SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 556

Usporedba heurističkih algoritama optimiranja

Ivan Kravarščan

Zagreb, lipanj, 2009.

Sadržaj

1. Uvod 1

2. Evolucijski algoritmi 2
   2.1. Općenito 2
   2.2. Simulirano kaljenje 3
   2.3. Genetsko kaljenje 3
      2.3.1. Funkcije i parametri 4
   2.4. Optimizacija rojem čestica 5
      2.4.1. Funkcije i parametri 6
   2.5. Optimiranje odbijanjem roja čestica 7
   2.6. Primjena 7
      2.6.1. Problem dijeljenja grafa 7
      2.6.2. Problem bojanja grafa 8

3. Programsko ostvarenje 9
   3.1. Opisnici rješenja 9
   3.2. Ostvarenje algoritama 11

4. Mjerenja 12
   4.1. Podešavanje parametara algoritama 12
      4.1.1. Parametri simuliranog kaljenja 12
      4.1.2. Parametri genetskog kaljenja 13
      4.1.3. Parametri PSO-a 14
      4.1.4. Parametri RPSO-a 16
   4.2. Uspoređivanje algoritama na GPP problemu 17
   4.3. Uspoređivanje algoritama na GCP problemu 18
   4.4. Uspoređivanje na velikom problemu 19

5. Zaključak 22

6. Literatura 23

7. Sažetak 24

1. **Uvod**

Elektronička računala imaju vrlo široku primjenu no granice njihovih mogućnosti ne obuhvaćaju sposobnost rješavanja mnogih praktičnih problema. Jedno od ograničenja jest složenost algoritama s kojima mogu raditi u realnom vremenu. Računala u većini slučajeva ne mogu potpuno riješiti NP težak problem. Razlog tome je činjenica da s povećanjem opsega problema broj operacija potrebnih za potpuno rješavanje te grupe problema raste eksponencijalno. Zato su razvijeni mnogi algoritmi kojima se za takve probleme mogu pronaći gotovo optimalna rješenja. Najjednostavniji takav algoritam je pohlepni algoritam. Njegova jednostavnost uzima danak u kvaliteti rješenja; pohlepni algoritam će dati zadovoljavajuće rješenje samo ako je odmah u početku pretpostavljeno rješenje koje je blizu nekog dobrog rješenja tj. ako početno rješenje nije blizu tzv. lokalnog optimuma. Mnogi heuristički algoritmi nadopunjuju pohlepni algoritam kako tehnikama bijega iz lokalnog optimuma tako tehnikama za bržu konvergenciju prema globalnom optimumu.

2. **Evolucijski algoritmi**

2.1. Općenito

Računski problemi se mogu kategorizirati prema vremenskoj i prostornoj složenosti potrebnoj da se odredi rješenje. Teorija računarstva definira mnogo razreda složenosti među kojima su u praktičnoj primjeni najvažniji P i NP razredi.

P problemi su oni problemi čije se rješenje može naći u tzv. polinomnom vremenu, koristeći deterministički Turingov stroj (matematički model po kojem funkcioniraju današnja računala). Pronalazak rješenja za problem veličine *n* (*n* je tipično broj elemenata u problemu koje treba obraditi), nekim postupkom, u polinomnom vremenu znači da se broj operacija koje će taj postupak provest, može aproksimirati polinomnom funkcijom s varijablom *n*, za dovoljno velike *n*-ove.

Skup NP problema je formalno nadskup skupa P, ali vrlo često se pod NP razmatraju oni problemi koji za koje nije nađen način kako ih svede na P složenost. NP znači "nedeterministički polinomalan" odnosno da se takav problem može riješiti nedeterminističkim Turingovim strojem (Turingov stroj koji može pokrenuti neograničen broj podstrojeva) u kojem svaki podautomat rješava problem P složenosti. Drugi način rješavanja NP problema moguć je pomoću determinističkog Turingovog stroja, algoritmom koji je složeniji od polinomne složenosti. U principu takvi algoritmi imaju eksponencijalnu složenost koja za relativno malu veličinu problema, zahtjeva vrlo dugo vrijeme rješavanja. Budući da suvremenom tehnologijom nije moguće izvesti stroj koji bi dovoljno dobro oponašao nedeterministički Turingov stroj i pošto su mnogi bitni problemi upravo NP složenosti, razvijeni su brojni algoritmi koji rješavanjem P problema nastoje doći do čim boljeg rješenja NP problema. Tome ide u prilog sljedeće svojstvo većine NP teških problema: valjanost rješenja može se provjeriti determinističkim algoritmom polinomne složenosti. Upravo zbog te činjenice većina tih algoritama NP probleme rješavaju pogađanjem rješenja, tzv. metaheurističkim postupkom.

Najjednostavniji takav algoritam je pohlepni algoritam, postupak koji u svakoj iteraciji pretpostavi neko nasumično rješenje i pamti samo najbolje dobiveno. Stohastična priroda takvog postupka implicira da je kvaliteta rješenja dobivenog tim algoritmom podložna faktoru sreće te da povećanjem prostora stanja opada vjerojatnost pronalaska dobrih riješenja. Kako bi se ta vjerojatnost povećala osmišljeni su tzv. algoritmi usmjerenog pretraživanja koji na temelju posjećenih stanja biraju slijedeće stanje. Dio takvih algoritama, a među njima su genetsko kaljenje i optimizacija rojem čestica, spada u evolucijske algoritme, algoritme koji oponašanjem evolucijskih procesa iz prirode usmjeravaju svoje pretraživanje.

2.2. Simulirano kaljenje

Postupak simuliranog kaljenja (engl. *simulated annealing*, u daljnjem tekstu SA), nezavisno su razvili S. Kirkpatrick, C. D. Gelatt i M. P. Vecchi 1983. te V. Černý 1985. Kaljenje je postupak polaganog hlađenja vruće mase (npr. stakla ili metala) kako bi se čim pravilnije formirali kristali i time dobio čvršći materijal. Simulirano kaljenje je heuristički algoritam koji oponaša pronalazak stanja minimalne energije u procesu kaljenja metala. To se postiže oponašanjem učinka temperature u gibanju čestica kroz stanja različitih energija i postepenim smanjivanjem te temperature. U simuliranom kaljenju gleda se ponašanje samo jedne čestice (rješenja) u procesu kaljenja.

Ovaj algoritam je nadogradnja pohlepnog algoritma konceptom temperature odnosno uvođenjem postupka hlađenja. Oponašanjem hlađenja nastoji se izbjeći loša karakteristika pohlepnog algoritma a to je zaustavljanje u lokalnom minimumu koji je puno lošiji od globalnog minimuma. Princip optimizacije je sljedeći: u svakoj iteraciji, za stanje *s* iz prethodne iteracije, bira se proizvoljno susjedno stanje *s'*, ukoliko je energija stanja *s'* manja od energije stanja *s*, čestica prelazi u stanje *s'* a ako je stanje *s'* stanje više energije, *s* prelazi u *s'* uz vjerojatnost koja ovisi o temperaturi i razlici energija stanja *s* i *s'*.

Algoritam kreće od neke zadane "visoke" temperature te ju kroz iteracije smanjuje do "apsolutne nule". Funkcija vjerojatnosti prelaska iz stanja niže u stanje više energije može biti bilo koja funkcija, uz ograničenja da je ta funkcija pozitivna te da njena vrijednost opada s padom temperature i porastom razlika energija između stanja.

```
s = početno_stanje
temperatura = početna_temperatura
dok temperatura > APSOLUTNA_NULA radi:
    s' = proizvoljan_susjed(s)
    dE = energija(s') - energija(s)
    ako dE < 0 ili f(temperatura, dE) > random() tada:
        s = s'
    temperatura = nova_temperatura(temperatura)
rješenje = s
```

Slika 2.1. Pseudokod simuliranog kaljenja.

2.3. Genetsko kaljenje

Kod pretraživanja simuliranim kaljenjem, nerijetko se prakticira višestruko ponavljanje postupka kako bi se dobilo što bolje riješenje. Algoritam genetskog kaljenja kojeg je Kennet V. Price 1994. objavio u časopisu "Dr. Dobb's Journal"

proširuje simulirano kaljenje upravo s tom idejom. Umjesto da se rješenja simuliranog kaljenja generiraju nezavisno jedno od drugog, generiraju se paralelno i u istom sustavu. Čestice, kao i u simuliranom kaljenju, nezavisno traže stanje minimalne energije ali razmjenjuju energiju koja omogućava prelazak u lošije stanje. Ukoliko neka čestica prijeđe u povoljnije stanje (stanje manje energije), ona emitira energiju u sustav i tu energiju ostale čestice mogu iskoristiti za prelazak u nepovoljnija stanja. Hlađenje se ostvaruje tako da se nakon svake iteracije dio slobodne energije, energije koja ne pripada niti jednoj čestici, izgubi (emitira van sustava). Na taj način se još bolje imitira stvarno kaljenje.

