4. Rezultati

Algoritmi su testirani na prethodno opisanim funkcijama na problemima dimenzija 1, 3, 6 i 10. Međutim, svi algoritmi su probleme dimenzija 1 i 3 rješili u svega neko-

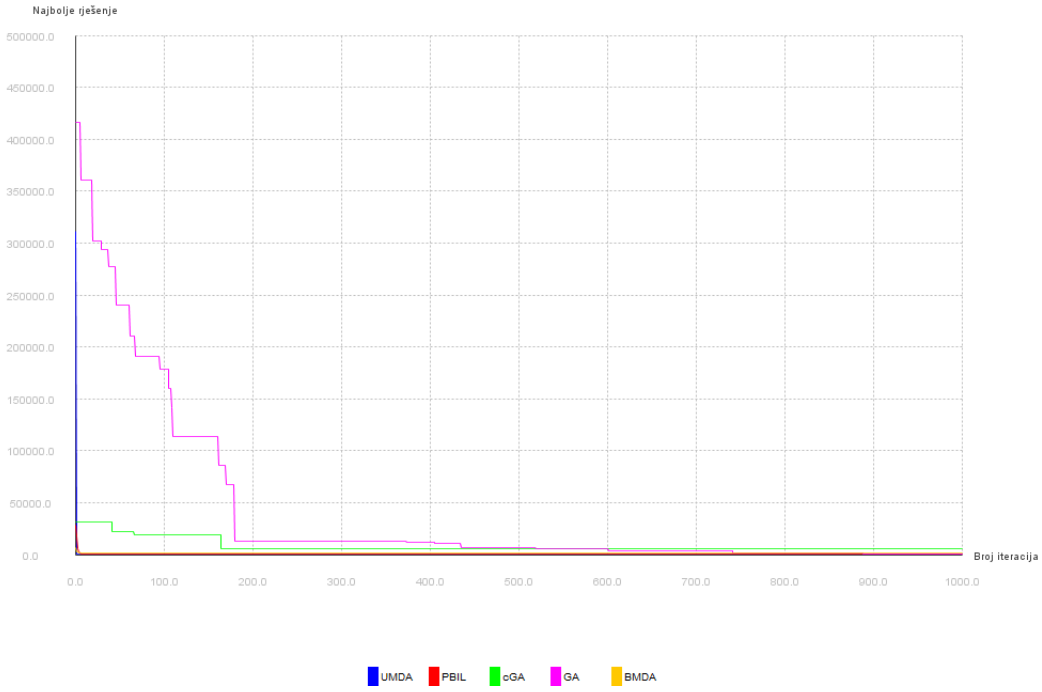
liko iteracija te zbog toga rezultati nisu predstavljeni. Rezultati optimizacije problema dimenzija 6 i 10 su predstavljeni u nastavku. Svi algoritmi su izvršeni 31 put te je odabran medijan njihovih rješenja. U nastavku ovog poglavlja slijedi analiza dobivenih rezultata.

5.4.1. Rosenbrockova dolina

Kao što je navedeno, Rosenbrockova dolina je unimodalna funkcija s blagim spustom prema globalnom optimumu što ju čini zahtjevnom za optimizaciju. Kao što se vidi iz priloženih grafova, svi testirani algoritmi došli su do globalnog optimuma. Najlošije rezultate ima jednostavni genetski algoritam, dok su probabilistički algoritmi dolazili do optimalnog rješenja u svega nekoliko iteracija. Od probabilističkih algoritama najsporiji je bio kompaktni genetski algoritam, međutim to treba uzeti s rezervom. Naime, kompaktni genetski algoritam u jednoj iteraciji stvara samo dvije jedinke i na temelju njih modificira vjerojatnosni vektor dok ostali algoritmi u jednoj iteraciji izgeneriraju više jedinki na temelju kojih ažuriraju vjerojatnosni vektor.



