SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT

Optimizacija rojem čestica (odbijanjem)

Zvonimir Fras

Voditelj: *Marin Golub*

Zagreb, studeni, 2008

Sadržaj

1. Uvod

Inspiracija za optimizaciju rojem čestica dolazi od činjenice da je učenje društvena aktivnost. Ljudi uče jedni od drugih ne samo činjenice već i načine kako rukovati tim činjenicama.

Ne samo da ljudi uče jedni od drugih, već kako se znanja i vještine šire od čovjeka do čovjeka, tako cijela populacija konvergira optimalnim rješenjima. Pračovjek je u davno doba uzeo oštar kamen i probo kožu neke životinje kako bi došao do mesa koje je želio pojesti. Drugi su ga vidjeli kako to radi i preuzeli to od njega, tražeći druge oštre predmete kako bi napravili isto. Danas imamo kirurške noževe velike oštrine i preciznosti.

Jedinke uče od drugih u svojoj blizini. Ljudi komuniciraju s drugim ljudima, sakupljajući informacije i zauzvrat daju informacije koje su njima poznate. Na taj način širenje znanja postaje grupni proces. Društvo je samo-organizirajući sustav koji posjeduje neka svojstva koja se ne mogu predvidjeti iz svojstva jedinki koje čine to društvo.

Obrazovanje optimira spoznaju. Iako se svaka interakcija odvija lokalno, informacije i inovacije se obrazovanjem prenose do udaljenih jedinki. Primjerice ja Vama kažem neku novost, Vi to kažete nekom svom prijatelju kojeg ja ne poznajem i ne mogu mu to reći, on to kaže nekom svom prijatelju... Informacija se širi. Nadalje, kombinacije u različitim metodama dovode do još naprednijih metoda. Ovaj globalni efekt je nevidljiv članovima društva koji od njega imaju koristi.

U metodama optimizacije rojem čestica koristit će se upravo činjenica da skup jedinki ima svojstvo da optimizira različite probleme, pri čemu jedinke koje sačinjavaju taj skup ne moraju znati ništa o problemu kojeg optimiziraju, nego rade svoje jednostavne operacije.

U ovom dokumentu bit će objašnjen općeniti problem optimizacije. Opisat će se primjena optimizacije rojem čestica na probleme optimizacije. Nakon toga opisat će se varijanta optimizacije rojem čestica nazvana optimizacija rojem čestica odbijanjem. Bit će dani primjeri korištenja optimizacije rojem čestica, te uspoređen klasični algoritam optimizacije rojem čestica s algoritmom optimizacije rojem čestica odbijanjem. Oba algoritma bit će ispitana na 5 optimizacijskih problema te rezultati predočeni u tablicama i grafički.

2. Optimizacija

2.1 Općenito o optimizaciji

Optimizacijski problemi su od izuzetne važnosti u industrijskom i znanstvenom svijetu. Primjeri praktičnih problema optimizacije uključuju bilo kakvo planiranje i izrada rasporeda (npr. vozni redovi vlakova), organizacija topologije komunikacijskih mreža i drugi.

Općenito, bilo koji optimizacijski problem *P* može se prikazati trojkom (*S*, *Ω*, *f*), gdje je

1. *S* prostor pretraživanja definiran konačnim skupom varijabli X_i, $i = 1,...,n$
2. *Ω* je skup ograničenja među varijablama
3. $f : S \rightarrow \mathbb{R}^+$ funkcija koja dodjeljuje pozitivnu vrijednost svakom elementu (ili rješenju) iz *S*

U slučaju kad varijable X_i imaju diskretnu domenu, radi se o diskretnoj ili kombinatoričkoj optimizaciji dok se u slučaju da imaju kontinuiranu domenu, radi o kontinuiranim optimizacijskim problemima. Mogući su i problemi koji imaju kombinaciju jednog i drugog.

Cilj optimizacije je pronaći rješenje $s \in S$ takvo da vrijedi $f(s) \leq f(s'), \forall s' \in S$ ukoliko želimo minimizirati funkciju, ili $f(s) \geq f(s'), \forall s' \in S$ ukoliko želimo maksimizirati funkciju [1]. U stvarnom životu često ćemo morati optimizirati više funkcija istovremeno.

Zbog velike praktične važnosti razvijeni su mnogi algoritmi koji rješavaju optimizacijske probleme. U kontekstu kombinatorne optimizacije (CO, *engl. combinatorial optimization*) razlikujemo potpune i aproksimacijske algoritme. Potpuni algoritmi garantiraju pronalaženje optimalnog rješenja u konačnom vremenu. Za probleme koji su NP-teški, nema potpunog algoritma koji daje rješenje u polinominalnom vremenu, u najgorem slučaju vrijeme eksponencijalno raste. To obično nije prihvatljivo za praktična rješenja pa se pribjegava aproksimaciji rješenja. Kod takvih metoda, kao što je optimizacija rojem čestica, žrtvujemo traženje optimalnih rješenja za traženje dovoljno dobrih rješenja u značajno smanjenom vremenu.

