SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1360

OPTIMIZACIJA ALGORITMOM GENETSKOG KALJENJA

Andrija Čajić

Zagreb, lipanj 2010.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1360

OPTIMIZACIJA ALGORITMOM GENETSKOG KALJENJA

Andrija Čajić

Zagreb, lipanj 2010.

ZAHVALA

Ovim putem zahvalio bih mentoru Doc.dr.sc. Domagoju Jakoboviću na vodstvu, strpljenju i entuzijazmu koje je pokazao tijekom čitavog procesa stvaranja ovog rada. Također hvala što mi je omogućio nastanak ovog rada te pomogao u njegovoj izradi.

SADRŽAJ

1. UVOD

Od pojave prvih računala pa sve do danas postoji trend sve veće digitalizacije svijeta u kojem živimo. Računala su pronašla mjesto u gotovo svim aspektima naše kulture kako bi nam život bio bolji, kvalitetniji, lakši... U svojim osnovama pomažu nam tako što dobar dio posla izvode automatizirano na mnogo točniji, brži i učinkovitiji način nego što bi to čovjek radio sam. No, uvijek je postojala granica između poslova koje ćemo povjeriti računalu i poslova za koje smo uvjereni da samo inteligentan čovjek može adekvatno riješiti. Upravo ta granica svakim se danom pomiče u korist računala zahvaljujući inovativnim algoritmima zaduženima za rješavanje novih vrsta problema. Ljudske aktivnosti kao što su organizacija poslova u vremenu, jezične analize, prepoznavanje slika i zvukova ili pak apstraktno slikarstvo ne možemo okarakterizirati kao poslove za koje bi naše osobno računalo bilo idealan kandidat. Također, postoje i brojni matematički formalizirani problemi koji do danas nisu riješeni pa su stoga poprilično interesantni brojnim programerima. Sve su to problemi koji zahtjevaju ili visoki stupanj logike, ili ljudsko životno iskustvo ili jednostavno čovjekov osjećaj na temelju kojeg donosi odluke. Upravo karakteristike koje je jako teško zapisati u oblik koji bi računalu bio razumljiv.

U ovom radu obrađuju se upravo teme vezane za takve primjene računala. Detaljnije se opisuju genetski algoritmi te konkretno primjena genetskog kaljenja na kombinatoričkom problemu trgovačkog putnika. Također, optimiraju se rješenja problema trgovačkog putnika postojećim genetskim algoritmima te analiziraju postignuti rezultati algoritama. Temeljem ovog rada pokušat će se izvući neki zaključci o genetskim algoritmima i o genetskom kaljenju koji će biti korisni ili u najmanju ruku zanimljivi za daljnje proučavanje ovog područja.

2. PROBLEM TRGOVAČKOG PUTNIKA

Problem trgovačkog putnika (eng. Travelling salesman problem, TSP) jedan je od najproučavanijih problema računalne matematike. Razlog toga možda je u jednostavnosti njegove formulacije.

Definiran je skup gradova te udaljenosti među njima. Potrebno je odrediti najkraći put kojim bi obišli sve gradove iz skupa. Put zapisujemo kao niz gradova napisanih slijedom kojim ih prolazimo.

Porijeklo ovog problema seže daleko u povijest. Nije sa sigurnošću jasno tko ga je osmislio. U priručniku iz 1832. godine problem se spominje na primjeru niza gradova u Njemačkoj i Švicarskoj ali ne tretira ga se kao matematički problem. TSP je prvi put postao predmet proučavanja matematičara tijekom 1930-ih na sveučilištima u Beču te na Harvardu. Prvi koji je matematički definirao problem je Karl Menger. On je ujedno i prvi razmatrao kompleksnost obrađivanja svih mogućih puteva te je primjetio neoptimalnost „pohlepnog" algoritma. Nedugo zatim Hassler Whitney uveo je i naziv „travelling salesman problem" (problem trgovačkog putnika). [1]

Problem se koristi kao mjerilo učinkovitosti među raznim optimizacijskim metodama. Unatoč tome što je problem računski vrlo težak, danas su poznate mnoge heurističke i egzaktne metode tako da su i primjeri s desetcima tisuća gradova rješivi. TSP čak i u svojoj najčišćoj formulaciji ima mnogobrojne primjene u planiranju, logistici te izradnji mikročipova. S manjim izmjenama možemo ga prepoznati u znanstvenim područjima sasvim stranima računarstvu i matematici, kao što je na primjer sekvenciranje DNA. U takvim primjenama gradove iz izvornog problema zamjenjuju kupci, točke lemljenja ili DNA fragmenti, a pojam udaljenosti između gradova zamjenjuju cijena puta, trajanje nekog posla ili mjera sličnosti između DNA fragmenata. Često u realnim primjenama nailazimo i na neka dodatna ograničenja kao što su nedostatna sredstva ili vremenski okviri pa takve stvari čine ovaj problem još težim.

Prema računarskoj teoriji složenosti, TSP spada u klasu NP-potpunih problema. NP-potpun problem je onaj za koji nije pronađen algoritam koji pronalazi rješenje u polinomijalnoj složenosti.

Shodno tome, opće prihvaćeno mišljenje je da ne postoji učinkovit način rješavanja problema trgovačkog putnika. Drugim riječima, svi algoritmi imaju isti problem: vrijeme rješavanja TSP-a eksponencijalno ovisi o broju gradova zadanih u problemu. Čak i TSP primjerci s nekoliko stotina gradova zahtijevaju mnogo procesorskih godina rješavanja da bi se pronašlo optimalno rješenje.

Već spomenuti „pohlepni“ algoritam opisuje traženje puta na način da krenemo iz jednog grada i uvijek odabiremo najbliži od susjednih gradova koji još nismo prošli. Takav algoritam pronalazi rješenja koja su često vrlo dobra, ali jednako često nisu optimalna.

3. EVOLUCIJSKO RAČUNANJE

U računarskoj znanosti, evolucijsko računanje spada u područje umjetne inteligencije namjenjeno optimiziranju kombinatoričkih problema. Karakteristično je po tome što se algoritmi evolucijskog računanja služe principima biološke evolucije kako bi došli do boljih rješenja ili kako bi unaprijedili svoj rad. Evolucijsko računanje koristi napredak pomoću iteracija, kao što je rast ili razvoj kroz generacije u populaciji živih bića. Ta populacija podliježe nekoj vrsti selekcije ostvarene pomoću usmjerene nasumične pretrage kako bi se u konačnici dostigao željeni cilj. Procesi selekcije često su inspirirani biološkim mehanizmima evolucije.

