SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt

Genetsko kaljenje

Ivan Kravarščan

Voditelj: *doc. dr. sc. Marin Golub*

Zagreb, prosinac, 2008.

Sadržaj

1. Uvod

Računala su široko primjenjiva u automatizaciji i računanju ali i ona imaju ograničenja svojih mogućnosti. Jedno od takvih ograničenja jest složenost algoritama s kojima mogu raditi u realnom vremenu. Elektronička računala u većini slučajeva ne mogu potpuno riješiti NP težak problem ali uz pomoć određenih algoritama mogu pronaći skoro 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.

Mnogi algoritmi nadopunjuju pohlepni algoritam tehnikama bijega iz lokalnog minimuma a među takvim algoritmima je simulirano kaljenje. Simulirano kaljenje se pak nadograđuje paralelnim ispitivanjem više rješenja u genetsko kaljenje, kako bi se još uspješnije pronašlo optimalno rješenje.

2. Genetsko kaljenje

2.1 Evolucijski algoritmi općenito

Teorija računarstva, po složenosti, dijeli probleme u dvije velike skupine: P i NP. P problemi su oni problemi čije se riješenje 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.

NP problemi su formalno skup svih problema, 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) tako da 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. Pošto 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 slijedeće svojstvo većine NP teških problema: valjanost rješenja može provjeriti determinističkim algoritmom polinomne složenosti. Stoga većina tih algoritama NP probleme rješavaju pogađanjem rješenja, tzv. metaheurističkim postupkom.

Dio takvih algoritama spada u evolucijske algoritme, algoritme koji oponašaju evolucijske procese iz prirode. Ti algoritmi rade po principu da pretpostave neki skup rješenja pa ponavljaju postupak provjere koliko su ta rješenja dobra i postupak generiranje novih rješenja, oponašanjem evolucijskih procesa. Takav stohastički pristup ne garantira da će svako izvršavanje algoritma proizvesti isto riješenje niti da će to riješenje biti blizu optimalnog ali praksa pokazuje da ti algoritmi s dobrom vjerojatnošću daju dovoljno dobra rješenja. Među evolucijskim algoritmima je i genetsko kaljenje.

2.2 Simulirano kaljenje

Kaljenje je postupak polaganog hlađenja vruće mase (npr. stakla) 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.

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′*. Ako je stanje *s′* stanje više energije, *s* prelazi u *s′* uz vjerojatnost koja ovisi o temperaturi.

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. Time se nastoji izbjeći loša karakteristika pohlepnog algoritma: zaustavljanje u lokalnom minimumu koji je puno lošiji od globalnog minimuma.

2.2.1 Pseudokod

```
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
```

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 stanja više energije. 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 jednostavniji 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 *E*/*N* (*N* je broj čestica) veću energiju od početne čestice.

2.3.1 Pseudokod

```
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)
```

2.3.2 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. Ovisno o pogledu na problem, može se definirati da bolja čestica ima veću vrijednost 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, koliki udio slobodne energije se zadržava u sustavu). 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 drugi sporije, kod nekih trećih pak intenzitet mutacije mora biti mali, itd.

U posebnim slučajevima 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 se radi samo s jednom česticom i kada se nepovoljna stanja nikad ne prihvaćaju ($C = 0$ i $N = 1$).

3. Primjena

Genetsko kaljenje se može primjenjivati u rješavanju velikog broja NP teških problema. Svoju široku primjenjivost zahvaljuje svojom jednostavnošću zahtjeva. Praktički svi heuristički i metaheuristički algoritmi zahtijevaju da se za dani problem definira kako će se kodirati riješenje (struktura čestice) te da se definira funkcija dobrote rješenja. Genetsko kaljenje uz te dvije definicije još zahtjeva definiciju funkcije mutacije a ona se često može lako izvesti iz strukture čestice.

U slijedećim poglavljima su opisani problem trgovačkog putnika i problem podijele grafa. Oba problema su statički problemi (problem se ne mijenja tokom vremena) no i dinamički problemi, kao što je problem mrežnog usmjeravanja, rješivi su genetskim kaljenjem.