2.2 Optimizacija rojem čestica

Optimizacija rojem čestica je stohastička optimizacijska metoda modelirana prema socijalnom ponašanju promatranih životinja i kukaca, primjerice jata ptica, stada životinja, roj mušica itd. Od početka privlači pažnju istraživača kao robusna i efikasna tehnika za rješavanje složenih optimizacijskih problema.

U PSO (*engl. Particle Swarm Optimization – Optimizacija rojem čestica*) svaka čestica roja predstavlja potencijalno rješenje i kreće se kroz prostor problema u potrazi za optimalnim ili dovoljno dobrim rješenjem. Čestice razašilju svoj trenutni položaj susjednim česticama. Pozicija svake čestice mijenja se sukladno brzini gibanja čestice, svojoj najboljoj poziciji i najboljoj poziciji njoj susjednih čestica. Kako model napreduje, roj se sve više usredotočuje na područje koje sadrži visokokvalitetna rješenja.

Izvorni algoritam sastoji se od dvije jednadžbe prema kojima se u svakom koraku osvježuju brzina i pozicija:

$$v_i \leftarrow \omega v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (p_g - x_i) \tag{2.1}$$

$$x_i \leftarrow x_i + v_i \tag{2.2}$$

gdje je:

v_i – brzina *i*te čestice

x_i – pozicija *i*te čestice

ω – inercijska konstanta, dobre vrijednosti su obično malo manje od 1

p_i – najbolji osobni rezultat (*engl. personal best*)

p_g – najbolji rezultat susjeda

r_1, r_2 - dvije odvojene funkcije od kojih svaka vraća vektor slučajnih vrijednosti u intervalu [0,1]

c_1, c_2 - koeficijenti ubrzanja, govore koliko je dobro čestica usmjerena prema dobrim pozicijama. Dobre vrijednosti su obično oko 1.

Jednadžba (2.1) pokazuje da brzina čestice ovisi o tri dijela: inerciji, osobnom dijelu i društvenom dijelu. Inerciju predstavlja brzina iz prethodnog koraka kojom se kretala čestica do tog trenutka. Osobni dio $c_1 r_1 (p_i - x_i)$, predstavlja težnju čestice da se vrati na najbolje mjesto na kojem je bila dok društveni dio $c_2 r_2 (p_g - x_i)$, predstavlja težnju čestice da ode na najbolje mjesto koje je našao čitav roj.

Na osnovu te dvije jednadžbe i dosad rečenog možemo napisati osnovni algoritam za optimizaciju rojem čestica. Algoritam prikazan na slici 1.

```
nasumično generiraj početni roj
repeat
   for svaku česticu i do
      if f(x_i) > f(p_i) then p_i ← x_i
      p_g=max(p_susjedi)
      izračunaj brzinu prema jednadžbi (2.1)
      izračunaj poziciju prema jednadžbi (2.2)
   end for
until zadovoljen traženi kriterij
```

Slika 1: Osnovni algoritam optimizacije rojem čestica (PSO)

Promatranjem je zaključeno da PSO teži preranom konvergiranju u lokalni optimum. Kako bi riješili taj problem, razvijene su različite izvedbe PSO. Između ostalih i RPSO, o kojem će biti više riječi u slijedećem poglavlju, disipacijski PSO koji povećava nasumičnost, FDR-PSO koji potiče komunikaciju među bliskim česticama i ostali.

2.3 Optimizacija rojem čestica odbijanjem

Optimizacija rojem čestica odbijanjem, RPSO (*engl. Repulsive Particle Swarm Optimization*), je varijanta klasičnog PSO. Posebno je efikasna u traženju globalnih optimuma izrazito složenih prostora pretraživanja iako može biti sporija kod određenih vrsta optimizacijskih problema[2].

Kod tradicionalnog RPSO, formule su nešto drugačije nego kod PSO:

$$v_i \leftarrow \omega v_i + \alpha r_1 (p_i - x_i) + \omega \beta r_2 (p_h - x_1) + \omega \gamma r_3 z \tag{2.3}$$

$$x_i \leftarrow x_i + v_i \tag{2.4}$$

gdje je:

v_i – brzina *i*te čestice

x_i – pozicija *i*te čestice

ω – inercijska konstanta, [0.01,0.7]

p_i – najbolji osobni rezultat (*engl. personal best*)

p_h – najbolji rezultat slučajno izabrane čestice iz roja

r_1, r_2, r_3 – slučajni brojevi u intervalu [0,1]

z – slučajni vektor brzine

α, β, γ – konstante

Povremeno, ukoliko proces zapne u nekom lokalnom optimumu, može biti potrebno kaotično uznemirenje pozicije i brzine nekih čestica.