Upotreba Darwinovih principa evolucije u automatiziranom rješavanju problema potječe iz 1950-ih godina. 1960-ih dogodilo se to da su se tri različite interpretacije ove ideje počele razvijati na tri različite lokacije. U SAD-u, Lawrence J. Fogel uveo je pojam evolucijskog programiranja. Također u SAD-u, John Henry Holland svoju je metodu nazvao genetski algoritam. U Njemačkoj su pak Ingo Rechenberg i Hans-Paul Schwefel radili na svojim evolucijskim strategijama. Ove ideje razvijale su se oko 15 godina nezavisno jedna o drugoj. Početkom 1990-ih pojavila se i četvrta struja koja je pratila iste osnovne principe već spomenutih metoda evolucijskog računanja. Ta struja nazvana je genetsko programiranje. Također u isto vrijeme, navedeni pristupi su unificirani kao različiti predstavnici jedne tehnologije nazvane evolucijsko računanje. [2]

4. GENETSKI ALGORITMI

Genetski algoritmi spadaju u evolucijsko računanje i tvore podskup evolucijskih algoritama. Koriste se u problemima pretraživanja ili optimiziranja kako bi se pronašlo optimalno (točno) ili približno točno rješenje. Najčešće karakteristike genetskih algoritama slijede u nastavku. Glavna ideja je u tome da se rješenje optimizacijskog problema zamisli kao genetski kod neke jedinke. Ta jedinka nalazi se u populaciji zajedno s mnogim drugim jedinkama koje također predstavljaju neke od rješenja za izvorni problem. Populacija jedinki se razmnožava i tako razmjenjuje genetski materijal. Također kroz mnoge generacije pojedine jedinke mutiraju i tako stvaraju novi genetski materijal. Cijelo vrijeme nad populacijom primjenjujemo prirodna pravila i zakone za koje vjerujemo da će prouzrokovati evoluciju, a time i poboljšanje pojedinačnih jedinki u populaciji.

Genetski algoritmi implementirani su kao računalne simulacije koje prikazuju napredak neke populacije kroz mnoge generacije.

4.1. PRIKAZ JEDINKE

Želimo li genetski algoritam primjeniti na neki problem, jedno od prvih stvari što trebamo napraviti je osmisliti prikladnu apstrakciju za rješenje toga problema. Takav apstraktni prikaz rješenja nazivamo još i genotip (kromosom, genom). Genotip neke jedinke u potpunosti određuje tu jedinku, dok dvije jedinike istih genotipa predstavljaju dva identična rješenja problema. Najčešće, za genotip se odabire zapis nizom bitova no postoje mnogi drugi zapisi genotipa. Mogu se koristiti nizovi nekih drugih tipova ili struktura podataka. Glavna prednost nizova u reprezentaciji genotipova je ta što su fiksne duljine i stoga su vrlo podobni za križanja. Za zapis genotipova ponekad se koriste i stablaste strukture koje su dominantne u genetskom programiranju, te grafovi koji se koriste u evolucijskom programiranju.

4.2. PSEUDOKOD

Evolucija obično započinje od populacije slučajno odabranih ili stvorenih jedinki. Svaka generacija počinje tako što se procjenjuje dobrota svake od jedinki u populaciji. Nakon toga, stohastičkim postupcima odabire se određeni broj jedinki iz populacije te se nad njima vrše modifikacije kao što su mutacije i križanja te se ponekad novonastale jedinke uključuju u stvaranje nove populacije. Nova populacija koristi se u sljedećoj generaciji. Algoritam završava kad se ispuni jedan od unaprijed navedenih uvjeta. Uobičajeno je taj uvjet maksimalan broj generacija koliko će algoritam obraditi ili dostignuće zadovoljavajuće dobrote jedinke. Naravno, ukoliko za uvijet završavanja rada postavimo maksimalan broj generacija, to nam ne daje nikakvu garanciju u vezi dostignute dobrote cjelokupne populacije.

U konačnici svaki genetski algoritam možemo prikazati pseudokodom prikazanim na slici 4.1.

1. Odaberi početnu populaciju
2. Procjeni dobrotu svake jedinke u populaciji
3. Ponavljaj za svaku generaciju sve dok se ne ispuni uvjet završetka rada algoritma: (vremensko ograničenje, postignuta zadovoljavajuća dobrota, itd.)
    1. Odaberi najbolje jedinke za reprodukciju
    2. Stvori novu jedinku kroz križanja ili mutacije postojećih
    3. Procjeni dobrotu novodobivenih jedinki
    4. Zamijeni loše jedinke u populaciji s novim jedinkama

Slika 4.1 Pseudokod genetskih algoritama

4.3. DOBROTA JEDINKE I FUNKCIJA DOBROTE

Dobrota (eng. *fitness*) neke jedinke govori o tome koliko je pojedina jedinka „dobra“, odnosno - u kojoj mjeri jedinka ispunjava svoju svrhu. Na primjer, ako

jedinka predstavlja rješenje problema optimizacije rasporeda poslova u vremenu, primjerena dobrota jedinke bi trebala odražavati prosječnu količinu posla obavljenog u jedinici vremena. Na taj se način za jedinku koja ima veliku dobrotu može reći da ona uistinu predstavlja rješenje koje dobro raspoređuje poslove u vremenu. U tome slučaju bi kroz rad algoritma tražili jedinke sa što većom dobrotom. No to nije uvijek slučaj. Dobrotu jedinke možemo definirati tako da ona bude proporcionalna količini vremena u kojem se ništa ne radi. U tom slučaju cilj bi bio pronaći jedinku sa što manjom dobrotom. U implementaciji, dobrotu jedinke obično predstavljamo nekom brojčanom vrjednošću kako bi je lako uspoređivali s dobrotom ostalih jedinki. Ono što odražava svojstva jedinke u konkretnu brojčanu vrijednost njezine dobrote zove se funkcija dobrote (eng. *fitness function*). Funkcija dobrote uvijek ovisi o problemu koji rješavamo.