3.1 Problem trgovačkog putnika

Problem trgovačkog putnika (engl. *travelling salesman problem* u daljnjem u TSP) je problem pronalaska najkraćeg puta tako da se obiđu svi unaprijed zadani gradovi. Matematički, taj se problem svodi na problem pronalaska najkraćeg Hamiltonovskog puta u težinskom grafu. Graf se načini tako da vrhovi predstavljaju gradove, bridovi veze između gradova a težina brida cijenu puta između povezanih gradova. Put koji je riješenje takvog problema može se definirati kao uređena n-torka (n = |V|, broj vrhova) koja predstavlja raspored kojim se gradovi/vrhovi posjećuju.

Egzaktno rješavanje problema bi zahtjevalo da se ispitaju sve moguće rute, odnosno sve permutacije rasporeda (n-torke) a složenost takvog postupka je $O(n!)$. Pomoću tzv. dinamičkog rješavanja, ta složenost se može smanjiti na $O(n^2 2^n)$ odnosno na eksponencijalnu složenost. Oba pristupa bi na suvremenim računalima već za 45 gradova zahtijevala preko godine dana računanja.

3.2 Problem dijeljenja grafa

Problem dijeljenja grafa (engl. *graph partitioning problem*, u daljnjem u 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.

Egzaktno rješavanje i ovog problema bi u općem slučaju zahtjevalo ispitivanje svih kombinacija. Kako je ukupan broj tih kombinacija m^n algoritam koji ispituje te kombinacije bi imao eksponencijalnu složenost.

4. Programsko ostvarenje

U sklopu projekta, napravljeno je programsko ostvarenje genetskog kaljenja u programskom jeziku Java. U programu se mogu rješavati (i pratiti tok rješavanja) osnovnih oblika TSP-a i GPP-a koristeći jedan od tri metaheurističkih algoritama: pohlepni algoritam, simulirano kaljenje i genetsko kaljenje.

TSP je pojednostavljen tako da su svi gradovi međusobno povezani, veze između gradova su dvosmjerne i cijena puta između dva grada je jednaka njihovoj međusobnoj udaljenosti. Drugim riječima graf TSP problema je potpuno povezani težinski graf. GPP je pojednostavljen tako je parametar *m* jednak 2.

Program je primarno namijenjen za rješavanje pomoću genetskog kaljenja ali ostali algoritmi su dodani zbog mogućnosti međusobne usporedbe njihovih performansi.

4.1 Ulaz

Ulazni podatci se mogu unijeti pomoću grafičkog sučelja odabirom unaprijed definiranih problema ili učitati iz datoteke (datoteka mora imati naziv ulaz.txt i mora biti u radnom direktoriju aplikacije). Ulazna datoteke je tekstualna datoteka slijedećeg oblika:

- Redovi koji počinju sa znakom `#` su komentari i njih program preskače
- Redovi oblika `!param x` postavljaju parametre algoritma (parametar `param` se postavlja na vrijednost `x`). Program prihvaća parametre GAN_N, GAN_C, GAN_EX i GAN_PR a ostale zanemaruje.
- Ostali redovi definiraju problem i njihov oblik ovisi o postavljenom problemu. Prvi red u definiciji problema je kodna oznaka problema. Kod `TSP` označava problem trgovačkog putnika (engl. *travelling salesman problem*) a kod `GPP` problem podjele grafa (engl. *graph partitioning problem*).

U koliko je riječ o problemu trgovačkog putnika, preostali redovi su oblika `x y` gdje su `x` i `y` realni brojevi između 0 i 1 i predstavljaju poziciju grada. Kod problemima dijeljenja grafa, nakon kodne oznake očekuju se redovi slijedećeg oblika i značenja:

```
m           - broj skupova u koje se vrhovi raspoređuju
n           - broj vrhova u grafu
vx1 vy1
...
vxi vyi     - koordinate i-tog grada
...
vxn vyn
e1a e1b
```

...