Algoritam tradicionalnog RPSO se od tradicionalnog PSO razlikuje samo u korištenim formulama za osvježavanje brzine i položaja.

Na tradicionalan algoritam su dodana određena poboljšanja kako bi dobili na performansama i preciznosti. Kod tradicionalnog RPSO čestice ne poklanjaju previše pažnje pretraživanju svoje okoline. Vođene su svojim iskustvom i komunikacijom s drugim česticama roja. Stoga je tradicionalan RPSO modificiran na taj način da svaka čestica bolje pretražuje svoju okolinu[2]. Ta pretraga je kontrolirana novim parametrom, *nstep*. Taj dodatak približava pretraživačke sposobnosti RPSO onima koje možemo vidjeti u stvarnosti.

Svaka čestica uči od odabranih kolega u roju. U tradicionalnom PSO, čestice su učile od najboljeg u roju. Međutim takvo učenje nije prirodno, ne možemo očekivati od individualne čestice da zna koja čestica je najbolja niti možemo očekivati od najbolje čestice da komunicira sa svakom česticom u roju. U roju u kojem je interakcija i pojedinačno znanje čestice ograničeno, svaka čestica može naučiti nešto od druge čestice i naučiti nešto drugu

česticu, tako da čestice ne znaju najbolju česticu u roju[2]. To nas dovodi do problema: kako izabrati s kim će koja čestica komunicirati? Jedna od mogućnosti koja se prirodno nalaže je da čestica komunicira s onim česticama koje su joj najbliže. Međutim, ako uzmemo u obzir prethodno navedeno poboljšanje (naša čestica sama istražuje svoju okolinu) jasno je da čestica neće imati previše koristi od komunikacije s česticama u svojoj okolini. Puno veću korist imala bi od komunikacije s nasumično odabranim česticama iz roja. Inspiracija za to dolazi iz ljudskog svijeta gdje se to svakodnevno događa. Na raznim mjestima, primjerice u vozilima javnog gradskog prijevoza, na koncertima, u crkvi i sl. susrećemo ljude koji dolaze iz različitih mjesta i razmjenjujemo potrebne informacije. Tu geografska udaljenost nije bitna, bitno je kako tuđa iskustva dođu do nas. Na taj način informacije koje dobivamo i dajemo dolaze iz praktički nasumičnih izvora. Kad to preslikamo u RPSO, svakoj čestici dodjeljujemo fiksan broj nasumično odabranih kolega s kojima komunicira. Kao varijacija, možemo taj fiksan broj također odabrati nasumično između gornje i donje granice.

Na taj način modificiran algoritam RPSO djeluje prirodnije i daje bolje rezultate.

nasumično generiraj početni roj
repeat
  for svaku česticu *i* **do**
    if $f(x_i) > f(p_i)$ **then** $p_i \leftarrow x_i$
    $p_g = \max(p_{susjedi})$
    izračunaj brzinu prema jednadžbi (2.3)
    izračunaj poziciju prema jednadžbi (2.4)
  end for
until zadovoljen traženi kriterij

Slika 2: Osnovni algoritam optimizacije rojem čestica odbijanjem (RPSO)

Vidimo da se od osnovnog PSO algoritma razlikuje samo u koracima u kojima se izračunava brzina i pozicija čestica.

3. Izvedba

3.1 Uvod

Cilj praktičnog dijela ovog projekta je realizirati osnovne PSO i RPSO algoritme i ispitati njihovu efikasnost na problemima optimizacije realnih funkcija.

Realne funkcije korištene u svrhu ispitivanja su:

- Schaffer F6
- De Jongs' Sphere
- Rosenbrock
- Rastrigin
- Griewangk

Sve funkcije su dobro poznati teški problemi na kojima se često ispituju optimizacijske metode i kao takve su dobar izbor za ispitivanje u ovom slučaju.

3.2 Razrada

Zbog želje za podržavanjem više platformi (različitih arhitektura i operacijskih sustava) korišteni programski jezik je C++ zbog svoje raširenosti i jednostavnosti prebacivanja s arhitekture na arhitekturu jednostavnim ponovnim prevođenjem programskog koda.

3.2.1 Ulaz

Program se pokreće iz komandne linije (*shell* na unix-like operacijskim sustavima, *command prompt* na windows-like operacijskim sustavima) te prima tri parametra:

- datoteku s opisom problema
- algoritam koji se koristi (pso ili rpso)
- prefiks za izlaznu datoteku

Datoteka s opisom problema treba pratiti pravila zapisa problema definirana programom.