Nakon što odaberemo prikladnu reprezentaciju genotipa i funkciju dobrote za neki problem, genetski algoritam započinje svoj rad tako što inicijalizira početnu populaciju te je poboljšava kroz repetitivne operacije mutacija, križanja i selekcije. Pri inicijalizaciji, veličina populacije najčešće varira ovisno o prirodi problema kojeg rješavamo. Uobičajeno je ta brojka nekoliko stotina ili tisuća ponuđenih rješenja odnosno jedinki.

Tijekom svake od uzastopnih generacija uzima se udio jedinki iz populacije kako bi one tvorile novu generaciju. Odabir jedinki koje će tvoriti novu generaciju ovisi o njihovoj dobroti. Što je jedinka „bolja“, imat će veću vjerojatnost da bude odabrana. Ovdje se možda najbolje vidi porijeklo samog naziva evolucijskog računanja.

> *“This survival of the fittest, which I have here sought to express in mechanical terms, is that which Mr. Darwin has called 'natural selection', or the preservation of favoured races in the struggle for life."* je rečenica poznatog engleskog filozofa Herberta Spencera i to je ujedno prvo pojavljivanje veoma poznate fraze *„survival of the fittest“*, u prijevodu „opstanak najjačih“. Zanimljivo je napomenuti da je ova rečenica izvučena iz konektsta u kojem englez povlači paralele između Darwinove teorije evolucije i teorija tadašnje ekonomije. Iz toga se može zaključiti da su već

tada u 19. stoljeću primjećene moguće primjene ovakvog načina razmišljanja na rješavanje konkretnih problema.

Vrijednost dobrote jedinke određuje funkcija dobrote karakteristična specifičnom problemu koji rješavamo. Neki algoritmi su napravljeni tako da se određuje dobrota svake jedinke u populaciji. Ponekad takav pristup može biti neprimjeren zbog veličine populacije i/ili složenosti funkcije dobrote. Zato neki algoritmi rade tako da uzmu slučajno odabrani uzorak populacije te samo njih evaluiraju. Također korisno je pobrinuti se da funkcija dobrote ne bude vremenski ili prostorno složena operacija budući da je to funkcija koja se poziva vjerojatno najviše puta tijekom cijelog izvođenja genetskog algoritma.

Većina operatora selekcije su stohastičke prirode. To znači da boljim jedinkama osiguravaju jedino veću vjerojatnost preživljavanja, ali to nipošto ne znači da će samo dobre jedinke biti odabrane, a sve lošije biti odbačene. Stohastički operatori selekcije osiguravaju da i mali udio loših jedinki bude odabrano u novu generaciju. Takav način selekcije povećava raznolikost genotipova u populaciji. To je vrlo bitno kako bi se izbjegle prerane konvergencije prema naizgled optimalnim rješenjima koja zapravo to nisu. Neke od popularnih operatora selekcije su takozvana „*roulette wheel*“ selekcija i turnirska selekcija kojima će biti riječi kasnije.

4.4. GENETSKI OPERATORI

Pri generiranju nove generacije, potrebno je nadomjestiti jedinke koje nisu prošle posljednju selekciju. One se nadomještaju novim jedinkama koje se stvaraju od onih koje su prošle selekciju. Ovdje se služimo operatorima mutacije i križanja.

Mutacija je unarni operator što znači da se primjenjuje na jednu jedinku. Nakon mutacije jedinka je izmjenjena tako što su se neki segmenti njezinog genotipa izmjenili. Odabir segmenata genotipa za promjenu te način i količina promjene koja će se dogoditi na samom genotipu najčešće je u potpunosti

slučajan. To znači da mutacija može utjecati na jedinku tako što je učini boljom ili lošijom. Štoviše, puno je veća vjerojatnost da će mutacija imati negativan učinak nego pozitivan. No, unatoč tome, mutacija je bitan čimbenik genetskih algoritama jer na vrlo jednostavan način unosi genetsku raznolikost u populaciju.

Križanje je binarni operator. Primjenjuje se na dvije roditeljske jedinke koje tvore novu jedinku. Nova jedinka poprima karakteristike obaju roditeljskih jedinki. Nasljeđivanje karakteristike roditelja ostvaruje se tako što se neki segmenti genotipa uzmu od prvog roditelja, a ostali segmenti od drugog roditelja te se na taj način kompletira nova jedinka. Naravno, odabir segmenata za križanje najčešće je slučajan. Iako je ovakav način reprodukcije populacije dosta zgodan jer je inspririran prirodom, istraživanja pokazuju da je za neke probleme prikladno definirati križanje kao razmjenu gena više od dvaju roditelja. To znači da bi jedinka nastala križanjem imala 3, 4 ili više jedinki „roditelja“. Proces križanja u konačnici stvara populaciju koja je različita od početne. Najčešće križanje dovodi do povećanja prosječne dobrote u populaciji jer se za roditeljske jedinke odabiru najbolje jedinke iz populacije.

Genetski algoritmi primjenjuju se na području računarske znanosti, bioinformatike, inženjerstva, ekonomije, kemije, fizike, matematike i još mnogih drugih. [3]

5. SIMULIRANO KALJENJE

Simulirano kaljenje (eng. *simulated annealing, SA*) je metahuristika za problem globalne optimizacije u primjenjenoj matematici, najčešće za pronalazak aproksimacije globalnog optimuma neke funkcije u velikom prostoru pretraživanja. Koristi se uobičajeno u problemima gdje je prostor pretraživanja diskretan. Inspiracija za ime dolazi od kaljenja u metalurgiji. Tehnika je to koja koristi zagrijavanje te kontrolirano hlađenje materijala kako bi se povećali njegovi kristali i smanjili njihovi nedostatci. Toplina utječe na atome tako što ih oslobađa od njihovog početnog položaja i dopušta im da se slobodno kreću na višim energetskim razinama. Polagano hlađenje daje atomima veću vjerojatnost da pronađu razmještaj u kojem će imati manju unutarnju energiju nego prije procesa kaljenja. Manja unutarnja energija atoma znači veću stabilnost spoja i bolje karakteristike.