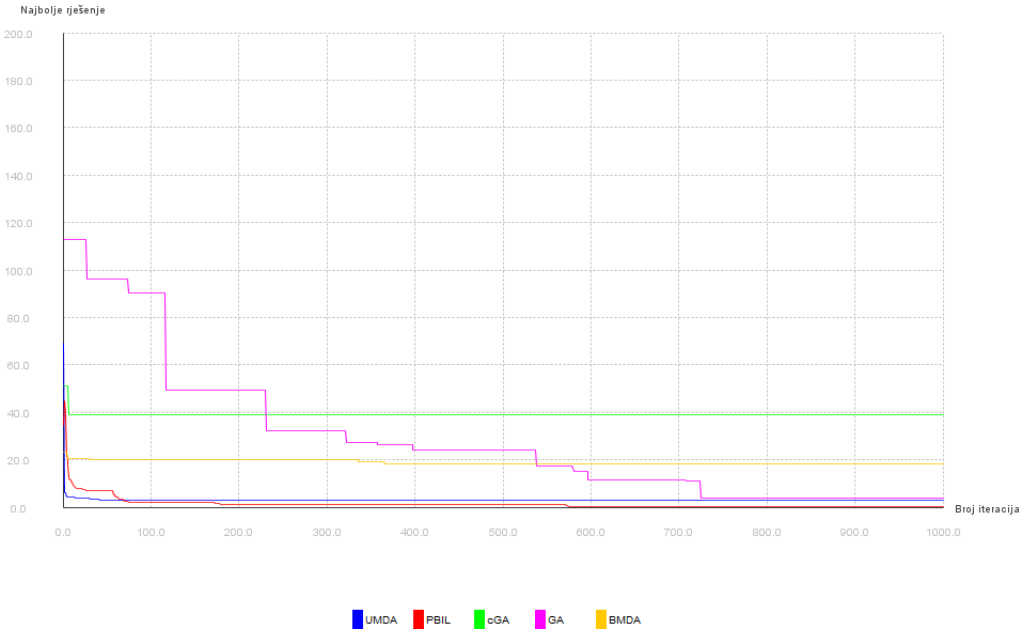
Slika 5.5: Rezultati optimizacije Rosenbrockove doline dimenzije 6



Slika 5.6: Rezultati optimizacije Rosenbrockove doline dimenzije 10

5.4.2. Rastriginova funkcija

Za razliku od Rosenbrockove doline, Rastriginova funkcija se pokazala zahtjevnom za optimizaciju. U 6-dimenzionalnom problemu samo je PBIL uspio doći do optimalnog rješenja u 1000 iteracija, dok u 10-dimenzionalnom ni jedan algoritam nije uspio. PBIL i UMDA su dali najbolje i najbrže rezultate. UMDA bi brže konvergirao, međutim u konačnici bi PBIL došao do boljih rješenja. Genetski algoritam je dao dobra rješenja, ali uz znatno većem broju iteracija. I dok su UMDA i PBIL zavrsili konvergenciju, za genetski algoritam ne možemo sa sigurnošću reći da ne bi dodatno poboljšao rješenja u narednim iteracijama. Algoritam BMDA je kao i UMDA brzo konvergirao, no zapeo je u lokalnom optimumu. Traženje zavisnosti, koje zapravo ne postoje, između dobrih rješenja je mogući razlog lošijih preformansi algoritma BMDA. Ukoliko su sve najbolje jedinke grupirane oko lokalnog optimuma, moguće je da BMDA stvara prejake pretpostavke o značajkama kvalitetnih rješenja. Najgore rezultate je ponovo ostvario kompaktni genetski algoritam. Na temelju rezultate možemo zaključiti da je algoritam zapeo u lokalnom optimumu i tu se vidi nedostatak mutacije koja bi održala raznolikost rješenja.