Svi problemi se zapisuju jednom linijom. Linija počinje kodnom oznakom problema (realne funkcije) koji se optimizira, nakon čega slijedi popis parametara koji se žele koristiti pri optimizaciji. U tablici 1 se nalaze parovi kod-ime.

Popis parametara mora biti u sljedećem nizu:

broj dimenzija, broj čestica, maksimalna koordinata, maksimalna brzina, maksimalan broj iteracija, cutoff, lijevi inicijalizacijski rang, desni inicijalizacijski rang, inercijska konstanta, tip inercijske konstante, broj ponavljanja.

U ulaznoj datoteci je moguće postavljati i komentare kako bi korisnicima omogućili primjerice opisivanje problema koji je definiran parametrima. Oznaka komentara je znak '#'

na početku linije. Na taj način se programu kaže da se radi o komentaru i da ne pokušava optimizirati problem definiran tom linijom.

Osim uobičajenog načina rada, u program je ugrađena i podrška za automatsko optimiziranje više problema jedan za drugim, bez dodatne interakcije s korisnikom. To je moguće na jednostavan način tako da se u ulaznu datoteku u više linije stave podaci za više problema (npr. 5 linija za 5 problema).

Kodna oznaka problema	Ime problema
RB	Rosenbrock
SF6	Schaffer F6
GW	Griewangk
RAS	Rastrigin
DJS	De Jongs' Sphere

Tablica 1: Parovi kod problema - ime problema

3.2.2 Izlaz

Rezultati optimizacije se spremaju u datoteke u formatu prilagođenom za čitanje s *gnuplot*om. Ime datoteke se generira pomoću zadanog prefiksa, koda problema, *underscore*a i rednog broja tog problema u datoteci. Tako prefiks "asdf", problem s kodom "RB" koji je prvi po redu, dobije ime izlazne datoteke "asdfRB_1". Takvu datoteku moguće je iskoristiti u *gnuplot*u za iscrtavanje grafa ili za obradu na neki drugi način. Osim toga, za svaki problem se generira .gnu datoteka kako bi se olakšalo iscrtavanje grafa onima koji nisu upoznati s radom u *gnuplot*u. U prethodnom primjeru, ta datoteka zvala bi se "asdfRB_1.gnu". Navedenu datoteku potrebno je samo otvoriti s *gnuplot*om.

3.2.3 Struktura i tok programa

Jezgra programa je razred *ParticleSwarmOptimizer*. On čuva podatke o svim česticama u roju, zadužen je za postavljanje parametara problema koji se optimizira, nasumično razmještanje čestica u roju, te sadrži predefinirane, već navedene, funkcije koje može optimizirati i jednu korisnički definiranu virtualnu funkciju, koju je moguće implementirati u razredu koji ga nasljeđuje. Osim toga, budući da ima pristup svim podacima, ima implementirane metode za pohranu rezultata u formatu prilagođenom za čitanje s *gnuplot*om, te za generiranje .gnu datoteka. Za samo optimiziranje zadužena je prava virtualna (*engl. pure virtual*) funkcija *Optimize*. Budući da prava virtualna funkcija nema implementaciju, potrebno ju je napraviti u razredu koji nasljeđuje razred *ParticleSwarmOptimizer*.

Svaka čestica je primjerak razreda *PParticle* i olakšava razredu *ParticleSwarmOptimizer* manipulaciju rojem budući da zna sve stvari koje jedna čestica mora znati, broj dimenzija

kroz koje se kreće, svoju poziciju u prostoru, svoju brzinu, svoj najbolji rezultat, te poziciju na kojoj je ostvaren taj rezultat.

Zahvaljujući načinu na koji je razred *ParticleSwarmOptimizer* osmišljen, moguće je nasljeđivanjem, te implementiranjem funkcije *Optimize* stvoriti razred koji predstavlja "bilo koju" metodu optimizacije rojem čestica, pritom zadržavajući sve blagodati koje razred pruža.

Na taj način su napravljeni razred *PSOptimizer*, koji implementiranjem funkcije *Optimize* utjelovljuje klasični PSO, te razred *RPSOptimizer* koji na isti način utjelovljuje klasični RPSO.

Glavni tok programa se oslanja na svoju jezgru. Na osnovu parametara komandne linije radi primjerak razreda *PSOptimizer* ili *RPSOptimizer*, dok pritom koristi pokazivač na *ParticleSwarmOptimizer* i tu prestaje briga o korištenom algoritmu. Posao preuzima *parser* ulazne datoteke koji funkcijama *ParticleSwarmOptimizer*a postavlja parametre problema. Nakon toga poziva se funkcija *Optimize*, te se nakon njenog uspješnog izvršavanja generiraju imena za izlaznu datoteku i pohranjuju rezultati.