Za analogiju rada algoritma s fizikalnim zakonima trebamo potrebno je staviti stvari u ispravan kontekst. Svaki korak rada simuliranog kaljenja zamjenjuje trenutno rješenje za nekim, njemu „bliskim“, slučajno odabranim rješenjem. Vjerojatnost odabira zamjenskog rješenja ovisi o tome kolika je razlika u energetskoj razini (prikladnosti, dobroti) rješenja kandidata i trenutačnog rješenja. Također ta vjerojatnost ovisi i o globalnom parametru T, koji označava temperaturu i postepeno se smanjuje tijekom rada algoritma kroz višestruke iteracije. Ovisnost je takva da se trenutačno rješenje zamjenjuje gotovo uvijek kada je T velik, neovisno je li zamjena pozitivno ili negativno utjecala na kvalitetu rješenja. Kada je T malen i približava se nuli, tada se pri zamjenama sve više inzistira da zamjensko rješenje bude uvijek bolje od trenutačnog rješenja. Početno dopuštanje uzlaznog trenda dobrote rješenja spriječava zaglavljivanje rješenja u lokalnom minimumu što je glavni problem kod pohlepnih algoritama.

```
Pseudokod:
s ← s0; e ← E(s)                    // Počento stanje, početna eergija.
sbest ← s; ebest ← e                // Početno "najbolje" rješenje
k ← 0                               // brojač iteracija.
while k < kmax and e > emax         // dok nismo prošli sve iteracije & nismo pronašli dovoljno dobro rješenje:
snew ← neighbour(s)                 // Odaberi slično (susjedno) rješenje.
 enew ← E(snew)                     // Izračunaj njegovu energiju.
 if enew < ebest then               // Je li to nova najbolja energija?
       sbest ← snew; ebest ← enew   // Spremi novu jedinku kao najbolju pronađenu.
 if P(e, enew, temp(k/kmax)) > random() then
                                    // Je li jedinka dovoljno dobra?
  s ← snew; e ← enew                // Da, promijeni stanje.
 k ← k + 1                          // Povećaj brojač iteracija
return sbest                        // Vrati najbolje pronađeno rješenje.
```

Slika 5.1 Pseudokod simuliranog kaljenja

Postupno hlađenje simuliramo funkcijom P koja vraća veću vrijednost što je k/kmax manji. Također vraća veću vrijednost ako je (e – enew) veći. Uspoređujemo je s nekom slučajnom vrijednosti što uvodi mjeru slučajnosti u cijeli postupak i tako prepušta konkretne odabire vjerojatnostima.

Postoje mnoge varijacije algoritama nastale na temelju simuliranog kaljenja. Neke od njih su stohastičko tuneliranje, prilagodljivo simulirano kaljenje i kvantno simulirano kaljenje. [4]

6. GENETSKO KALJENJE

Genetsko kaljenje (*Genetic Annealing*, GA) je algoritam koji je nastao kao hibrid između simuliranog kaljenja i genetskih algoritama. Tvorac koncepta genetskog kaljenja je Kenneth Price koji je 1994. u svome članku u popularnom programerskom časopisu „Dr. Dobb's Journal“ (DDJ) prvi put iznio ideju o spajanju simuliranog kaljenja i genetskih algoritama. GA radi na istom principu kao i simulirano kaljenje samo što algoritam provodi na populaciji potencijalnih rješenja umjesto na samo jednom te uvodi pojam razmjene energije između jedinki. Također provode se operacije mutacija i srastanja radi stvaranja novog i razmjene postojećeg genetskog materijala. Srastanje (eng. *splicing*) je proces sličan križanju koji se najčešće provodi nad genotipovima niza bitova. Ovim postupkom zamjenjujemo dio niza bitova iz genotipa s odgovarajućim nizom bitova iz nekog drugog, slučajno odabranog, genotipa iz populacije. Zatim procjenjujemo je li novodobivena jedinka dovoljno dobra da zamijeni jedinku nad kojom se vrši srastanje. Slučajno odabrani genotip koji šalje kopiju segmenta svog genetskog materijala kao probnu mutaciju ostaje nepromjenjen.

Umjesto korištenja natjecateljskih turnira ili sličnih metoda selekcije koji se baziraju na međusobnom uspoređivanju jedinki, genetsko kaljenje koristi prilagodljivi, termodinamički kriterij preuzet od simuliranog kaljenja. Umjesto globalnog parametra T koji je označavao temperaturu, genetsko kaljenje uvodi tzv. „*energy bank*“, parametar koji označava ukupnu slobodnu energiju koju mogu koristiti sve jedinke kako bi skočile na višu energetsku razinu i tako izbjegle lokalne minimume. Jedinke koje skoče na nižu energetsku razinu i tako snize svoju unutarnju energiju, višak svoje energije otpuštaju u spremnik slobodne energije (*energy bank*) koji zatim koriste sve ostale jedinke. Hlađenje u genetskom kaljenju simulira se na puno prirodniji način nego u simuliranom kaljenju. Postoji globalni parametar – faktor hlađenja (eng. *cooling factor*). U svakoj iteraciji spremnik slobodne energije smanji se tako da se pomnoži s faktorom hlađenja. Faktor hlađenja je konstanta određena kao parametar samog algoritma te poprima

realne vrijednosti iz intervala [0, 1]. Ako ta vrijednost teži k nuli, hlađenje će biti brzo, a ako pak teži jedinici, hlađenje će se odvijati sporo. Kako se to reflektira na rad algoritma? Zgodno je primjetiti da ako odaberemo nulu za početnu vrijednost spremnika slobodne energije i nulu za faktor hlađenja, naš algoritam genetskog kaljenja bit će zapravo obični pohlepni algoritam koji se primjenjuje paralelno na sve jedinke u populaciji. Doduše, pohlepnost ovdje nema identično značenje kao za već navedeni pohlepni algoritam u rješavanju problema trgovačkog putnika. Sličnost im je u tome što uvijek gledaju kratkoročnu dobit i na taj način funkcioniraju cijelim postupkom. Ako pak za faktor hlađenja odaberemo jedinicu, prosječna dobrota jedinki u populaciji nakon proizvoljnog broja iteracija ostat će više-manje ista kao i početku rada algoritma.

```
Pseudokod:
1. Nasumično odaberi početnu populaciju od N čestica.
2. brojač iteracija = 0
3. dok uvjet zaustavljanja nije zadovoljen:
        4. prag = slobodna energija / N
        5. slobodna energija = 0
        6. za svaku česticu:
                7. mutant = mutiraj(čestica)
                8. ako energija(mutant) < energija(čestica) + prag:
                        9. razlika = energija(čestica) + prag – energija(mutant)
                        10. slobodna energija = slobodna energija + razlika
                        11. čestica = mutant
                12. inače slobodna energija = slobodna energija + prag
        13. brojač iteracija = brojač iteracija + 1
        14. slobodna energija = slobodna energija * C
15. rješenje = najbolja(čestica)
```