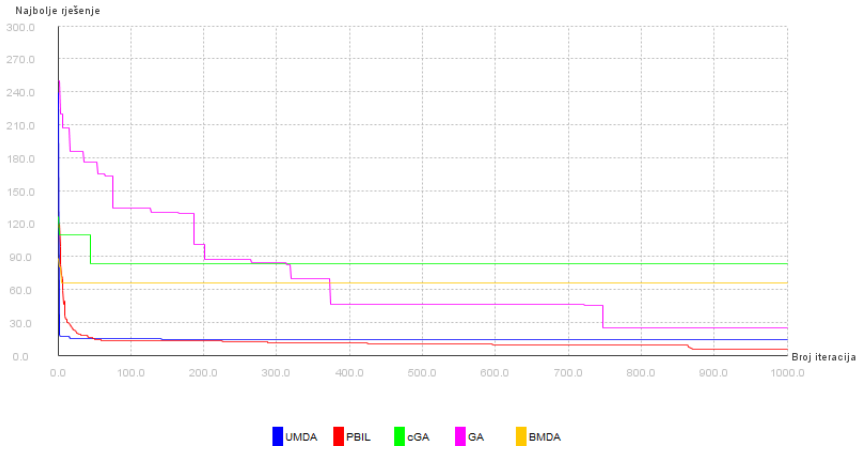
Slika 5.7: Rezultati optimizacije Rastriginove funkcije dimenzije 6

5.4.3. Schwefelova funkcija

Ispitivanje Schwefelove funkcije dalo je rezultate slične onima prilikom ispitivanja Rosenbrockove doline, iako funkcije imaju različite karakteristike. PBIL i UMDA su pronalazili globalni optimum, dok su jednostavni genetski i BMDA dolazili vrlo blizu globalnog optimuma. Ponovo je najsporije konvergirao genetski algoritam. Optimalno rješenje je najbrže pronalazio UMDA, a zatim PBIL. Kompaktni genetski algoritam je ponovno zaostajao za ostalim algoritmima. BMDA bi brzo dolazio do dobrog rješenja, međutim onda bi zapeo u njemu i ne bi uspio doći do globalnog optimuma iako se približio njemu.

5.4.4. Decepcijska funkcija trećeg reda

Decepcijska funkcija tipa 3 pokazala se najzahtjevnijom od svih ispitanih funkcija. Jedini algoritam koji je uspio doći do globalnog optimuma u oba ispitana slučaja je PBIL. Iako je sporije konvergirao od algoritama UMDA i BMDA, PBIL je uspio doći do globalnog optimuma. UMDA i BMDA su izrazito brzo konvergirali na početku optimiranja, međutim zapeli su u lokalnim optimumima. Kao i kod ostalih ispitanih funkcija, UMDA se pokazao boljim od BMDA. Kompaktni genetski algoritam i jed-



Slika 5.8: Rezultati optimizacije Rastriginove funkcije dimenzije 10

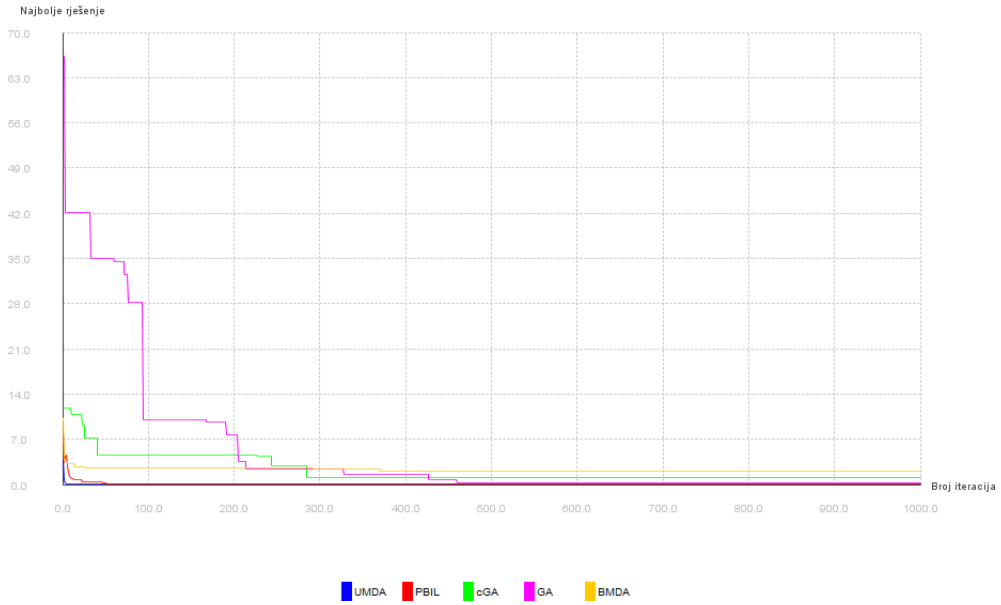
nostavni genetski algoritam pokazali su se izrazito neučinkovitim prilikom optimiranja ovog decepcijskog problema. U odnosu na ostale algoritme, i kompaktni i jednostavni genetski algoritam su sporo konvergirali i dali loša rješenja. Kompaktni genetski algoritam je nešto brže konvergirao od jednostavnog genetskog, no ni jedan algoritam nije uspio doći ni blizu globalnog optimuma. Kvaliteta dobivenih rješenja kompaktnim i jednostavnim genetskim algoritmom približno je jednaka.