Genetsko kaljenje na jednostavan način može odlučiti da li čestica može preći u nepovoljnije stanje. Metoda koju je Price predložio je da se prije svake iteracije slobodna energija *E* jednoliko rasporedi po česticama tako da svaka može preći u stanje koje ima maksimalno za $\frac{E}{N}$ (*N* je broj čestica) veću energiju od početne čestice.

```
čestice = nasumične_čestice(N)
iteracija = 0
dok iteracija < brIteracija:
    prag = slobodna_energija / N
    slobodna_energija = 0;
    za svaku česticu:
        mutant = mutiraj(česticu)
        ako energija(mutant) < energija(čestica) + prag:
            razlika = energija(čestica) + prag – energija(mutant)
            slobodna_energija = slobodna_energija + razlika
            čestica = mutant
    iteracija = iteracija + 1
    slobodna_energija = slobodna_energija * C
rješenje = najbolja(čestice)
```

Slika 2.2. Pseudokod genetskog kaljenja.

2.3.1. Funkcije i parametri

Kao što se vidi iz pseudokoda, da bi se problem mogao riješiti genetskim kaljenjem, potrebno je odrediti dvije funkcije nad česticom: funkciju energije (dobrote) i funkciju mutacije. Implicitno se nameće i potreba za definicijom čestice. Za česticu je potrebno pronaći strukturu kojom će se jednoznačno opisivati riješenje zadanog problema. Dizajn strukture čestice bitno utječe na performanse navedenih dviju funkcija a time i na performanse cijelog algoritma.

Funkcija energije za zadanu česticu računa apstraktnu vrijednost dobrote odnosno energije. Ovisno o pogledu na problem, može se definirati da bolja

čestica ima veću vrijednost te funkcije (npr. veću dobrotu) ili da ima manju vrijednost (npr. manju energiju).

Funkcija mutacije za danu česticu stvara njezinu izmijenjenu varijantu. Tehnike i intenzitet mutacije bitno utječu na kvalitetu rješenja. Usprkos tome što raspoložive tehnike uvelike ovise o problemu koji se rješava a još više o načinu na koji se kodira riješenje, iz Price-ovog članka može se zaključiti da kombinacija više tehnika rezultira boljim mutantom.

Nakon definiranja tih dviju funkcija, potrebno je odrediti parametre algoritma. Genetsko kaljenje samo po sebi zahtjeva dva parametra: *N* (broj čestica) i *C* (koeficijent hlađenja odnosno koliki udio slobodne energije se zadržava u sustavu na kraju svake iteracije). Ovisno o strategiji mutacije, algoritam može imati dodatne parametre. Vrijednosti parametara su također bitna tema jer svaki problem ima drugačiji odziv na te vrijednosti. Kod nekih problema produktivnije je brže hlađenje, kod nekih drugih sporije, kod nekih trećih pak intenzitet mutacije mora biti mali, itd.

Pri posebnim vrijednosti parametara, genetsko kaljenje se može svesti na simulirano kaljenje i pohlepni algoritam. Ako genetsko kaljenje radi s jednom česticom ($N = 1$), tada je ekvivalentno simuliranom kaljenju. Pohlepni algoritam se dobiva kada genetsko kaljenje radi samo s jednom česticom i kada se nepovoljna stanja nikad ne prihvaćaju tj. kada ukloni utjecaj temperature ($C = 0$ i $N = 1$).

2.4. Optimizacija rojem čestica

Algoritam optimizacije rojem čestica (engl. *particle swarm optimization* u daljnjem tekstu PSO) je stohastički, populacijski algoritam iz kategorije inteligencije roja. Prvi su ga opisali dr. James Kennedy i dr. Russell C. Eberhart 1995. Ideja algoritma se bazira na sociološko-psihološkim principima i motivirana je ponašanjem raznih grupa organizama poput jata ptica ili ribljih plova.

Ako promatramo jato ptica koje traže hranu, sve ptice će prvotno samostalno tražiti hranu a kada neka pronađe dobar izvor hrane, ostale će je slijediti. Međutim, neke ptice se privremeno odvajaju od jata kako bi potražile još bolje hranilište. Uspjehom u takvoj potrazi pomažu cijelome jatu.

Izvorni PSO je baziran na tim činjenicama i oponaša takvo ponašanje: inicijalno su sve jedinke (čestice) na nasumičnim pozicijama i kreću se u nasumičnom smjeru a tokom svake iteracije svaka pojedina čestica mijenja svoji smjer kretanja dijelom prema najboljem rješenju kojeg je sama našla a dijelom prema najboljem riješenju susjednih čestica. Prema tome, svaka čestica vrši optimizaciju pozivajući se na vlastito znanje i na znanje susjednih čestica (ako se sve čestice promatraju kao susjedi tada je riječ o znanju cijelog roja).

To ponašanje je opisano jednadžbom 2.2 gdje su $\overrightarrow{x_{k+1}}$ vektor novog položaja, $\overrightarrow{x_k}$ vektor položaja iz prethodne, *k*-te iteracije, koeficijent dt koji oponaša vremenski razmak između dviju iteracija te brzina $\overrightarrow{v_{k+1}}$. Brzina $\overrightarrow{v_{k+1}}$ predstavlja brzinu čestice u *k+1* iteraciji a njena promjena u odonosu na *k*-tu iteraciju, računa prema jednadžbi 2.1. Koeficijenti ω, c_l, $c_g$određuju utjecaj inercije, utjecaj vlastitog znanja te utjecaj znanja roja. Vektorima $\overrightarrow{x_k}$, $\overrightarrow{v_k}$, $\overrightarrow{p_{lk}}$, $\overrightarrow{p_{gk}}$ određeni su položaj i

brzina u prethodnoj iteraciji te pozicija vlastitog najboljeg rješenja i pozicija najboljeg rješenja susjeda. r_l i r_g su nasumične vrijednosti iz intervala $[0, 1]$.

$$\overrightarrow{v_{k+1}} = \omega\overrightarrow{v_k} + c_l r_l(\overrightarrow{p_{lk}} - \overrightarrow{x_k}) + c_g r_g(\overrightarrow{p_{gk}} - \overrightarrow{x_k}) \quad (2.1)$$

$$\overrightarrow{x_{k+1}} = \overrightarrow{x_k} + dt * \overrightarrow{v_{k+1}} \quad (2.2)$$

```
čestice = nasumične_čestice(N)
iteracija = 0
dok iteracija < brIteracija:
    za svaku česticu:
        ako energija(čestica) < energija(čestica.najbolje):
            čestica.najbolje = čestica
        ako energija(čestica) < energija(najbolje):
            najbolje = čestica
    za svaku česticu:
        rl = random()
        rg = random()
        čestica.brzina = omega * čestica.brzina
                + cl * rl * (čestica.najbolje – čestica.pozicija)
                + cg * rg * (najbolje – čestica.pozicija)
        Čestica.pozicija = čestica.pozicija
                + dt * čestica.brzina
```

Slika 2.3. Pseudoko algoritma optimizacije rojem čestica.

2.4.1. Funkcije i parametri

Funkcija energije čestica je jedina funkcija koju ovaj algoritam zahtjeva no da bi se nad česticom mogla vršiti operacija kretanja, oblik čestice mora biti takav da se iz njega mogu izlučiti pozicija u n-dimenzijonalnom prostoru te da se na njega može primjenjivati brzina.

Jedan od parametara algoritma je, kao i u simuliranom i genetskom kaljenju, broj čestica *N*. Preostali parametri vidljivi su iz jednadžbi 2.1 i 2.2 a to su dt, ω, c_l i c_g.

2.5. Optimiranje odbijanjem roja čestica

Optimizacija odbijanjem roja čestica (engl. *Repulsive particle swarm optimization*, u daljnjem tekstu RPSO), je jedna od inačica PSO algoritma. Promjena u odnosu na osnovni PSO jesu jednadžbe kojima se računaju brzina i pozicija:

$$\overrightarrow{v_{k+1}} = \omega\overrightarrow{v_k} + ar_l(\overrightarrow{p_{lk}} - \overrightarrow{x_k}) + \omega br_y(\overrightarrow{p_{yk}} - \overrightarrow{x_k}) + \omega cr_3\vec{z} \tag{2.3}$$

$$\overrightarrow{x_{k+1}} = \overrightarrow{x_k} + \overrightarrow{v_{k+1}} \tag{2.4}$$

U jednadžbi brzine utjecaj znanja susjednih čestica je zamijenjen utjecajem blizine nasumične čestice. Vektor $\overrightarrow{p_{yk}}$ je pozicija nasumce odabrane čestice roja i od nje se čestica za koju se računa brzina nastoji odbiti. Da bi zbilja dolazilo do odbijanja, parametar b mora imati negativnu vrijednost. U jednadžbu je još dodan novi član, nasumični vektor $\vec{z}$ koji brzini dodaje dozu kaotičnosti.