Slika 6.1 Pseudokod genetskog kaljenja

Na temelju genetskog kaljenja nastala je i diferencijska evolucija (*Differential evolution*, DE). To je metoda koja spada u skupinu evolucijskih strategija. DE je nastala iz Price-ovih pokušaja rješavanja aproksimacije funkcija Čebiševljevim polinomima. Problem je riješen modifikacijom genetskog kaljenja tako da se koristi broj s pomičnim zarezom umjesto niza bitova za reprezentaciju genotipa i vektor aritmetičkih operacija umjesto vektora logičkih operacija. DE se koristi za optimizaciju funkcija više varijabli i ima veliku vjerojatnost pronalaska globalnog optimuma. Najvažniji dio DE-a je shema za generiranje probnih parametarskih vektora [3]. Shema je samoorganizirajuća pa nisu potrebne vjerojatnosne distribucije koje koriste evolucijske strategije. Diferencijska evolucija se od drugih evolucijskih strategija najviše razlikuje u fazi mutacije. [5]

7. PRIMJENA GENETSKOG KALJENJA NA PROBLEM TRGOVAČKOG PUTNIKA

Kako bismo primjenili algoritam genetskog kaljenja na problem trgovačkog putnika (TSP), moramo definirati sve bitne elemente algoritma. Algoritam genetskog kaljenja izvodit će se u radnom okruženju ECF (*Evolutionary Computation Framework*). ECF je razvojni alat za primjenu proizvoljnog tipa evolucijskog računanja na proizvoljan problem. ECF podržava odabir vlastitog genotipa za prikaz rješenja, odabir parametara algoritama, paralelno izvođenje algoritma, varijabilnu veličinu populacije, odabir uvjeta završavanja rada algoritma, te vjerojatnosti križanja i mutacije. [6]

Prikaz rješenja odnosno genotip jedinke mora jednoznačno opisivati svako moguće rješenje problema. Ako imamo niz od 6 gradova {A, B, C, D, E, F} vidimo da bilo koja permutacija ovih 6 slova jednoznačno označava put između svih 6 gradova, a samim time predstavlja i idealan prikaz rješenja. U sklopu ECF-a ponuđen je genotip permutacijskog vektora koji odgovara problemu trgovačkog putnika ako svakom gradu pridružimo jedan broj te se nadalje koristimo vektorom cijelih brojeva. Mutacije i križanja nad permutacijskim vektorom definirani su na više načina. U sklopu ECF radnog okruženja postoje tri vrste križanja i tri vrste mutacija. Od križanja ponuđeni su „partial-mapped crossover“ (PMX), „order crossover“ (OX) i „position based crossover“ (PBX). Za operator mutacije moguće je odabrati između obrtanja poretka (eng. *inversion mutation*), ubacivanja (eng. *insertion mutation*) i nasumične zamjene elemenata (eng. *mutation toggle*). [7]

Podaci, gradovi i njihove međusobne udaljenosti, zadani su dvodimenzionalnom tablicom u kojoj svaka ćelija pokazuje kolika je udaljenost između pripadna dva grada u km kao što je prikazano u tablici 1. Konkretan problem koji će biti predmet optimizacije za algoritam genetskog kaljenja bit će zadan s 29 njemačkih gradova. Problem je preuzet sa [8].

Taj problem sadrži 8,84 * 10^30 mogućih permutacija vektora odnosno 4,42 * 10^30 potencijalnih ruta u obilasku gradova ako istu rutu prijeđenu suprotnim smjerom računamo kao jednu rutu. Najkraći mogući put kroz svih 29 gradova je poznat i on iznosi 2020 kilometara. Ovaj podatak služi kao mjerilo kvalitete ostalih rješenja kojima je cilj dostići upravo ovu vrijednost.

tablica 7.1. Udaljenosti između gradova

	Zagreb	Split	Rijeka
Zagreb	0	409	161
Split	409	0	414
Rijeka	161	414	0

7.1. ISPITIVANJE PARAMETARA

Radi bolje prilagodbe parametara algoritma problemu trgovačkog putnika, genetsko kaljenje je primjenjeno više puta s različitim parametrima. Kako bi usporedba različitih konfiguracija algoritma bila što vjernija, uvjet završavanja je postavljen na maksimalno trajanje od 15 sekundi.

7.1.1. ISPITIVANJE 1

Prvo ispitivanje, kao i sva ostala ispitivanja provedeno je sa populacijom veličine 50. Uvijet završavanja je vremensko ograničenje od 15 sekundi.

tablica 7.2 Problem: TSP (29 gradova); Algoritam: GeneticAnnealing

Energy Bank	Cooling Factor	Broj izvođenja	Najbolja dobrota (prosjek)	Najjbolja dobrota (minimum)	Najbolja dobrota (maksimum)	Generacija najboljeg rješenja	Energy bank u posljednoj generaciji	Energetski prag (dE)
10000	0,97	10	2418	2367	2501	1059,3	1432,18	23,54
200	0,97	10	2449	2294	2630	963,1	1529,32	29,67

U dva navedena primjera (tablica 7.2) razlikuje se samo parametar inicijalne vrijednosti spremnika slobodne energije odnosno „energy bank“. Ono što prvo možemo zamjetiti je (ne)uspješnost algoritma prikazana dobrotom najbolje pronađene jedinke. Ako se uzme u obzir da je globalni optimum problema dobrota od 2020, onda je jasno da su dobiveni rezultati relativno loši. Slabe rezultate možemo objasniti velikim koeficijentom hlađenja (0,97) koji ne stigne u dovoljnoj mjeri ohladiti čestice tijekom 15 sekundi predodređenog vremena za rad algoritma. U sljedećem ispitivanju snižavanjem faktora hlađenja pokušava se postići brža konvergencija boljim rješenjima. Rezultati i novi skupovi parametara prikazani su u tablici 7.3.