5.4.5. Osvrt na rezultate u cjelini

Temeljem dobivenih rezultata na ispitnim funkcijama PBIL se istaknuo s najboljim rezultatima. Iako nije u svim ispitnim funkcijama došao do globalnog optimuma, u svim je došao do najboljeg rješenja u odnosu na ostale ispitane algoritme. Jedno zanimljivo ponašanje algoritma PBIL uočeno u evaluaciji, iako nije vidljivo iz grafova, jest implicitni elitizam. Iako se PBIL ne brine o preživljavanju najbolje jedinke, u svim iteracijama algoritma je najbolje dobiveno rješenje bilo bolje ili jednako dobro kao najbolje rješenje dobiveno u prethodnoj iteraciji.

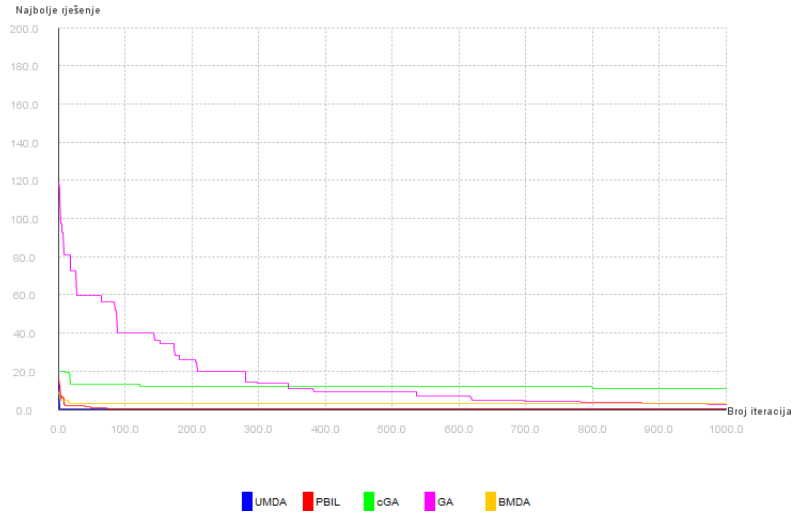
Nakon algoritma PBIL najbolje rezultate imao je UMDA. UMDA je konvergirao čak i brže od PBIL algoritma, ali je redovito pronalazio lošija rješenja. Također, verzija algoritma UMDA je imala ekspliticno dodan elitizam, preživljavanje najbolje jedinke u populaciji. Bez eksplicitnog elitizima UMDA je imao značajno lošije rezultate te je bio čest slučaj značajnog pogoršanja najboljih rješenja.

Po dobivenim rezultatima, treći najbolji algoritam je bio BMDA. Algoritam je uzimao višestruko više resursa od ostalih algoritama, kako memorijski tako i vremen-