3.3 Ispitivanje

Kao što je rečeno u uvodu u izvedbu, jedan od ciljeva projekta je ispitati osnovne algoritme PSO i RPSO i usporediti njihovu efikasnost odnosno performanse. U tu svrhu napravljen je niz ispitivanja s opisanim programom. Rezultati su prikazani u tablici 2. Za svaki problem za dane parametre i metodu dan je broj iteracija potreban da se optimizacija izvrši. Za obje metode rezultati su pisani jedan do drugog kako bi se olakšala usporedba.

d	br	maxX	maxV	init rang		w	RB		SF6		GW		RAS		DJS	
				left	right		pso	rpso	pso	rpso	pso	rpso	pso	rpso	pso	rpso
2	10	100	100	50	100	0.9	3	3	6	7	2	2	8	7	5	3
20	100	100	100	50	100	0.9	375	275	345	315	255	111	460	305	900	680
20	100	100	100	50	100	0.7	110	43	95	130	35	15	115	45	80	40
50	100	100	100	50	100	0.9	610	480	315	290	580	385	660	510	640	450
50	500	100	100	50	100	0.9	550	395	245	100	555	360	605	460	607	410
20	100	500	500	50	100	0.9	550	385	457	450	505	330	570	400	540	380
20	100	500	200	50	100	0.9	405	305	375	260	380	260	550	380	480	390
20	100	100	100	20	200	0.9	350	240	360	195	220	158	350	250	320	210
20	100	300	200	20	200	0.9	400	295	297	350	310	242	405	330	390	315
50	100	100	100	50	100	0.7	305	195	175	50	220	117	400	210	330	200
50	500	100	100	50	100	0.7	270	130	95	40	237	57	270	150	280	105
20	100	500	500	50	100	0.7	165	90	53	90	185	59	245	105	215	70
20	100	500	200	50	100	0.7	160	55	540	150	96	33	198	90	107	50
20	100	100	100	20	200	0.7	60	35	115	60	37	25	70	40	50	40
20	100	300	200	20	200	0.7	150	50	70	205	100	33	145	47	140	50