7.1.2. ISPITIVANJE 2

tablica 7.3 Problem: TSP (29 gradova); Algoritam: GeneticAnnealing

Energy Bank	Cooling Factor	broj izvođenja	Najbolja dobrota (prosjek)	Najbolja dobrota (minimum)	Najbolja dobrota (maksimum)	Generacija najboljeg rješenja	Energy bank u posljednoj generaciji	Energetski prag (dE)
25000	0,97	10	2430,1	2222	2565	1002,3	1494,52	28,99
25000	0,9	10	2257,4	2115	2317	1054,1	120,23	2,16

U nova dva primjera ispitan je utjecaj faktora hlađenja na rad algoritma. Iz podataka u tablici 7.3 može se očitati drastično bolji učinak algoritma s manjim faktorom hlađenja. Također slobodna energija u algoritmu s nižim faktorom hlađenja ukazuje na to da je taj algoritam više inzistirao na uniformnom napretku nego algoritam sa nižim faktorom hlađenja. Prosjek energetskog praga pri završetku drugog izvođenja u tablici 7.3 iznosi 2,16. To znači da bi pri nastavku rada algoritam od mutiranih jedinki prihvaćao samo one koje su bolje od originala ili za 2,16 lošije. Ako se postavi u kontekst priče s problemom trgovačkog putnika, algoritam je mutacijom spreman kompromitirati duljinu svoje trenutne rute za 2,16

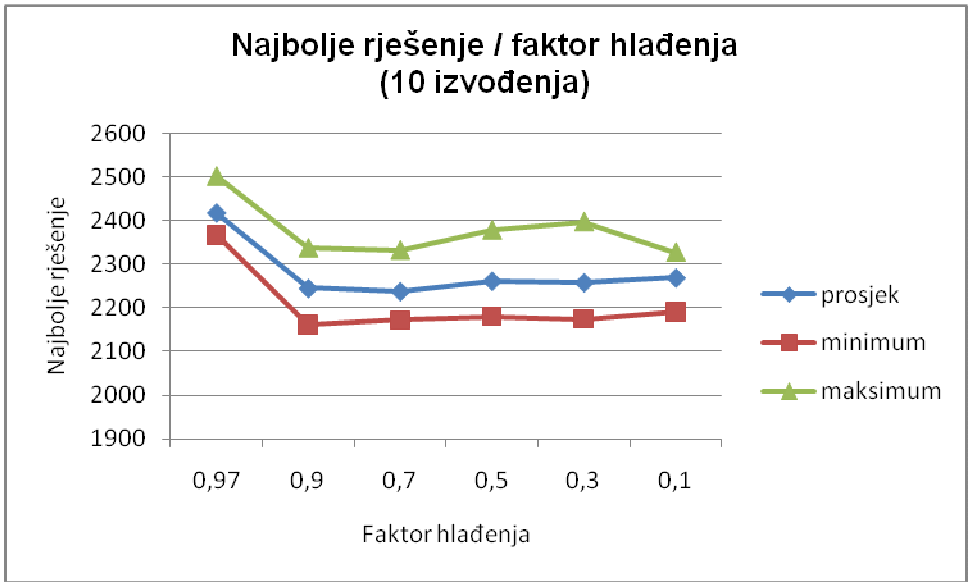
km. S obzirom da su u našem konkretnom problemu udaljenosti između gradova u okvirima 100, 200 ili 300 km, vidimo da pri kraju rada algoritam postaje poprilično „pohlepan“.

7.1.3. ISPITIVANJE 3

U nastavku je detaljni prikaz ovisnosti rada algoritma o parametrima faktora hlađenja (tablica 7.4 i slika 7.1).

tablica 7.4 Problem: TSP (29 gradova); Algoritam: GeneticAnnealing

Pokretanje / faktor hlađenja	0,97	0,9	0,7	0,5	0,3	0,1
1	2438	2185	2264	2212	2335	2191
2	2390	2295	2216	2205	2219	2302
3	2423	2267	2180	2180	2238	2298
4	2389	2291	2332	2269	2305	2237
5	2367	2309	2172	2221	2265	2279
6	2395	2337	2173	2257	2240	2232
7	2476	2177	2273	2378	2397	2252
8	2501	2209	2235	2354	2196	2327
9	2403	2161	2284	2266	2214	2265
10	2397	2221	2252	2278	2174	2309
min	2367	2161	2172	2180	2174	2191
max	2501	2337	2332	2378	2397	2327
avg	2417,9	2245,2	2238,1	2262	2258,3	2269,2



Slika 7.1 Grafički prikaz ovisnosti rada genetskog kaljenja o faktoru hlađenja (cooling factor)

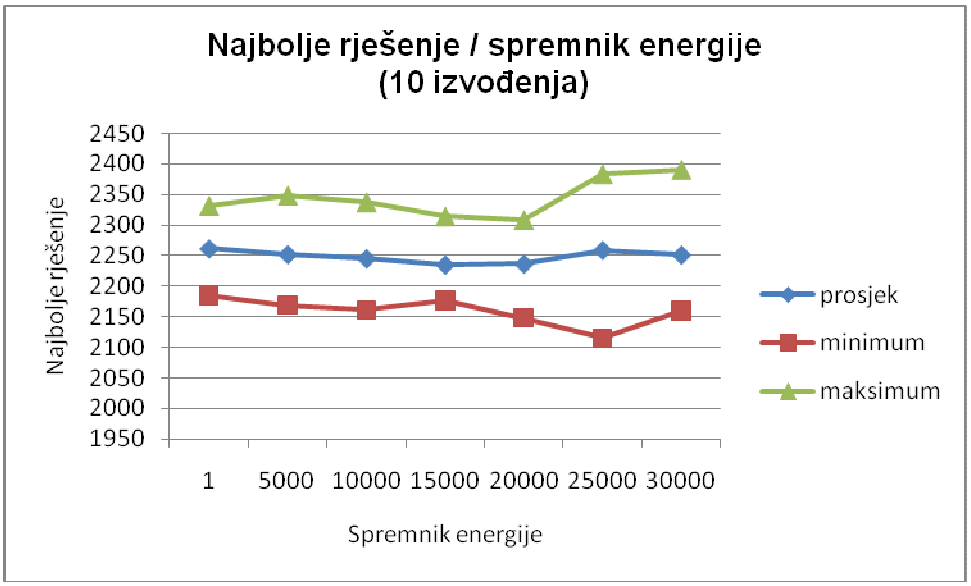
Iz grafičkog prikaza (slika 7.1) vidi se da algoritam postiže slabe rezultate samo uz jako visoki faktor hlađenja (~1). Osim toga, faktor hlađenja nema prevelikog utjecaja na uspješnost algoritma. Najbolje rješenje pronađeno je pri faktoru hlađenja od 0,9, a u prosjeku najbolja rješenja dobivena su faktorom hlađenja od 0,7.