Slika 5.9: Rezultati optimizacije Schwefelove funkcije dimenzije 6

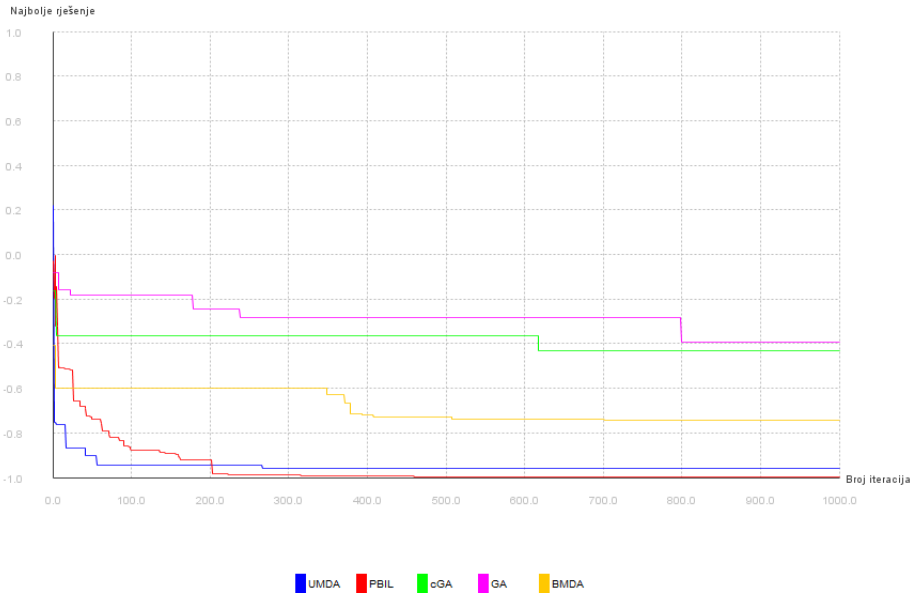
ski. S težinom problema vremenska zahtjevnost je rasla vrlo brzo te se postavlja pitanje isplativosti algoritma ukoliko nije unaprijed poznato postojanje zavisnosti između podvarijabli rješenja. Algoritam za razliku od ostalih algoritama koristi i vjerojatnosnu matricu zbog čega je memorijski izrazito zahtjevan. S obzirom da ostali algoritmi koriste vjerojatnosni vektor, izračun zauzeća memorije je jednostavan. Ukoliko se pretražuje područje veličine 10 s preciznošću od 0.01 u jednoj dimenziji, vjerojatnosni vektor zauzimat će $10/0.01 = 1000$ bitova, dok će vjerojatnosna matrica zauzimat $(10/0.01)^2 = 1000000$ bitova. Već s 10 dimenzija matrica zauzima $(10 * 10/0.01)^2 = 100000000$ bitova u usporedbi s vjerojatnosnim vektorom koji zauzima $100/0.01 = 10000$ bitova. Zbog prethodno opisane velike memorijske složenosti isplatilo bi se koristiti zapis jedinki koji zauzima manje prostora od Grayevog koda. Vremenska složenost jedne iteracije algoritma BMDA je $O(n^3)$, gdje je n broj bitova koji koristimo za prikaz rješenja, što je značajno sporije od ostalih algoritama kojima je složenost jedne iteracije $O(n)$. Algoritmu je, kao i UMDA algoritmu, eksplicitno dodan elitizam, no imao je vrlo mali utjecaj na konačne rezultate. Osim nepostojanja zavisnosti između građevnih blokova, mogući razlog lošijih rezultata BMDA je i broj rješenja na temelju kojih su se računale nove vjerojatnosti. Ukoliko je broj jedinki bio premali, moguće je da su sve jedinke unutar istog lokalnog optimuma. Tada bi BMDA stvorio zavisnosti koje ne karakteriziraju globalno, već lokalno dobro rješenje.