`eia eib` - brid e_i veže gradove v_{eia} i v_{eib}

...

`ena enb`

U datoteci je moguće zadati da je broj skupova za GPP veći od 2, ali program može raditi samo sa problemom u kojem je broj skupova jednak 2.

4.2 Izlaz

Izlazi iz programa su grafički prikaz trenutnog rješenja, graf evolucije i datoteka s podacima grafa evolucije. Podatci iz datoteke se mogu grafički prikazati pomoću *gnuplot* alata.

4.3 Realizacija algoritma

Dio programa koji direktno vrši optimizaciju zasniva se na dva sučelja: `IMetaHeuristicAlgorithm` i `IConfiguration` i njihovim implementacijama. `IMetaHeuristicAlgorithm` služi kako jedinstven pristup grafičkog sučelja algoritmima. Svaki algoritam kao strukture rješenja (čestice) prima listu objekata iz razreda koji implementiraju sučelje `IConfiguration`.

Definicija sučelja IMetaHeuristicAlgorithm

```
public interface IMetaHeuristicAlgorithm {
        public IConfiguration nextIteration();
        public IConfiguration getBestConfiguration();
        public String getName();
}
```

Definicija sučelja IConfiguration

```
public interface IConfiguration {
        public double getEnergy();
        public IConfiguration generateMutant(
                int mutationIntensity,
                double pr);
}
```

Tablica 4.1: Definicije sučelja IMetaHeuristicAlgorithm i IConfiguration

Načinjena su dva razreda koji implementiraju sučelje `IConfiguration`: `ConfigurationTSP` za TSP problem i `ConfigurationGPP` za GPP problem. Oba razreda unutar sebe imaju, prema zahtjevu sučelja, ostvarenu funkciju energije i mutacije te konstruktor koji generira nasumično rješenje.

U `ConfigurationTSP` rješenje je kodirano u listu (*array list*) cijelih brojeva *L* i ta lista predstavlja redoslijed obilaska gradova, energija rješenja je jednaka sumi težina bridova a mutacija je izvedena kombinacijom dviju strategija:

1. Zamjena dvaju nasumičnih elemenata liste *L*, osnovna strategija.
2. Obrtanje elemenata u podnizu liste L, alternativna strategija.

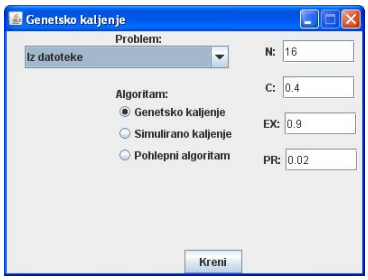
Parametar `pr` funkcije `generateMutant` određuje vjerojatnost da se primjeni alternativna strategija a parametar `mutationIntensity` intenzitet mutacije. U osnovnoj strategiji `mutationIntensity` određuje koliko puta treba izvršiti zamjenu parova a u alternativnoj, kolika je duljina podniza.

`ConfigurationGPP` rješenje kodira u dva skupa (*hash set*) vrhova koji direktno predstavljaju način podjele grafa. Energija rješenja je broj bridova koji povezuju vrhove iz različitih skupova. Mutacije u ovom razredu vrše se tako da se odaberu parovi vrhova (jedan član para iz jednog skupa, drugi iz drugog) i zamjene skupovi kojima oni pripadaju. Parametar `mutationIntensity` određuje broj parova a parametar `pr` je zanemaren.