2.6. Primjena

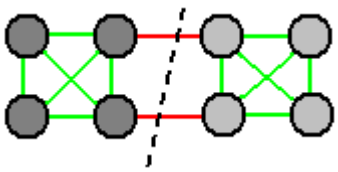
Heuristički algoritmi se mogu primjenjivati u rješavanju velikog broja NP teških problema. Genetsko i simulirano kaljenje za opis rješenja zahtijevaju samo da se definiraju funkcija dobrote rješenja i funkcija mutacije i zbog toga su primjenjivi na jako velik broj problema. PSO i RPSO su s druge strane ograničeni na probleme kod kojih se mogu definirati pojmovi pozicija i brzina. Zahvaljujući raznim metodama opisivanja stanja kao točke u n-dimenzijonalnom prostoru, to je još uvijek širok opseg problema.

U slijedećim poglavljima su opisani su problem dijeljenja grafa i problem bojanja grafa. Oba problema su statički problemi (problem se ne mijenja tokom vremena) no heurističkim algoritmima mogu se rješavati i dinamički problemi kao što je npr. problem mrežnog usmjeravanja.

2.6.1. Problem dijeljenja grafa

Problem dijeljenja grafa (engl. *graph partitioning problem*, u daljnjem tekstu GPP) je problem podjele vrhova grafa u *m* skupova tako da su ti skupovi otprilike jednake veličine i da minimalan broj bridova veže vrhove u različitim skupovima. Praktični primjer tog problema je podjela *n* komponenti na *m* čipova, tako da su čipovi minimalno povezani.

Za $m = 2$ problem je riješiv u polinomnom vremenu no egzaktno rješavanje bi za $m > 2$ zahtjevalo ispitivanje svih kombinacija. Kako je ukupan broj tih kombinacija m^n algoritam koji ispituje te kombinacije bi imao eksponencijalnu složenost.

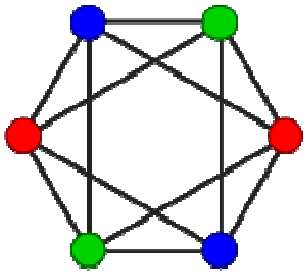


Slika 2.4. Primjer podjele grafa. Tamno sivi vrhovi dodijeljenj su jednom skupu a svijetlo sivi drugom.

2.6.2. Problem bojanja grafa

Problem bojanja grafa (engl. *graph coloring problem*, u daljnjem tekstu GCP) je problem u kojem svaki vrh grafa treba obojati jednom od raspoloživih boja tako da susjedni vrhovi nemaju istu boju. Slikovito se taj problem može prikazati kao problem kada na zemljopisnoj karti treba obojati regije tako da nema susjednih regija iste boje. Mnogi problemi su svedivi na problem bojanja grafa, među najpoznatijim takvim problemima su raspoređivanje poslova i dodijela registra tokom prevođenja programskih jezika.

GPP i GCP su dva jako slična problema koji se razlikuju po tome što je jedan minimizacija a drugi maksimizacija susjednosti vrhova dodijeljenih različitim skupovima. Ta razlika bitno mijenja rješenja problema ali minimalno utječe na način rješavanja pomoću heurističkih algoritama.



Slika 2.5. Primjer grafa obojanog s tri boje.

3. **Programsko ostvarenje**

U sklopu rada, u programskom jeziku Java, načinjeno je programsko ostvarenje sva četiri algoritma za probleme podijele i bojanja grafa. Algoritmi su dizajnirani tako da ne znaju koji konkretan problem rješavaju odnosno da rade nad listom apstrahiranih opisnika problema. Tu listu pozivatelj, prije pokretanja algoritma, popunjava s populacijom opisnika konkretnog problema. Time je ostvareno da algoritmi nisu specijalizirani za rješavanje točno određenog problema nego da mogu rješavati bilo koji problem opisan razredom koji nasljeđuje apstraktan opisnik.

3.1. Opisnici rješenja

Struktura podataka u razredima koji opisuju rješenja (čestice) GPP i GCP problema je identična a razlomljena je u dvije cjeline: tablica mapiranja koja sadrži informacije koji vrh pripada kojem skupu i 3 liste realnih brojeva koje predstavljaju trenutnu poziciju, brzinu čestice te poziciju najboljeg rješenja te čestice. Pošto algorimi rade sa apstraktnim opisnikom, odgovornost za sve promjene na česticom je dodijeljena konkretnim razredima opisnika. Zato opisnici sadrže ostvarenje mutacije za SA i GAn, metode za pomak pri optimizaciji rojem čestica i funkciju energije.

Osnovna struktura kojom se unutar opisnika opisuje rješenje je tablica mapiranja vrhova po skupovima. Ključ mapiranja je objekt koji opisuje vrh grafa a vrijednost cijeli broj koji predstavlja skup kojem je vrh pridjeljen. Skupovi su „imenovani“ cijelim brojevima iz intervala [0, *m*-1] gdje je *m* broj skupova.

vrh1 → 0	vrh4 → 0
vrh2 → 1	vrh5 → 1
vrh3 → 1	vrh6→ 0

Slika 3.1. Primjer tablice mapiranja šest vrhova u dva skupa.

Funkcija energije kod opisnika GPP problema opisana je pseudokodom 3.2. Za svaki vrh *v*, preko reference na opisnik grafa, dohvaća se skup vrhova s kojima je *v* susjedan. Za svakog susjeda v_s vrha *v* koji prema tablici mapiranja ne pripada skupu u kojem je *v*, energija čestice povećava se za jedan. Tako ostvarena funkcija energije će svaki konfliktni brid prebrojati dva puta. Funkcija energije kod opisnika za GCP problem ostvarena je na istini način s izmjenom da se energija povećava za susjede koji su pridjeljenji istom skupu.

```
energija = 0
za svaki vrh v grafa:
    za svakog susjeda v_s vrha v:
        ako skup(v) ≠ skup(v_s):
            energija = energija + 1
```

Slika 3.2. Pseudokod funkcije energije opisnika GPP-a.

Funkcija mutacije u oba opisnika ostvarena je na jednak način. Prvo se bira jedna od dviju strategija. Jedna strategija je zamjena pripadnosti, jednostavna strategija koja odabire dva nasumična vrha i u tablici mapiranja zamjenjuje skupove kojima oni pripadaju. Druga strategija je obrtanje podniza liste vrhova. Naime lista vrhova nije pohranjena u opisniku čestice već u objektu koji opisuje graf. Pohrana vrhova u listi implicitno dodjeljuje vrhovima indekse (možemo odrediti koji vrh je prvi, koji drugi, ... i koji je zadnji) te definira njihov redoslijed što je pogodno za neke operacije. Mutacija obrtanjem odabire podniz u toj listi i mijenja tablicu mapiranja u čestici kao da je redoslijed vrhova u odabranom podnizu obrnut.

Prije mutacije:

vrh1	vrh2	vrh3	vrh4	vrh5	vrh6
0	1	2	0	1	2

Nakon mutacije podniza od vrha 2 do vrha 4:

vrh1	vrh2	vrh3	vrh4	vrh5	vrh6
0	0	2	1	1	2

Slika 3.3. Primjer mutacije obrtanjem.

Tablica mapiranja samo po sebi bi mogla biti dovoljna za definiranje pozicije tokom optimizacije rojem čestica ali pomoću nje je teško definirati brzinu. Zato je za definiranje pozicije i brzine potrebna je drugačija struktura podataka. Najpogodnija takva struktura jest lista realnih brojeva u kojoj svaki element predstavlja vrijednost pojedine komponente pozicije u n-dimenzionalnom prostoru. Preslikavanje tablice mapiranja u takvu listu je jednostavno, svaki vrh se izrazi kao jedna komponenta prostora tako da je vrijednost te komponente jednaka broju skupa (kada se brojčano ime promatra kao broj) kojem vrh pripada. No da bi se nakon primjene brzine na poziciju iz liste mogla izgraditi tablica mapiranja, potreban je složeniji postupak jer treba paziti da se veličine skupova ne mijenjaju te da se ne izgube postojeći ili stvore novi skupovi. Korišteni postupak za to preslikavanje je sljedeći. Recimo da imamo *m* skupova i *n* vrhova. Prvih $\frac{n}{m}$ vrhova

koji imaju najmanju vrijednost u listi, smještaju se u skup „0“ , slijedećih $\frac{n}{m}$ po vrijednosti, u skup „1“ itd. Pomoću lista realnih brojeva i ovih dvaju preslikavanja, u opisnicima GPP-a i GCP-a ostvarene su informacije o poziciji, brzini i najboljem rješenju tokom optimizacije PSO-om i RPSO-om.