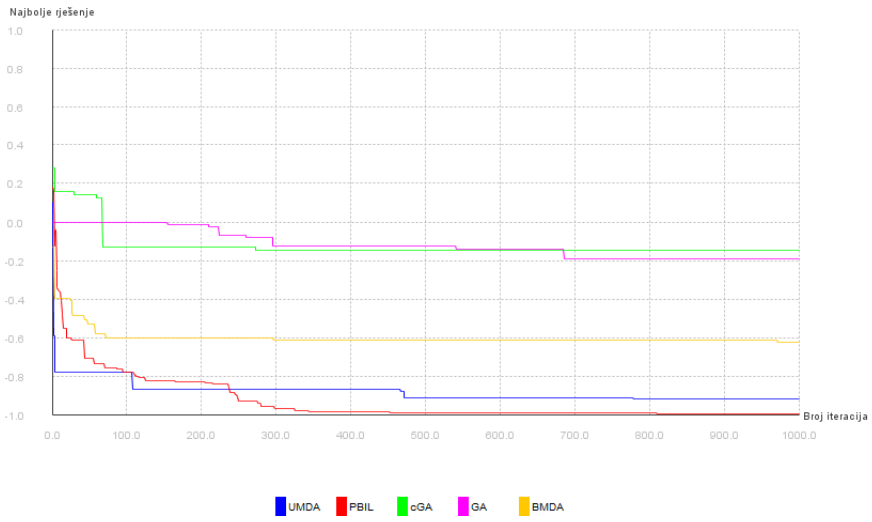
Slika 5.10: Rezultati optimizacije Schwefelove funkcije dimenzije 10

Kompaktni genetski algoritam je bio najlošiji od testiranih probabilističkih algoritama. Konvergenciju sporiju od ostalih algoritama može se objasniti činjenicom da algoritam uzorkuje samo dvije jedinke u jednoj iteraciji, dok se kod ostalih algoritama redovito radi o cijeloj populaciji. Puno veći problem kompaktnog genetskog algoritma je zapinjanje u lokalnim optimumima. Glavni razlog lošijih rezultata kompaktnog genetskog algoritma je nedostatak mehanizma koji bi očuvao gensku raznolikost u populaciji te je zbog toga neusporedivo lošiji od ostalih ispitanih algoritama.

Genetski algoritam se od probabilističkih algoritama odvaja ponajprije s značajno sporijom konvergencijom k dobrim rješenjima. Gledano ukupnu kvalitetu rješenja, osim kod decepcijske funkcije, genetski algoritam nije zaostajao puno za najboljim probabilističkim algoritmima, algoritmima PBIL i UMDA. Optimizacija decepcijske funkcije genetskim algoritmom daje izrazito loša rješenja, neusporedivo lošija rješenja od ostalih ispitanih algoritama. S obzirom da je decepcijska funkcija napravljena s ciljem da izolira rješenje i vodi genetski algoritam na krivi trag, lošiji rezultati dobiveni genetskim algoritmom nisu neočekivani. Kvalitetna rješenja dobivena algoritmima PBIL i UMDA u odnosu na genetski algoritam pokazuju da su oni značajno robusniji i otporniji na decepcijske probleme od genetskog algoritma.



Slika 5.11: Rezultati optimizacije decepcijske funkcije tipa 3 dimenzije 6



Slika 5.12: Rezultati optimizacije decepcijske funkcije tipa 3 dimenzije 10

6. Zaključak

Genetski algoritam s vjerojatnosnom razdiobom stohastička je metoda optimizacije koja problem optimizacije rješava kroz niz uzastopnih poboljšavanja vjerojatnosnog modela na temelju uzorkovanih jedinki. PMBGA su razvijeni zbog nedostataka tradicionalnih evolucijskih tehnika prilikom rješavanja problema s raspršenim građevnim blokovima.

U radu su opisani najpoznatiji PMBGA algoritmi te su programski ostvareni sljedeći algoritmi: postupno učenje zasnovano na populaciji, kompaktni genetski algoritam, algoritam univarijantne marginalne distribucije, algoritam bivarijantne marginalne distribucije i jednostavni genetski algoritam. Algoritmi su evaluirani na četiri optimizacijske funkcijama različitih svojstava. Evaluacija je pokazala da svi PMBGA algoritmi konvergiraju značajno brže od jednostavnog genetskog algoritma. Do najboljih rješenja na svim optimizacijskim funkcijama došao je algoritam postupnog učenja zasnovanog na populaciji. U usporedbi s ostalim programski ostvarenim algoritmima, jednostavni genetski algoritam nadmašio je samo kompaktni genetski algoritam dok je u odnosu na ostale algoritme redovito davao lošija rješenja.

LITERATURA

[1] S. Baluja. Population-based incremental learning. *Tech. reports*, 1994.