Tablica 2: Tablica s rezultatima ispitivanja osnovnih PSO i RPSO algoritama

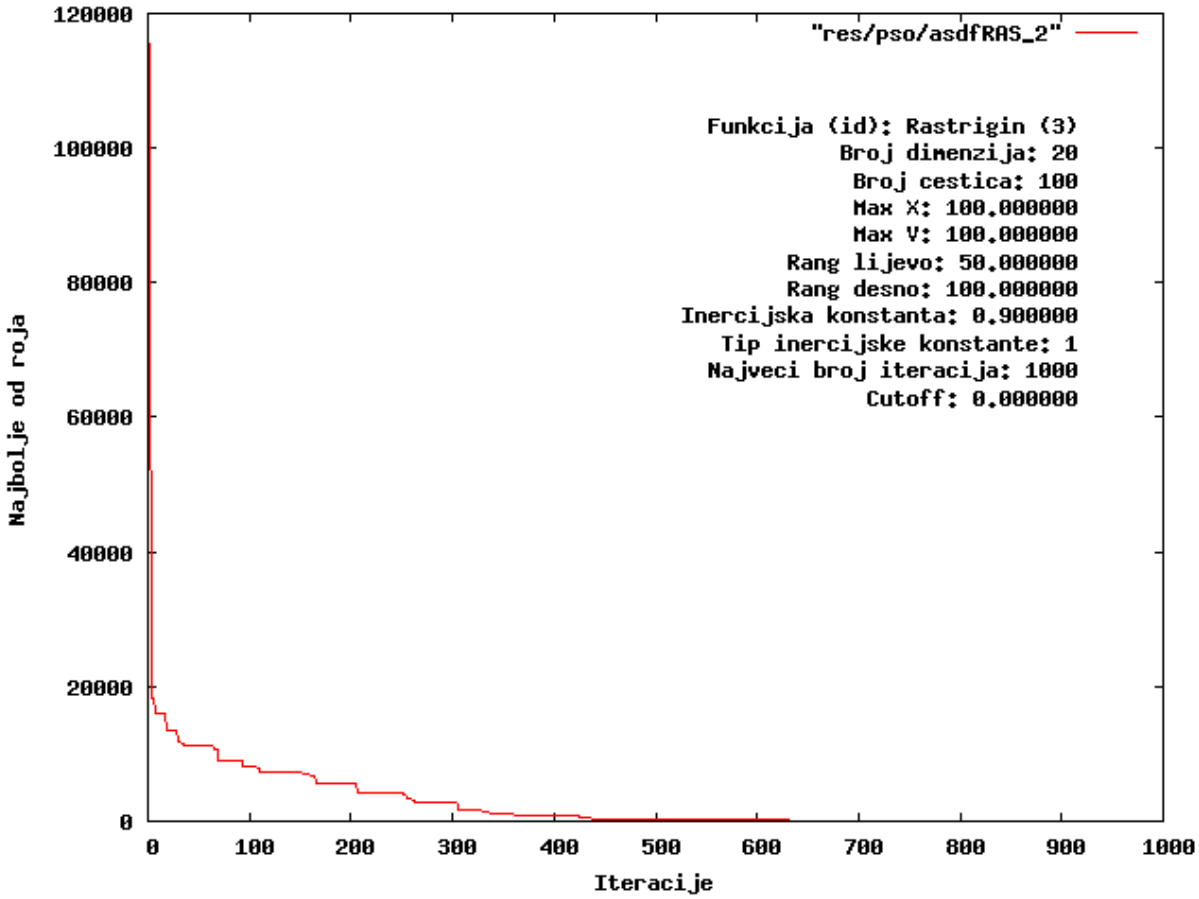
3.3.1 Performanse

U gotovo svim pokusima RPSO je obavio optimizaciju u manjem broju iteracija nego PSO. U nekoliko pokusa sa Schafferovom F6 funkcijom za određene parametre PSO je postigao bolji rezultat nego RPSO. U svim ostalim pokusima RPSO je imao i po nekoliko puta bolji rezultat.

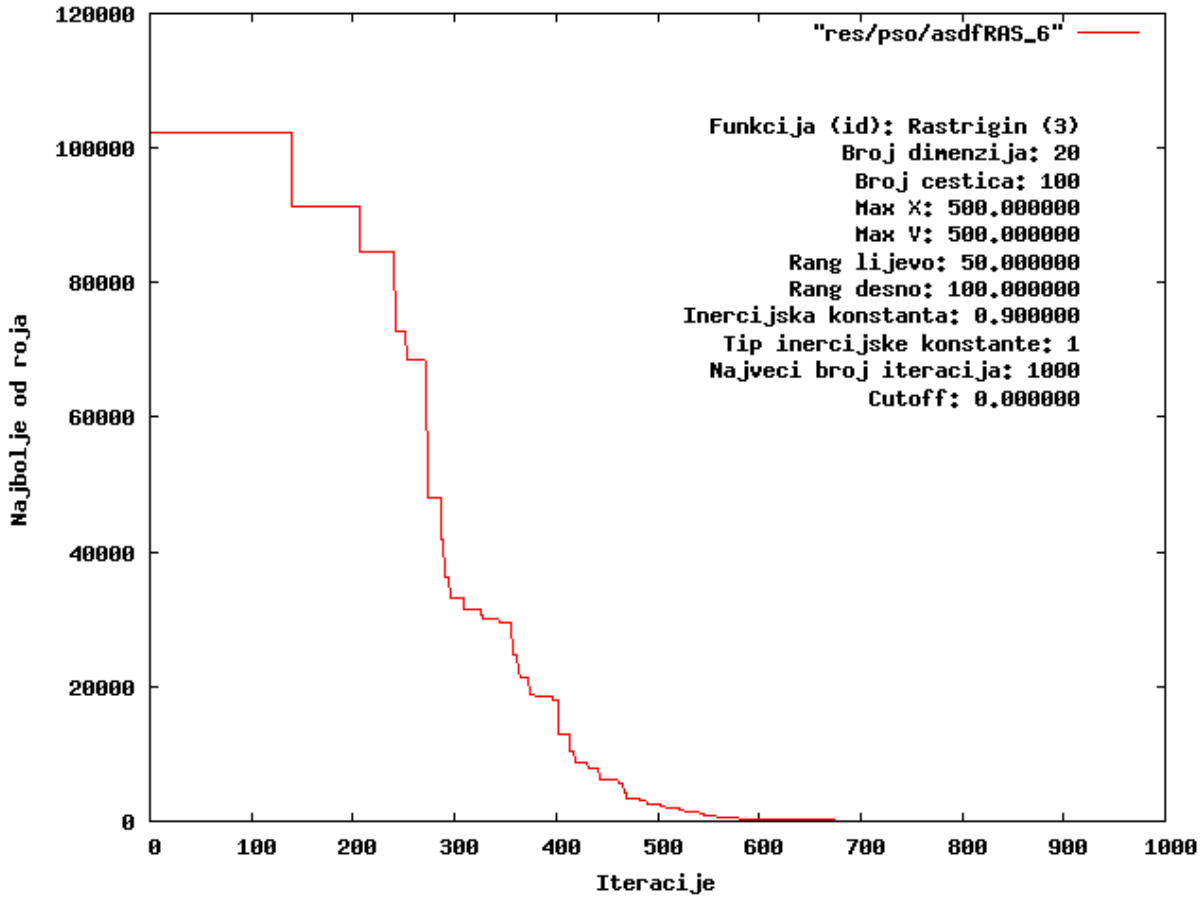
Osim tog, iz pokusa se mogu donijeti i drugi zaključci, prvenstveno o ponašanju obaju metoda ovisno o parametrima.