7.1.4. ISPITIVANJE 4

Također, upotpunjen je i prikaz ovisnosti rada algoritma genetkog kaljenja o parametru spremnika energije (tablica 7.5 i slika 7.2).

tablica 7.5 Problem: TSP (29 gradova); Algoritam: GeneticAnnealing

Pokretanje / energy bank	1	5000	10000	15000	20000	25000	30000
1	2185	2169	2185	2237	2148	2211	2160
2	2266	2179	2295	2266	2165	2115	2216
3	2229	2207	2267	2176	2296	2317	2390
4	2313	2348	2291	2179	2179	2274	2284
5	2247	2193	2309	2201	2308	2213	2261
6	2293	2247	2337	2259	2276	2348	2196
7	2331	2322	2177	2290	2275	2383	2211
8	2294	2325	2209	2314	2210	2219	2225
9	2193	2279	2161	2252	2291	2212	2277
10	2264	2252	2221	2177	2217	2299	2294
min	2185	2169	2161	2176	2148	2115	2160
max	2331	2348	2337	2314	2308	2383	2390
avg	2261,5	2252,1	2245,2	2235,1	2236,5	2259,1	2251,4



Slika 7.2 Grafički prikaz ovisnosti rada genetskog kaljenja o spremniku energije (energy bank)

Na slici 7.2 primjetno je lagano poboljšanje izvedbe kada parametar spremnika energije postavimo na vrijednost od ~25000.

7.2. USPOREDBA ALGORITAMA

Sljedeće, i možda najzanimljivije, je napraviti procjenu rada genetskog kaljenja u usporedbi s nekim drugim evolucijskim algoritmima. Za usporedbu su odabrana dva evolucijska algoritma u sklopu ECF-a – „*Roulette wheel*“ i „*Steady state tournament*“. Oba algoritma u svojoj esenciji razlikuju se po načinu selekcije jedinki koje će se propagirati u buduće generacije.

„*Roulette wheel*“ je zapravo alternativni naziv selekcije koja odabire jedinke s vjerojatnostima koje su proporcionalne njihovoj dobroti. Još jedan način prikaza ovakvog načina selekcije je da se za svaku jedinku odredi njezina dobrota te se ta vrijednost pribroji na brojevni pravac označujući da taj segment brojevnog pravca pripada upravo toj jedinki. Nakon što smo pribrojili vrijednosti za sve jedinke odabiremo slučajan broj s brojevnog pravca počevši od nule do maksimalne

vrijednosti koju smo dosegli pribrajanjem dobrota svih jedinki. Odabravši broj, provjerimo čijem segmentu brojevnog pravca pripada te pripadnu jedinku propagiramo u novu generaciju. Uz ovakav odabir može se nerijetko dogoditi da najbolja jedinka bude izostavljena iz budućih generacija pogotovo ako su dobrote jediniki podjednakih vrijednosti. Stoga se često prakticira prvo odabir većeg skupa najboljih jedinki iz populacije a zatim nastavak s odabirom proporcionalnim dobroti. Ovakav prikaz pomalo podsjeća na rulet, pa od toga dolazi i sam naziv.

„*Steady state tournament*" je način selekcije u kojem se iz populacije nasumično odabire n jedinki te ih se zatim stavlja u turnirsko međusobno natjecanje. Iz turnirskog natjecanja *k* najlošijih jedinki ispadaju te one jedinke koje su ostale budu propagirane da tvore buduću generaciju. Turnirsko natjecanje nije ništa drugo nego sortiranje jedinki prema njihovoj dobroti. U ovom primjeru koristi se turnir od 3 jedinke iz kojeg će se birati 1 najlošija jedinka za eliminaciju.

7.2.1. ISPITIVANJE 1

tablica 7.6 Problem: TSP (29 gradova); Algoritam: GeneticAnnealing

Energy Bank	Cooling Factor	Broj izvođenja	Najboja dobrota (prosjek)	Najboja dobrota (minimum)	Najboja dobrota (maksimum)	Generacija Pronađenog Optimuma	Energy bank u posljednoj generaciji	Energetski prag (dE)
10000	0,7	10	2238,1	2172	2332	913	41,42	0,58

U tablici 7.6 prikazani su rezultati izvođenja genetskog kaljenja s početnom slobodnom energijom od 10 000 te faktorom hlađenja 0,7. Može se primjetiti nešto bolji uspjeh nego u tablici 7.2 gdje je početna slobodna energija bila 25 000 i faktor hlađenja nešto viši. Taj podatak je objašnjiv tako što algoritam dopušta manje „lutanja" jer odmah u startu postavi energetski prag na neku razumnu razinu koja neće dopustiti da se rješenje mutacijom pretvori u nešto sasvim neupotrebljivo, što se tijekom ograničenog vremenskog perioda (15 sekundi) neće stići pretvoriti u neko korisno rješenje. S druge strane ako pogledamo tablicu 7.2 vidljivo je drugo mjerenje unatoč tome što ima manju slobodnu energiju na početku (*energy bank* =

200) postiže nešto slabije rezultate od algoritma iz prvog mjerenja (*energy bank* = 10 000).