Tablica mapiranja:

vrh1	vrh2	vrh3	vrh4	vrh5	vrh6
0	0	2	1	1	2

Pozicija čestice:

0	0	2	1	1	2

Pozicija nakon pomaka:

0.5	0.8	1.2	1.4	0.7	2.2

Tablica mapiranja nakon pomaka:

vrh1	vrh2	vrh3	vrh4	vrh5	vrh6
0	1	1	2	0	2

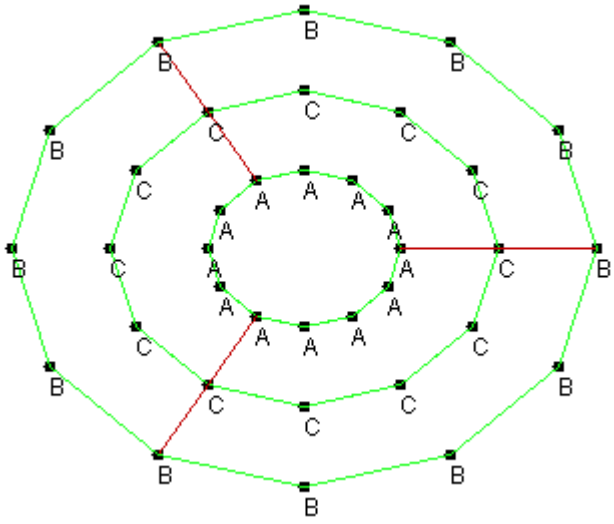
Slika 3.4. Primjer preslikavanja iz tablice mapiranja u listu i obrnuti postupak.

3.2. Ostvarenje algoritama

Implementacije algoritama su napravljene u skladu sa pseudokodovima iz 2. poglavlja. Simulirano kaljenje je ostvareno kao genetsko kaljenje s jednom česticom. Oponašanje utjecaja temperature u oba algoritma je ostvareno je po Priceovoj metodi: svaki prijelaz iz lošijeg u bolje stanje emitira energiju u sustav i tu energiju u slijedećoj iteraciji čestice mogu koristiti za prelaz u lošije stanje. Pošto su implementirane dvije strategije mutacije SA i GA uz parametre *N* i *C* imaju dodatne parametre *EX* i *PR* koji određuju inteznitet i izbor metode mutacije. Parametar *EX* za mutaciju zamjenom određuje ograničenje na broj parova nad kojima se vrši zamjena. Tokom mutacije zamjenom, opisnik problema nasumično bira broj parova koji je u rasponu $[1, \frac{n*EX}{2}]$. Za mutaciju obrtanjem, parametar *EX* određuje najveću duljinu podniza za obrtanje. Duljine se biraju iz raspona $[2, \frac{n*EX}{2}]$. Parametar *PR* određuje strategiju mutacije tako da se tokom svake mutacije uz vjerojatnost *PR* odabire mutacija obrtanjem a inače se vrši mutacija zamjenom.

4. Mjerenja

Odabrani problemi na kojima su vršena mjerenja su GPP i GCP na grafu prikazonom na slici 4.1. Broj skupova u koje se raspoređuju vrhovi je tri. Na toj je slici također prikazano najbolje riješenje za GPP problem. Dodatno, izvršeno je mjerenje na većem problemu.



Slika 4.1. Graf na kojem su vršena mjerenja.

4.1. Podešavanje parametara algoritama

Prije uspoređivanja algoritama bilo je potrebno odrediti parametre pri kojima će algoritmi najbolje rješavati zadani problem. Svaki algoritam ugađan je na zaseban način a ono što je zajedničko svim ispitivanjima jest da je za svaku konfiguraciju parametera izvršeno 31 pokretanje algoritma te zabilježena suma energija rješenja generiranih tom konfiguracijom, unutar 2.5 sekunde rada algoritma.

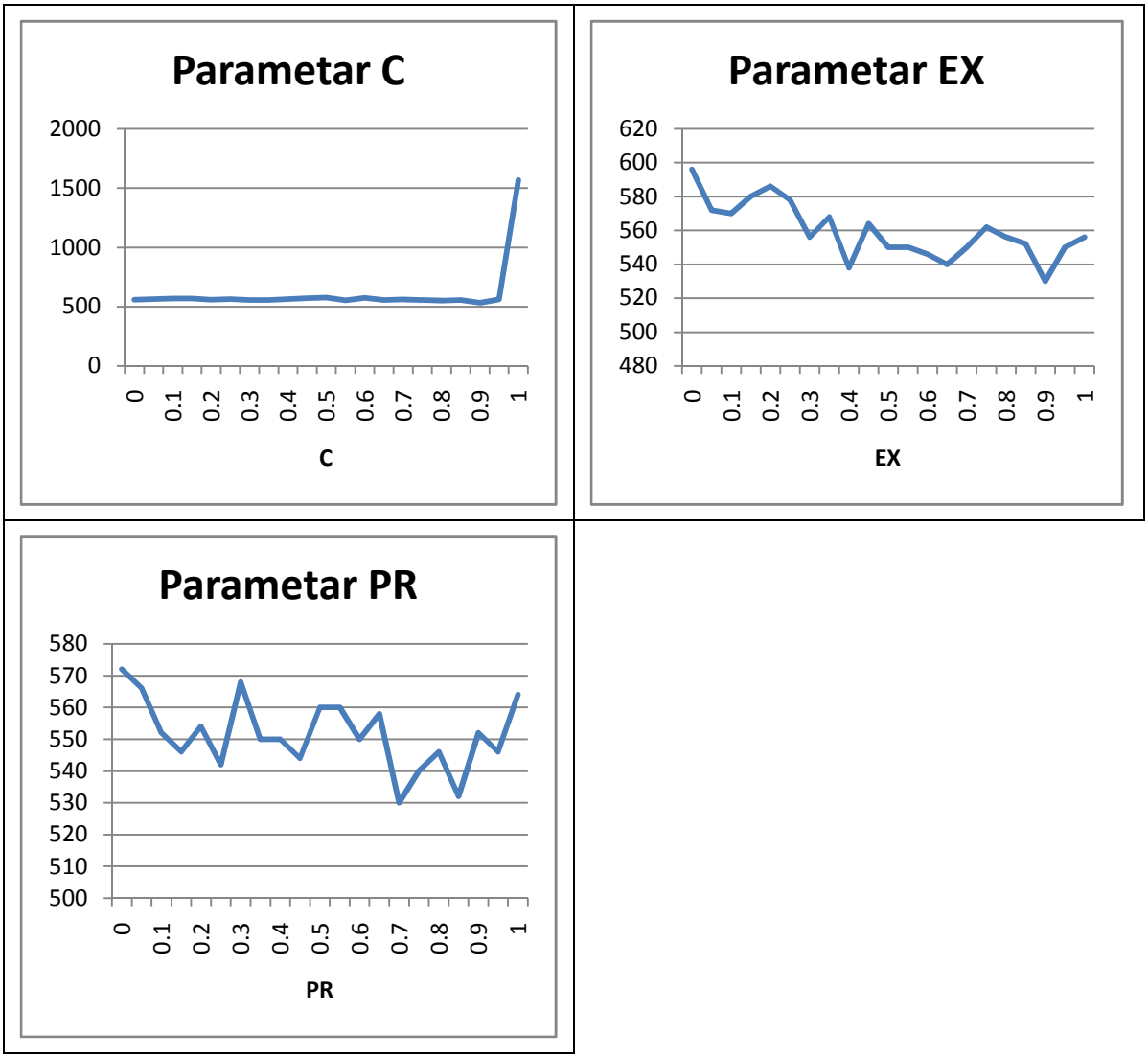
4.1.1. Parametri simuliranog kaljenja

Parametri za simulirano kaljenje određivani su slijedećim koracima:

1. Određivanje parametra *C* uz unaprijed zadane parametre *EX* i *PR.*
2. Određivanje parametra *EX* uz unaprijed zadan parametar *PR* te uz najbolju vrijednost za parametar *C* iz prethodnog koraka.
3. Određivanje parametra *PR* uz najbolje vrijednosti za parametara *C* i *EX* iz prethodnih koraka.