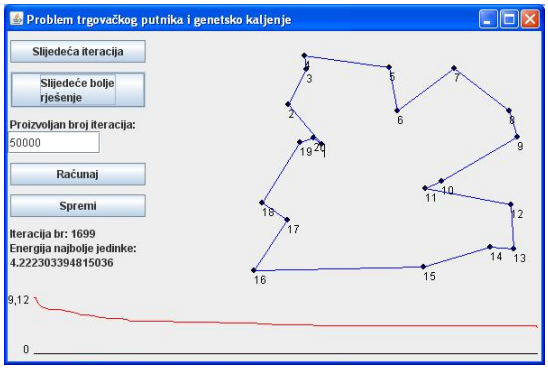
Genetsko kaljenje se vrši prema pseudokodu opisanom u poglavlju 2.3.1 uz promjenu da se prije mutacije računa intenzitet mutacije (`mutationIntensity` parametar metode `generateMutant`). Intenzitet je prirodan broj čija se vrijednost podvrgava geometrijskoj razdiobi s parametrom *q*. Iz svega navedenog, parametri algoritma su slijedeći:

- *N* – broj čestica
- *C* – koeficijent hlađenja
- *EX* – parametar *q* geometrijske razdiobe
- *PR* – vjerojatnost primjene alternativne strategije mutacije

4.4 Grafičko sučelje



Slika 4.1: Prozor za odabir optimizacijskog problema i podešavanje algoritma



Slika 4.2: Prozor za prikaz rada algoritma

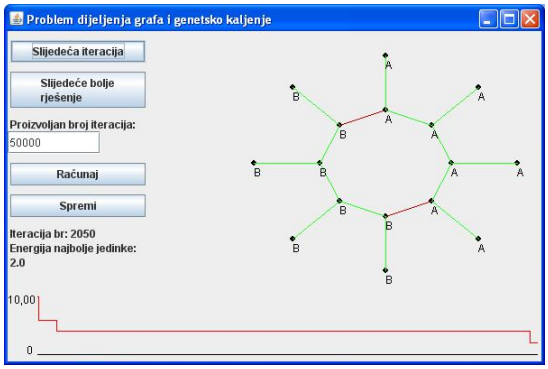
Grafičko sučelje je podijeljeno na dva prozora. Prvi prozor (slika 4.1), nazovimo ga "pokretač", omogućava odabir optimizacijskog problema, odabir algoritma te unos parametara. Drugi prozor (slika 4.2), nazovimo ga "pogled", omogućava kontrolu rada algoritma, prikazuje trenutno najbolje rješenje i prikazuje graf evolucije.

Kada se aplikacija pokrene, otvara se prozor "pokretač". Nakon odabira problema, algoritma i parametara, pritiskom na gumb "Kreni", otvara se prozor "pogled" u kojem se rješava odabrani problem. Omogućeno je da se u "pokretaču" pokrene više prozora za rješavanje problema. "Pogledi" su međusobno neovisni i mogu rješavati međusobno različite probleme te raditi različitim algoritmima s različitim parametrima.

Upravljanje radom algoritma vrši se pomoću tri gumba:

- "Sljedeća iteracija" – algoritam napreduje za jednu iteraciju
- "Sljedeće bolje rješenje" – algoritam računa sve dok ne pronađe bolje riješenje (u odnosu na ono koje je bilo prije pritiska na gumb) ili dok se ne prekorači vremensko ograničenje od 0.25 s.
- "Računaj" – algoritam napreduje za broj iteracija koji je zadan u polju za unos teksta

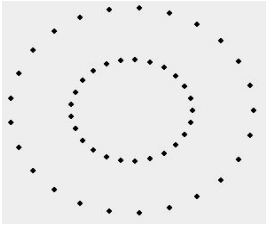
Informacije koje "pogledi" pružaju su grafički prikaz rješenja, graf evolucije, broj trenutne iteracije i energija najboljeg rješenja u trenutnoj iteraciji. Grafički prikaz rješenja ovisi o problemu. Za TSP to je graf na kojem su bridovi koji čine rutu trenutnog rješenja obojani plavo, bridovi koji ne čine rutu nisu prikazani a vrhovi su označeni brojevima koji predstavljaju redoslijed obilaska (slika 4.2). Kod GPP-a taj prikaz je isto graf ali sa sljedećim oznakama: vrhovi su označeni sa slovima "A" i "B" (vrhovi dodijeljeni istom skupu označeni su istim slovom), bridovi koji povezuju vrhove dodijeljene različitim skupovima su crvene boje a ostali bridovi obojani su zelenom bojom (slika 4.3). Graf evolucije prikazuje energije najboljih rješenja u prethodnim iteracijama.