Smanjivanjem inercijske konstante, rezultati se dobivaju u značajno manjem broju iteracija. Povećavanjem ranga za inicijalizaciju povećavamo početnu raspršenost čestica roja, tako da većim rangom dobijemo veću početnu pokrivenost i time uglavnom bolji rezultat (u nekim slučajevima, taj rezultat je nešto lošiji dok je u nekim slučajevima višestruko bolji). S druge strane, povećanjem veličine prostora, čestice su prisiljene pretražiti veći prostor i treba im više vremena, dakle veći broj iteracija. Povećanjem prostora, drastično se mijenja tijek pronalaženja optimuma, npr. od uobičajenog na slici 3 do ovog na slici 4.

Zanimljivo je primijetiti utjecaj inercijske konstante na performanse. Malim smanjenjem inercijske konstante (s 0.9 na 0.7) moguće je optimizirati problem u više nego duplo više dimenzija (s 20 na 50) u manjem broju iteracija.



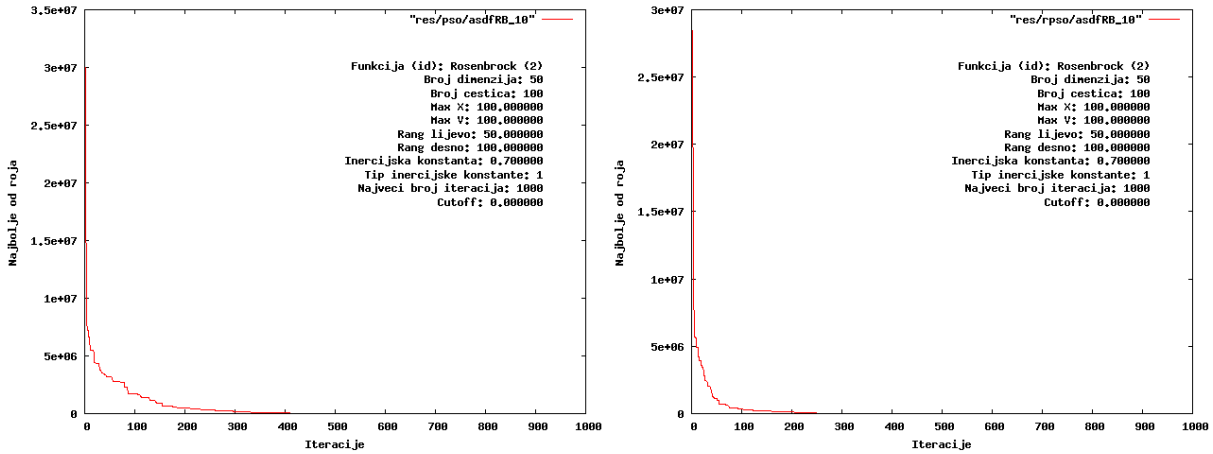
Slika 3: Uobičajeni tijek optimizacije



Slika 4: Tijek optimizacije pri povećanom prostoru pretraživanja

3.3.2 Zaključak

Iako je u nekim slučajevima PSO dao bolji rezultat (Schaffer F6), RPSO je ipak davao bolje rezultate u svima ostalima, uključujući i Schafferovu F6 funkciju (uz drugačije parametre). Budući da je u najvećem broju slučajeva RPSO dao bolji rezultat, RPSO je bolja metoda za optimizaciju realnih funkcija nego standardni PSO.



Slika 5: Usporedba tijeka optimizacije PSO i RPSO

4. Performanse – usporedbe s drugim metodama

Opisana napredna RPSO metoda je ispitana različitim funkcijama, te su rezultati uspoređeni s optimizacijom istih funkcija pomoću genetskog algoritma (GA – *engl. Genetic Algorithm*) ili simuliranog kaljenja (SA – *engl. Simulated Annealing*).