Dobiveni rezultati prikazani su u tablici 4.1. Kao što se može vidjeti, vrijednosti parametra *C* slabo utječu na kvalitetu rješenja. Iznimka je slučaj kada je $C = 1$ jer se tada sustav ne hladi i time sva rješenja bivaju prihvaćena. Ostali parametri u većoj mjeri utječu na rješenja te kao što se vidi iz dijagrama za parametar *PR*, za dani problem mutacija obrtanjem je pogodnija od mutacije

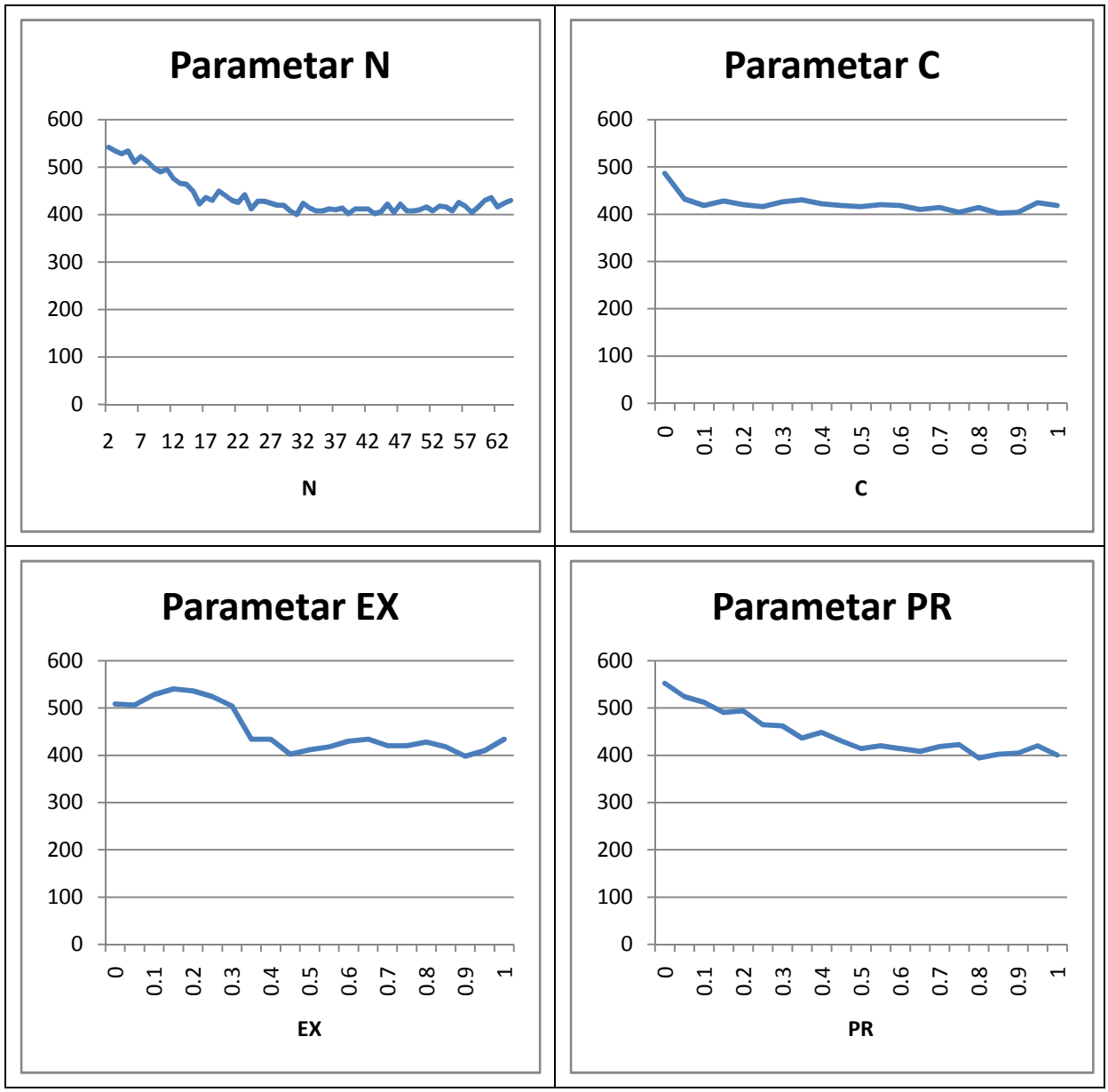
zamjenom. Prema mjerenjima, najbolji parametri za simulirano kaljenje su *C* = 0.85, *EX* = 0.9 i *PR* = 0.7.



Tablica 4.1. Rezultati određivanja parametara simuliranog kaljenja.

4.1.2. Parametri genetskog kaljenja

Parametri za genetsko kaljenje određivani su na sličan način kao i prametri za simulirano kaljenje s dodatkom da je ispitivano ponašanja za prametar *N*.



Tablica 4.2. Rezultati određivanja parametara genetskog kaljenja.

Prema mjerenjima, najbolji parametri za genetsko kaljenje koje riješava problem sa slike 4.1 jesu *N* = 31, *C* = 0.85, *EX* = 0.9 i *PR* = 0.8. Zgodno je za primjetiti da za *C* = 1 genetsko kaljenje nije dalo loše rezultate kao simulirano kaljenje. To bi se moglo obrazložiti time što je ispitivanje parametara *N* bilo prije ispitivanja ostalih parametara te se stoga ispitivanje parametra *C* vršilo s populacijom od 31 jedinke. Za ostale parametre genetsko kaljenje je pokazalo slično ponašanje kao simulirano kaljenje.

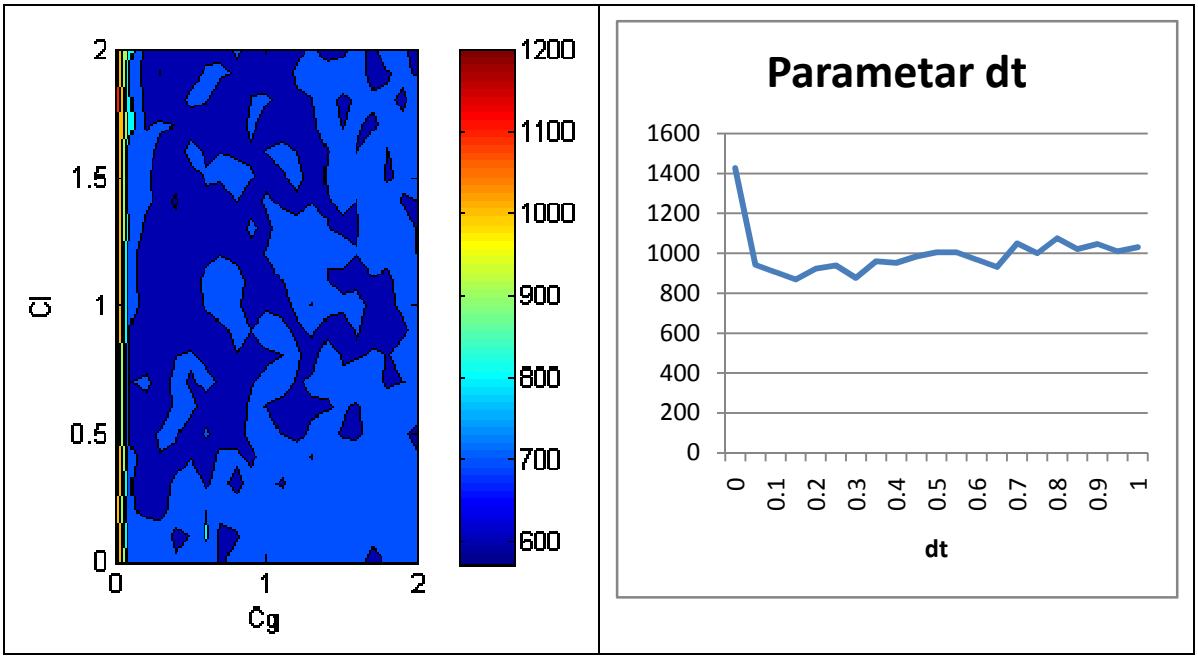
4.1.3. Parametri PSO-a

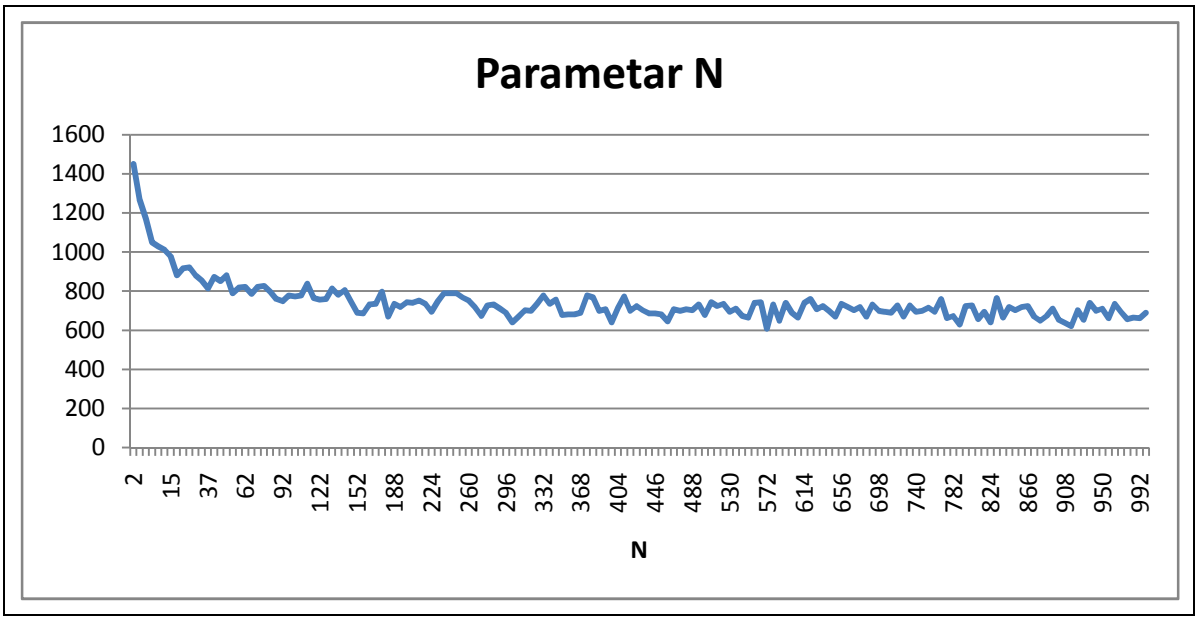
Zbog ovisnosti brzine čestice o parametrima ω, c_l i c_g, bilo je potrebno drugačije pristupiti ispitivanju utjecaja tih parametara. Utjecaj brzine na poziciju zbog množenja sa parametrom dt moguće je podešavati omjerom $\omega : c_l : c_g$. Pošto