[2] S. Baluja i S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, DTIC Document, 1997.

[3] J.S. De Bonet, C.L. Isbell, P. Viola, et al. Mimic: Finding optima by estimating probability densities. *Advances in neural information processing systems*, stranice 424–430, 1997.

[4] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Pearson Education, 1989.

[5] M. Golub. *Genetski algoritam*, 2004. URL `http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf`.

[6] P.E. Greenwood i M.S. Nikulin. *A Guide to Chi-Squared Testing*. Wiley Series in Probability and Statistics. Applied Probability and statistIcs. Wiley, 1996. ISBN 9780471557791. URL `http://books.google.hr/books?id=bc8zfQSKOwIC`.

[7] G.R. Harik, F.G. Lobo, i D.E. Goldberg. The compact genetic algorithm. *Evolutionary Computation, IEEE Transactions on*, 3(4):287–297, 1999.

[8] J.H. Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. 1975.

[9] T. Mahnig i H. Muhlenbein. A new adaptive boltzmann selection schedule sds. U *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, svezak 1, stranice 183–190. IEEE, 2001.

[10] M. Molga i C. Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

[11] H. Mühlenbein i G. Paass. From recombination of genes to the estimation of distributions i. binary parameters. *Parallel Problem Solving from Nature—PPSN IV*, stranice 178–187, 1996.

[12] M. Pelikan i H. Mühlenbein. The bivariate marginal distribution algorithm. *Advances in Soft Computing-Engineering Design and Manufacturing*, stranice 521–535, 1999.

[13] M. Pelikan, D.E. Goldberg, i E. Cantú-Paz. Bayesian optimization algorithm, population sizing, and time to convergence. U *Proceedings of the Genetic and Evolutionary Computation Conference*, stranice 275–282, 2000.

[14] M. Pelikan, D.E. Goldberg, i F.G. Lobo. A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, 21 (1):5–20, 2002.

[15] K. Sastry i D.E. Goldberg. On extended compact genetic algorithm. U *Late-Breaking Paper at the Genetic and Evolutionary Computation Conference*, stranice 352–359, 2000.

[16] D. Thierens. Analysis and design of genetic algorithms. stranica 216, 1995.

[17] M. Čupić. *Prirodom inspirirani optimizacijski algoritmi*, 2009. URL `http://www.fer.unizg.hr/_download/repository/Cupic2009-PrirodomInspiriraniOptimizacijskiAlgoritmi.pdf`.

Genetski algoritam s vjerojatnosnim modelom

Sažetak

Genetski algoritam s vjerojatnosnim modelom podvrsta je evolucijskih algoritama koja pretražuje prostor rješenja pomoću vjerojatnosnog modela. Algoritam je razvijen zbog nedostataka konvencionalnog genetskog algoritma prilikom rješavanja problema problema s raspršenim građevnim blokovima dobrih rješenja. U ovom radu programski su ostvareni postupno učenje zasnovano na populaciji, kompaktni genetski algoritam, algoritam univarijantne marginalne distribucije i algoritam bivarijantne mariginalne distribucije te je uspoređena njihova uspješnost na nekoliko optimizacijskih problema.

Ključne riječi: Genetski algoritam s vjerojatnosnim modelom, Postupno učenje zasnovano na populaciji, Algoritam univarijantne marginalne distribucije, Algoritam bivarijantne marginalne distribucije, Kompaktni genetski algoritam

Probabilistic model-building genetic algorithm

Abstract

Probabilistic model-building genetic algorithm is subset of evolutionary algorithms which searches through solution space using probabilistic model. Algorithm was developed due to genetic algorithms poor preformace on problems with sparce building blocks. Population Based Incremental Learning, Univariate Marginal Distribution Algorithm, Compact Genetic Algorithm and Bivariate Marginal Distribution Algorithm are implemented and their performance has been compared on several optimization problems.

Keywords: probabilistic model-building genetic algorithms, Population Based Incremental Learning, Univariate Marginal Distribution Algorithm, Compact Genetic Algorithm, Bivariate Marginal Distribution Algorithm