Rezultati upućuju da nijedna od metoda ne može sa sigurnošću naći optimum neobuzdane funkcije. U slučaju *Needle-eye* i *Corana* funkcija, sve tri metode su podjednako dobre dok kod Bukinove funkcije sve tri metode daju rezultate koji su daleko od ispravnih. U nekim slučajevima bolje rezultate daje RPSO, a u nekima GA ili SA. U nešto većem broju slučajeva RPSO daje bolje rezultate. Iz tog bi se brzopleto moglo zaključiti da je RPSO bolji od GA, ali autori pokusa poručuju da je jedino što su zaključili da nitko ne može zajamčiti prevlast jedne metode nad drugom[2]. Svaka metoda podbaci u nekom slučaju i svaka briljira u nekom drugom.

Potrebno je naći neki kriterij prema kojem se mogu klasificirati problemi koji odgovaraju (ili ne odgovaraju) određenoj metodi. Ta klasifikacija će istaknuti prednosti korištenja određene metode kod bavljenja s određenom klasom problema.

5. Primjene i primjeri

Široka primjena PSO algoritama počinje od klasičnih problema izrade rasporeda, problema trgovačkog putnika, treniranja živčanih mreža (*engl. neural networks*) i raspodjele zadataka, pa sve do specijaliziranih problema kontrole snage reaktora i kontrole napona, u medicini, te čak za komponiranje glazbe. Zadnjih godina PSO je postao posebno popularan za optimizaciju problema s više ciljeva, te optimizaciju problema čije se rješenje dinamički mijenja.

Jedna od prvih primjena PSO bila je razvoj struktura živčanih mreža. Zbog svojeg svojstva da brzo konvergira, korištenje PSO kao zamjenu za tradicionalne algoritme značajno je smanjilo vrijeme učenja.

Živčana mreža uzima vektor nezavisnih varijabli i daje procijenjeni vektor zavisnih izlaznih varijabli. Mreža je strukturirana kao skup težina, obično organiziran u slojeve i optimizacijski problem je naći vrijednosti za te težine tako da mreža daje preslike s najmanjom greškom. Mreža je zadužena za razumijevanje i stoga se ponaša kao individua te kao takva donosi odluke o podražaju na osnovu vlastitog iskustva. S druge strane, razumijevanje kod roja čestica je na razini cijelog roja. Svaka čestica leti kroz mrežu i komunicirajući s ostalima optimiziraju mrežu koju svaka čestica za sebe ne razumije, ali ovdje je razumijevanje osobina roja, kojeg je svaka čestica jedan mali dio. [5]

U raznim Java appletima koji su dostupni na internetu možemo s korisničke strane ispitati PSO raznim predefiniranim funkcijama koje se često koriste kao ispitne funkcije za optimizacijske probleme (kugla, Rosenbrock, Rastrigrin, Griewank, DeJong, Ackley i druge) te promatrati ponašanje roja kroz korake algoritma. [4]

Razvijen je i PSO toolbox [3], alat otvorenog koda, pomoću kojeg je moguće optimizirati korisnički definirane funkcije i sustave. Trenutno je u beti i sastoji se od Matlab (.m) funkcija, te ne podržava optimizaciju FIS-a (*engl. Fuzzy Inference System*). U izradi je Java port toolboxa te podrška za optimizaciju FIS-a.

6. Zaključak

U ovom tekstu upoznali smo neke probleme optimizacije, što je to optimizacija matematički gledano, te neka od rješenja iz područja inteligencije roja za optimizacijske probleme. Tako smo saznali što je to PSO, kako tradicionalni algoritam PSO izgleda i jednu njegovu varijantu RPSO, napravili usporedbu PSO i RPSO i donijeli zaključke o njihovim performansama.

Dani su primjeri korištenja, kako s istraživačke, tako i s praktične strane. Istraživačka primjerna je u optimizaciji složenih funkcija pri čemu se pokušavaju postići što bolje performanse, dok je praktična primjerna u treniranju živčanih mreža, te optimizaciji FIS-a.

Ispitivanje je pokazalo da RPSO općenito daje bolje rezultate nego PSO pri optimizaciji realnih funkcija. S druge strane, uspoređujući RPSO s ostalim metodama optimizacije iz područja umjetne inteligencije (GA, SA) dolazi se do zaključka da nema "srebrnog metka", algoritma koji je pogodan za rješavanja svih problema nego da je probleme potrebno kategorizirati i za odgovarajuću kategoriju koristiti odgovarajući algoritam.

7. Literatura

[1] Swarm Intelligence, Introduction and Applications – Christian Blum, Daniel Merkle

[2] Repulsive Particle Swarm Method on Some Difficult Test Problems of Global Optimization - Mishra, SK (2006)

[3] http://psotoolbox.sourceforge.net/

[4] http://www.engr.iupui.edu/~shi/PSO/AppletGUI.html

[5] Swarm Intelligence – James Kennedy