je $ax:ay:az$ isti omjer kao i $x:y:z$ a to je pak ekvivalentno omjeru $1:\frac{y}{x}:\frac{z}{x}$ problem ispitivanja uređene trojke parametara može se svesti na ispitivanje uređenog para. Tako je u ispitivanju parametar ω bio postavljen na neku fiksnu vrijednost dok su se ispitivale i mijenjale vrijednosti parametara c_l i c_g. Koraci koji ma se vršilo ispitivanje:

1. Fiksirati parametar ω na neku vrijednost.
2. Određivanje parametara c_l i c_g uz unaprijed zadane parametre *N* i dt.
3. Određivanje parametra dt uz unaprijed zadan parametar *N* te uz najbolju vrijednost za parametara c_l i c_g iz prethodnog koraka.
4. Određivanje parametra *N* uz najbolju vrijednost za parametre dt, c_l i c_g iz prethodnih koraka.

Rezultati dobiveni ispitivanjem pokazuju kako PSO zahtjeva puno veću populaciju rješenja nego genetsko kaljenje te da su najbolji parametri za ovaj algoritam *N* = 572, ω = 1, c_l = 1.4, c_g= 0.4 i dt = 0.15.



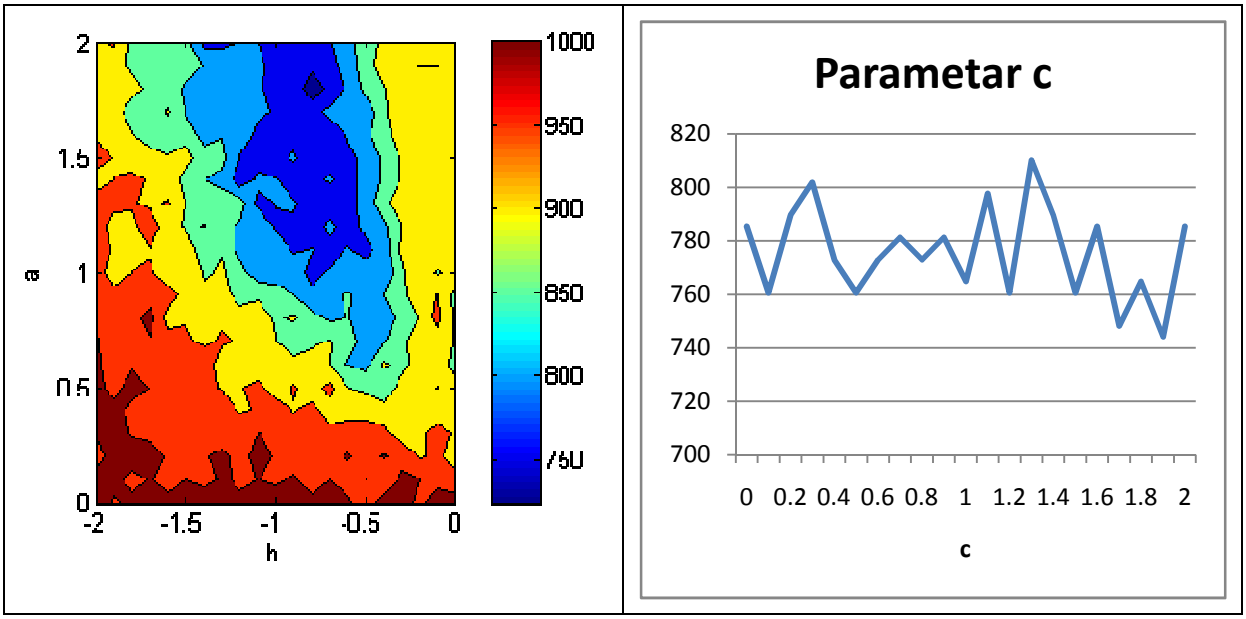


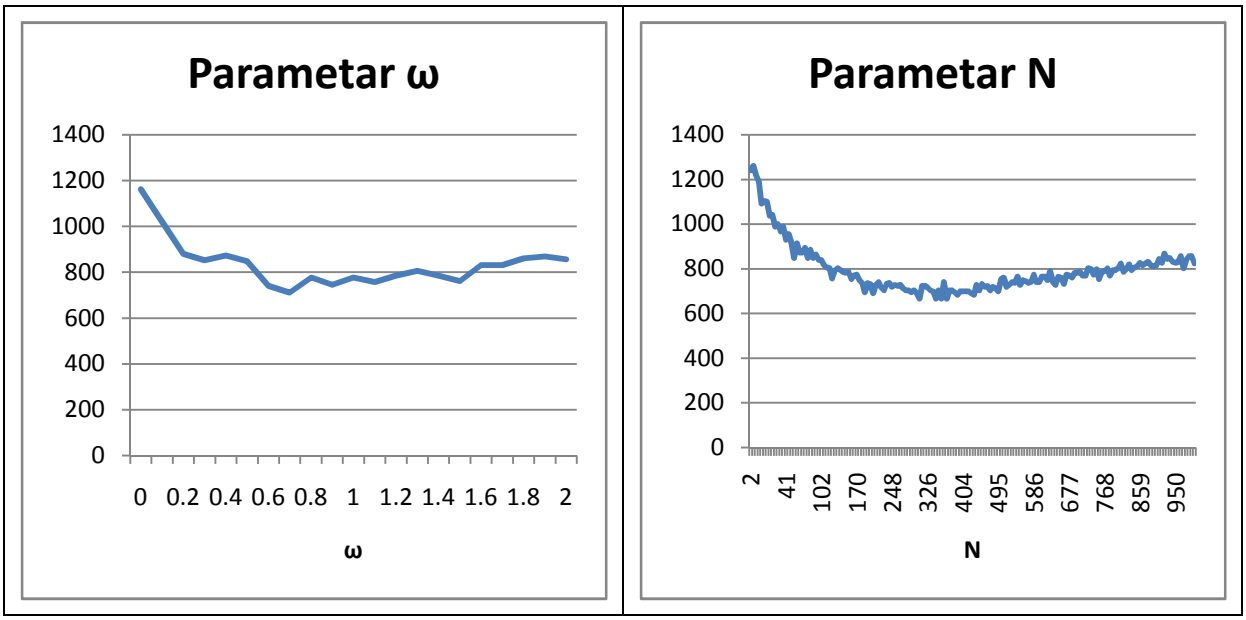
Tablica 4.3. Određivanje parametara za PSO.

4.1.4. Parametri RPSO-a

Ispitivanje parametara za RPSO vršeno je slično ispitivanju za PSO:

1. Određivanje parametara *a* i *b* uz unaprijed zadane parametre *N, c* i ω.
2. Određivanje parametra *c* uz unaprijed zadane parametre *N* i ω te uz najbolju vrijednost za parametara *a* i *b* iz prethodnog koraka.
3. Određivanje parametra ω uz unaprijed zadan parametar *N* te uz najbolju vrijednost za parametara *a*, *b* i c iz prethodnih koraka.
4. Određivanje parametra *N* uz najbolju vrijednost ostalih parametara izvedenih u prethodnim koracima.