7.2.2. ISPITIVANJE 2 I 3

Tablica 7.7 Problem: TSP (29 gradova); Algoritam: RouletteWheel

Crxprob	Selpressure	Broj izvođenja	Najbolja dobrota (prosjek)	Najbolja dobrota (minimum)	Najbolja dobrota (maksimum)	Generacija Pronađenog Optimuma
0,5	10	10	2487,6	2079	2800	722

Tablica 7.8 Problem: TSP (29 gradova); Algoritam: SteadyStateTournament

Tournament Size	Broj izvođenja	Najbolja dobrota (prosjek)	Najbolja dobrota (minimum)	Najbolja dobrota (maksimum)	Generacija Pronađenog Optimuma
3	10	2155,8	2051	2364	578,1

Iz tablice 7.8 vidimo da od ponuđenih algoritama turnirska selekcija daje najbolje rezultate. Turnirska selekcija ima svojstvo elitizma, što znači da najbolja jedinka nikada neće biti izbačena iz populacije. Tijekom izvođenja genetskog kaljenja čest je slučaj da algoritam pronađe relativno dobru jedinku u sredini svog izvođenja te je zatim zbog još prevelike razine slobodne energije zamjeni poprilično lošijim rješenjem. Stoga se pretpostavlja da je upravo svojstvo elitizma omogućilo ovako dobre rezultate turnirske selekcije. Algoritam genetskog kaljenja moguće je doraditi tako da najbolja jedinka u svakoj iteraciji ima energetski prag jednak nuli. Na taj način bi, primjerice, elitizam mogao biti ugrađen i u genetsko kaljenje. Također, pokazalo se da genetsko kaljenje ne zaostaje puno za turnirskom selekcijom, a u odnosu na „*roulette wheel*“ algoritam često daje i mnogo bolje rezultate. Algoritam proporcionalne selekcije („roulette wheel“) pokazao je veliku disperziju u rezultatima. Od 10 izvođenja najuspješnije izvođenje postiglo je dobrotu od 2079, a najlošije 2800. Jedna stvar koju je bitno napomenuti je podatak generacije pronađenog najboljeg rješenja. Genetsko kaljenje obično obradi više generacija u istom vremenskom intervalu u odnosu na ostala dva algoritma. To je pokazatelj da među ova tri algoritma, genetsko kaljenje najbrže obrađuje jedinke te stvara nove generacije.

8. ZAKLJUČAK

Među evolucijskim algoritmima, genetsko kaljenje ima relativno jednostavan princip rada, no upravo ta jednostavnost omogućuje mu brži rad, veći broj iteracija u jednakoj količini vremena i ponekad bolje rezultate. Problem trgovačkog putnika ispostavio se kao dobar primjer primjene genetskog kaljenja. Algoritam pokazuje bolje rezultate ako je podešen tako da dopušta manje negativnih mutacija, odnosno da više sliči pohlepnom algoritmu. Jednostavnost genetskog kaljenja otvara prostor razmišljanju u smjeru poboljšanja ili specijalizacije algoritma. Postoje brojni algoritmi nastali iz simuliranog kaljenja (kvantno simulirano kaljenje, prilagodljivo simulirano kaljenje) koji bi se mogli modificirati tako da se izvršavaju u skupini (populaciji) rješenja te da rješenja izmjenjuju informacije. U tom slučaju imali bi algoritme poput kvantnog genetskog kaljenja ili prilagodljivog genetskog kaljenja što su zasad tek ideje za razmatranje.

Postavlja se pitanje kada je prikladno koristiti genetsko kaljenje? Koji su problemi lakše rješivi genetskim kaljenjem nego ostalim algoritmima? Iako se oko ovih pitanja uvijek može raspravljati, konačan odgovor leži u eksperimentiranju i metodologiji pokušaja i pogrešaka.

LITERATURA

[1] Travelling salesman problem, June 19th 2010,
http://en.wikipedia.org/wiki/Travelling_salesman_problem, 25. 06. 2010.

[2] Evolutionary computation
http://en.wikipedia.org/wiki/Evolutionary_computation 28. 05. 2010.

[3] Genetic algorithm
http://en.wikipedia.org/wiki/Genetic_algorithm, 28.05. 2010.

[4] Simulated annealing
http://en.wikipedia.org/wiki/Simulated_annealing, 29. 05. 2010.

[5] Dr. Dobb's Algorithm Alley, October 1st 1994,
http://www.ddj.com/architect/184409333?pgno=10, 29. 05. 2010.

[6] ECF - Evolutionary Computation Framework, 11. 05. 2010.,
http://gp.zemris.fer.hr/ecf/, 25. 06. 2010.

[7] Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem, October 17th 2007,
http://dx.doi.org/10.1016/j.amc.2007.10.013, 26. 06. 2010.

[8] The TSPLIB Symmetric Traveling Salesman Problem Instances, June 1st 1995,
http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/, 25. 06. 2010.

OPTIMIZACIJA ALGORITMOM GENETSKOG KALJENJA

SAŽETAK

U velikoj mjeri ovaj rad pokriva teoretsku pozadinu evolucijskih algoritama kao i još nekih prirodom inspiriranih algoritama. Zamišljen je tako da približi čitatelju ideju koja je iza svakog pokušaja da se matematički ili kombinatorički problem riješi prirodnim zakonima i načinima. Glavni je cilj, doduše, bio primjena genetskog kaljenja na optimizaciju problema trgovačkog putnika. U ovoj praktičnoj primjeni genetskog kaljenja, namjera je bila iznijeti karakteristike algoritma kako bi se naglasile njegove mogućnosti i nedostatci, kvalitete i mane. Višestrukim pokretanjem algoritma s različitim parametrima te analizom rezultata pokušalo se obuhvatiti esenciju algoritma genetskog kaljenja. Na kraju je obavljena usporedba genetskog kaljenja s nekim drugim evolucijskim algoritmima rješavanjem istog problema različitim algoritmima pod istim uvjetima.

KLJUČNE RIJEČI

Genetsko kaljenje, evolucijski algoritmi, genetski algoritmi, simulirano kaljenje, problem trgovačkog putnika, prirodom inspirirani algoritmi

GENETIC ANNEALING OPTIMIZATION

SUMMARY

In a significant part this thesis covers the theoretical background of evoulutionary algorithms and some other nature-inspired algorithms. It is supposed to portray the idea that is behind every attempt to solve a mathematical or a combinatorial problem using the principles and tools of the nature. The main goal, however, is the application of a genetic annealing algorithm for optimizing the travelling salesman problem. In this practical use of genetic annealing, intention was to bring out the characteristics of the algorithm in order to express it's capabilities and shortcomings, qualities and flaws. By reconfiguring the algorithm with a different parameters and then observing the results attempt was made to capture the essence of genetic annealing. In the end, a comparison was made of genetic annealing with some of the other evolutionary algorithms by applying them on the same problem under the same conditions.

KEY WORDS

Genetic annealing, evoulutionary algorithms, genetic algorithms, simulated annealing, travelling salesman problem, nature-inspired algorithms