Slika 4.3: Prikaz rješavanja GPP problema

4.5 Mjerenja

Kao što je opisano, genetsko kaljenje je osmišljeno kako nadogradnja simuliranog kaljenja a simulirano kaljenje kao nadogradnja pohlepnog algoritma. Mjerenjem je ispitano da li su nadogradnje zbilja poboljšale sposobnost optimizacije. Problem koji se u pokusima rješavao bio je TSP problem u kojem je 50 gradova raspoređeno dva prstena, oba sa 25 gradova, kao na slici 4.4. Procijenjena energija optimalnog rješenja je 20.19. Svi pokusi su se vršili sa slijedećim parametrima: N = 8, C = 0.4, EX = 0.9 i PR = 0.02. Genetskom kaljenju dano je N puta manje iteracija jer ono u jednoj iteraciji obavi N operacija nad rješenjima.



Slika 4.4: 50 gradova raspoređeno u dva prstena

Rezultati pokusa prikazani su u tablici 4.2. Prema tim rezultatima možemo zaključiti da simulirano kaljenje pronalazi bolja rješenja od pohlepnog algoritma te da genetsko kaljenje, nakon većeg broja iteracija, pronalazi još bolja.

R. br. pokusa	Pohlepni algoritam, 80000 iteracija	Simulirano kaljenje, 80000 iteracija	Genetsko kaljenje, 10000 iteracija	Pohlepni algoritam, 800000 iteracija	Simulirano kaljenje, 800000 iteracija	Genetsko kaljenje, 100000 iteracija
1	28.42	27.54	33.24	27.90	27.24	26.79
2	30.91	29.19	31.48	28.45	28.59	25.96
3	32.85	27.00	32.20	29.70	26.34	29.88
4	32.03	35.71	32.16	28.05	27.74	27.15
5	35.30	29.96	34.71	30.85	28.73	28.32

Table 4.2: Energije dobivenih rješenja nakon 80000 i nakon 800000 operacija nad rješenjima

5. Zaključak

Mnogi NP teški problemi nisu rješivi egzaktnim postupcima i zbog toga su razvijeni brojni algoritmi koji na pametan način pokušavaju pogoditi rješenje. Jedan od jednostavnijih je genetsko kaljenje. Zbog svoje jednostavnosti ono možda ima lošije performanse do složenijih algoritma ali zato se može primijeniti na velikom broju problema. U usporedbi s algoritmima iz kojeg se razvilo (pohlepni algoritam i simulirano kaljenje), genetsko kaljenje za isti broj operacija pronalazi bolje rješenje.

6. Literatura

[1] Dr. Dobb's Algorithm Alley, october 1st 1994, http://www.ddj.com/architect/184409333?pgno=10, 17. 12. 2008.

[2] Evolutionary algorithm http://en.wikipedia.org/wiki/Evolutionary_algorithm, 17. 12. 2008.

[3] Simulated annealing http://en.wikipedia.org/wiki/Simulated_annealing, 17. 12. 2008.

[4] NP-complete http://en.wikipedia.org/wiki/NP-complete,17. 12. 2008.

7. Sažetak

Za rješavanje NP teških problema postoji mnogo algoritama a među njima je genetsko kaljenje. Algoritam genetskog kaljenja objavio je Kennet V. Price 1994. godine u časopisu Dobb's Journal kao proširenje simuliranog kaljenja. Simulirano kaljenje radi po principu oponašanja čestice u procesu kaljenja a genetsko kaljenje proširuje taj postupak na skup čestica.

Algoritam genetskog kaljenja opisan je u ovome radu. Potom je unutar programskog ostvarenja napravljena usporedba s simuliranim kaljenjem i pohlepnim algoritmom.