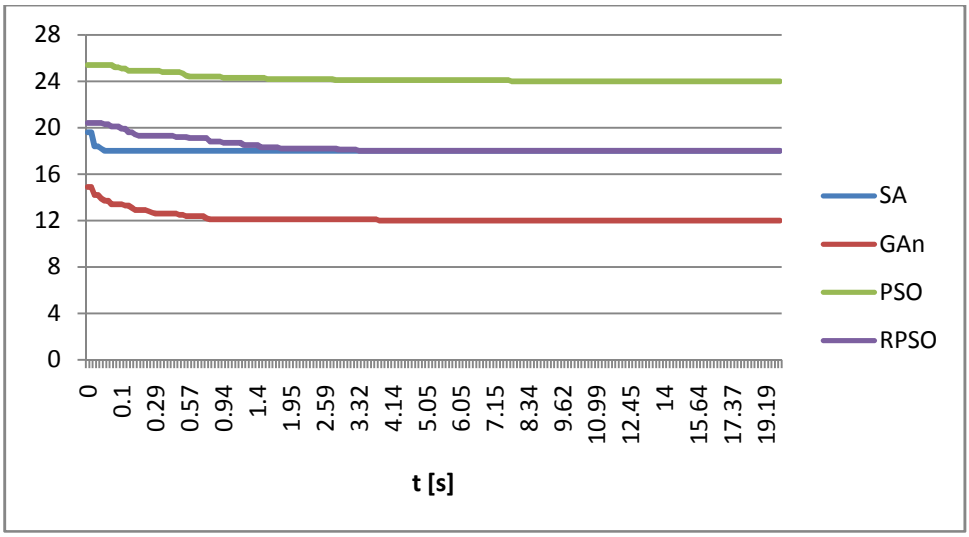


Tablica 4.4: određivanje parametara RPSO-a

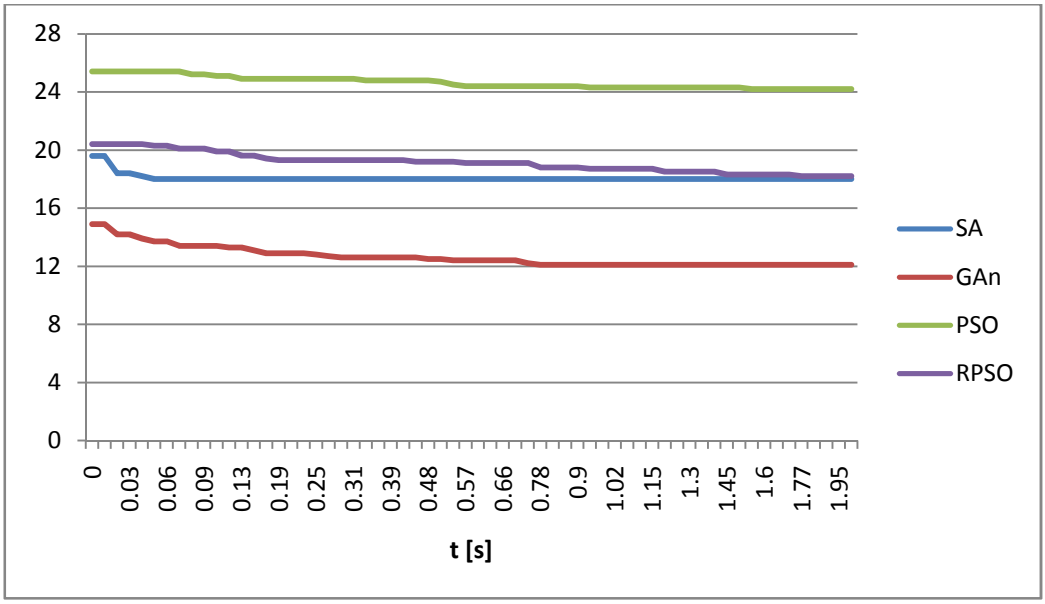
Najbolji parametri za RPSO su $N = 308$, $\omega = 0.7$, $a = 1.8$, $b = -0.8$ i $c = 1.9$. Makar je napravljeno 31 mjerenje za svaku vrijednost parametra c, rezultati su i dalje kaotični no ipak se može zaključiti da su najpogodnije vrijednosti u intervalu [1.7, 1.9].

4.2. Uspoređivanje algoritama na GPP problemu

Uspoređivanje algoritama izvršeno je tako da je svakim algoritmom 20 sekundi vršena optimizacija problema. Na slici 4.2 prikazani su prosječni rezultati od deset mjerenja. Kao što se može vidjeti, genetsko kaljenje je bilo najuspješnije (energija optimalnog rješenja je 12) u pronalaženju rješenja. PSO i RPSO se nisu pokazali vrlo uspješnima. Razlog tome vjerojatno leži u vrlo velikoj dimenzionalnosti prostora. Graf je sastavljen od 36 vrhova što znači da je optimizacija rojem čestica morala raditi sa 36-dimenzionalnim prostorom. Drugi razlog bi mogla biti činjenica da su korištene osnovne varijante tih algoritama. Simulirano kaljenje se pokazalo nešto bolje od PSO-a ali zbog rada nad samo jednom česticom brzo je došlo do stanja stagnacije. Na slici 4.3. prikazani su rezultati prvih 2 sekunde istih mjerenja kako bi se vidjelo napredovanje optimizacije prije stagnacije.



Slika 4.2. Rezultati uspoređivanja algoritama na GPP problemu.

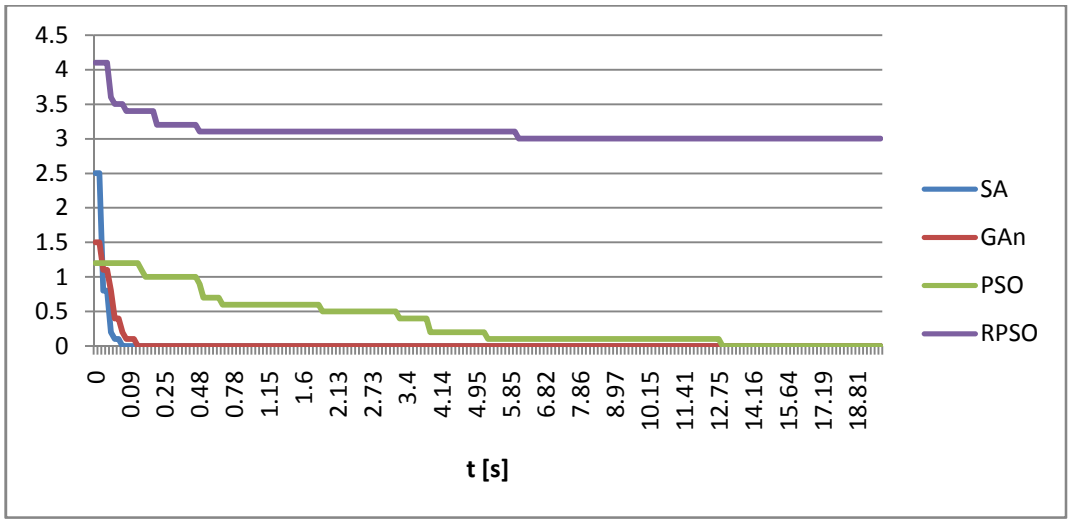


Slika 4.3. Rezultati za prve dvije sekunde rada algoritama.

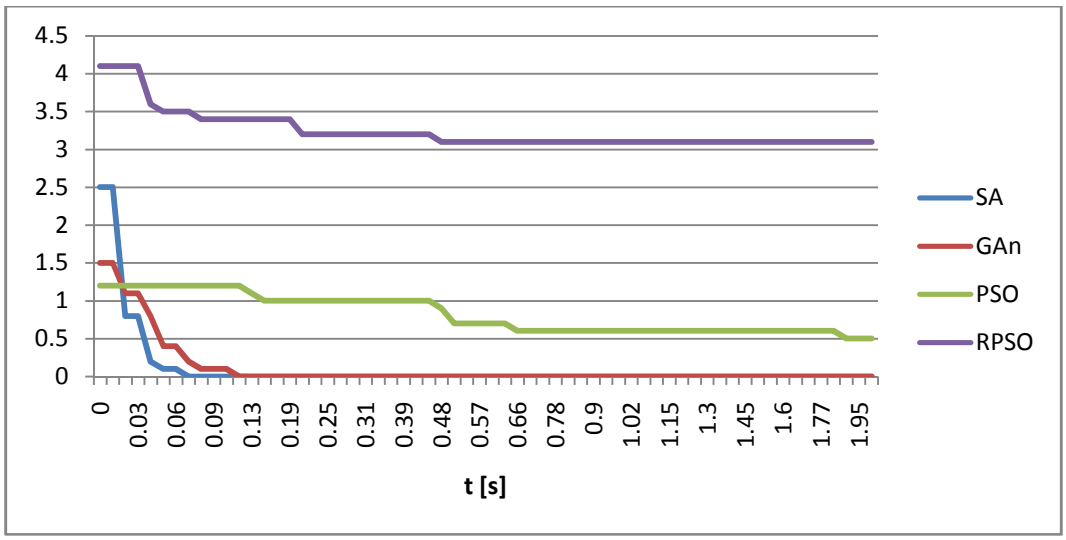
4.3. Uspoređivanje algoritama na GCP problemu

Uspoređivanje algoritama na GCP problemu obavljeno je na isti način kao i na GPP problemu. Na slici 4.4. prikazani su rezultati prosječni mjerenja nakon 20 sekundi optimizacije. Stanje je vidljivo drugačije nego kod uspoređivanja na GPP problemu. Svi algoritmi osim RPSO-a su uspjeli prije stagnacije pronaći optimalno rješenje koje je u ovome slučaju imalo energiju jednaku nuli (graf se može obojati bez konflikata). Odnosi brzina napretka su se također promjenili. Simulirano kaljenje je u prosjeku najbrže stizalo do optimuma a genetsko kaljenje za dvadesetinku sekunde kasnije. PSO se pokazao učinkovit u rješavanju ovog problema.

Kao što se može vidjeti iz mjerenja na ova dva problema, niti jedan algoritam nije najbolji. U slučaju GPP-a, RPSO bio bolji od PSO-a a GAn bolji SA dok se kod problema bojanja grafa ustanovilo drugačije.



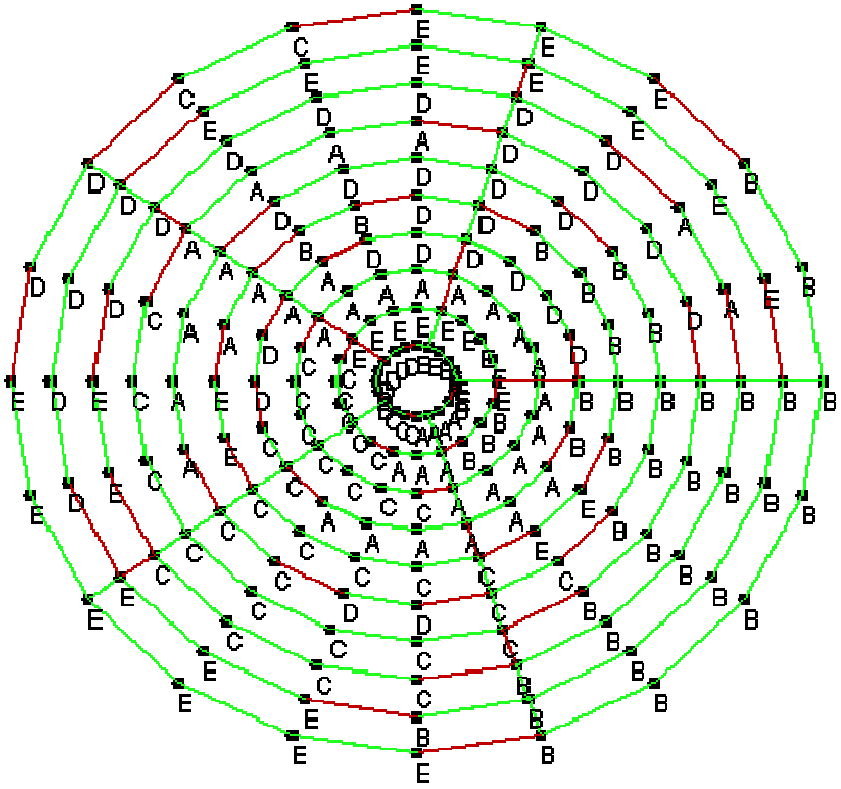
Slika 4.4. Rezultati uspoređivanja algoritama na GCP problemu.



Slika 4.5. Rezultati za prve dvije sekunde rada algoritama.

4.4. Uspoređivanje na velikom problemu

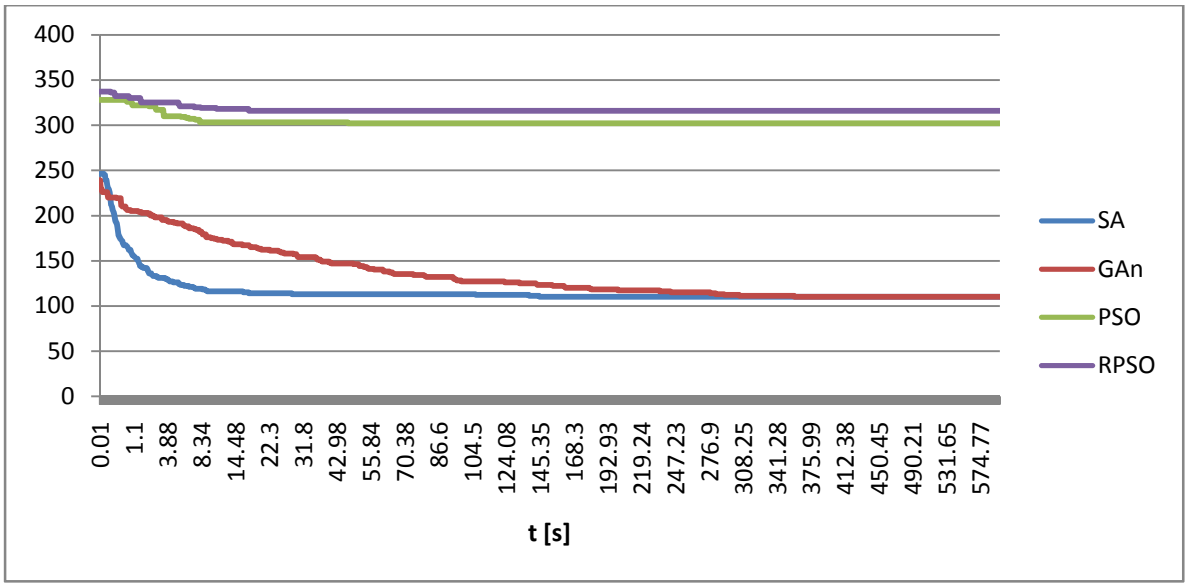
Graf sa slike 4.1., kao što je već rečeno, ima svega 36 vrhova a stvarni problemi su često puno opsežniji. Da bi se ispitalo ponašanje algoritama na opsežnijim problemima, izvršeno je mjerenje optimizacije za GPP problem na grafu sa slike 4.7. (graf sa 200 vrhova).



Slika 4.6. Graf velikog problema.

Prosjek rezultata mjerenja prikazan je na slici 4.7. Rezultati ovog mjerenja slični su prethodnim rezultatima mjerenja za GCP problem. Simulirano i genetsko kaljenje u roku od 10 minuta, koliko su trajala mjerenja, pronašla su najbolja rješenje s time da je simulirano kaljenje do tog rješenja došlo znatno brže. To pokazuje da je lokalno pretraživanje s povoljnim parametrima mutacije vremenski brže pronašlo rješenje od globalnog pretraživanja. S druge strane, vremenski svaka iteracija genetskog kaljenja zbog vrijednosti parametra N, odgovara otprilike 31 iteraciji simuliranog kaljenja što znači da je globalno pretraživanje u manje pokušaja došlo do istih rezultata kao i lokalno pretraživanje.

PSO i RPSO su se pokazali još manje uspješnima nego na prethodna dva problema. Njihovo rješenje tek je malo uznapredovalo u odnosu na početno rješenje. Ako je u prethodnim problemima uzrok tome bila dimenzionalnost prostora, tada je taj uzrok na ovome problemu još izraženiji.



Slika 4.7. Rezultati mjerenja optimizacije velikog problema

5. **Zaključak**

Mnogi praktični problemi kategorizirani su kao NP teški problemi i nisu rješivi egzaktnim postupcima. Problemi podijele i bojanja grafa su upravo takvi problemi. Za rješavanje takvih problema razvijeni su brojni algoritmi koji na pametan način pokušavaju pogoditi rješenje, tzv. heuristički algoritmi. Među četiri ispitana algoritma (simulirano kaljenje, genetsko kaljenje, optimizacija rojem čestica i optimizacija odbijanjem roja čestica) genetsko kaljenje se pokazalo kao najuspješnije u pronalasku rješenja problema podjele grafa dok je u rješavanja bojanja grafa prednjačilo simulirano kaljenje. Osnovne varijante optimizacije rojem čestica i optimizacije odbijanjem roja čestica su se pokazale manje uspješnima na tim problemima.

6. **Literatura**

[1] Dr. Dobb's Algorithm Alley, october 1st 1994,
http://www.ddj.com/architect/184409333?pgno=10, 3. 6. 2009.
[2] Evolutionary algorithm
http://en.wikipedia.org/wiki/Evolutionary_algorithm/, 3. 6. 2009.
[3] Simulated annealing
http://en.wikipedia.org/wiki/Simulated_annealing, 3. 6. 2009.
[4] NP-complete
http://en.wikipedia.org/wiki/NP-complete/, 3. 6. 2009.
[5] Optimizacija rojem čestica
http://www.zemris.fer.hr/~golub/ga/studenti/projekt2008/pso/, 3. 6. 2009.
[6] Optimizacija rojem čestica (odbijanjem)
http://www.zemris.fer.hr/~golub/ga/studenti/projekt2008/rpso/, 3. 6. 2009.

7. **Sažetak**

Mnogi praktični problemi kao npr. problem bojanja grafa, kategorizirani su kao NP teški problemi i nisu rješivi egzaktnim postupcima. Za rješavanje takvih problema razvijeni su brojni tzv. heuristički algoritmi koji na pametan način pokušavaju pogoditi rješenje. U ovome radu ispitana je uspješnost simuliranog kaljenja, genetskog kaljenja, optimizacije rojem čestica i optimizacije odbijanjem roja čestica u rješavanju problema bojanja grafa i problema podjele grafa.