SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT

Optimizacija rojem čestica

Daniel Domović

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

Zagreb, Studeni, 2008.

Sadržaj

1. Uvod

Optimizacija rojem čestica (Particle Swarm Optimisation, PSO) je stohastični algoritam koji se bazira na populaciji rješenja. Pripada skupini algoritama inteligencije roja (Swarm inteligence) koji se temelje na sociološko-psihološkim principima i pružaju uvid u sociološka ponašanja te pomoću njih pridonose inženjerskim aplikacijama. Osmislili su ga dvojica znanstvenika: James Kennedy i R. C. Eberhart ne tako davne 1995. godine. Motivaciju su pronašli u društvenom ponašanju raznih tipova organizama poput jata ptica ili ribljih plova. Tehnike programiranja su otada uvelike unaprijeđene, a originalan algoritam je gotovo i neprepoznatljiv u odnosu na današnje verzije.

1.1. Prirodno uporište

Promatramo jato ptica koje će svoj položaj mijenjati vođeno instinktom za hranjenjem. Sve ptice u jatu traže hranu na nekom prostoru. Vrlo je vjerojatno da će jato slijediti onu pticu koja je osjetila ili pronašla dobar izvor hrane. No, svaka ptica pojedinačno u sebi ima instinkt kojim želi za sebe pronaći još bolje hranilište, a kako bi to i postigla ona se nakratko odvaja od jata. Samim pronalaskom boljeg hranilišta, pomogla je jatu u cjelini jer će se ostale ptice preseliti na bolje hranilište. [2]

1.2. Socijalno uporište

Socijalni utjecaji i društveno učenje pomažu osobi da lakše spozna samu sebe tj. da uspostavi kognitivnu stabilnost. Ljudi rješavaju probleme razgovarajući jedan s drugim i prilikom interakcije s drugim ljudima njihova vjerovanja, pogledi na probleme i ponašanje se mijenjaju (u sociokognitivnom prostoru te se promjene prikazuju kretanjem individualaca). Nadalje, neki znanstvenici predlažu da znanje biva optimizirano socijalnom interakcijom i da razmišljanje nije samo privatno nego i interpersonalno.

PSO kao optimizacijsko oruđe pruža procedure za pretragu bazirane na populaciji u kojoj individualci (čestice) mijenjaju svoju poziciju (stanja) u vremenu. U skladu s tim, u PSO sustavu koji je inicijaliziran populacijom nasumičnih rješenja - česticama, čestice lete po multidimenzionalnom prostoru za pretraživanje. Za vrijeme leta, svaka čestica podešava svoju poziciju na bazi vlastitog iskustva i na temelju iskustva svojih najbližih susjeda te sa tim znanjima iskorištava najbolju poziciju na koju je naišla ona sama ili njen susjed. Detaljnije, svaka čestica unutar sustava pamti koordinate unutar prostora problema koje predstavljaju najbolje dosad postignuto rješenje te čestice. Nazovimo tu vrijednost vlastito najprikladnije rješenje (personal_best). Ako se u roju odrede međusobni susjedi svake čestice, čestica mora pamtiti i najbolje rješenje do kojeg je došla bilo koja susjedna čestica naše promatrane čestice (local_best). Ukoliko su sve čestice međusobni topološki susjedi, najbolje rješenje takvog roja je globalni optimum (global_optimum). Takvi načini pretraživanja imaju svoje prednosti, ali i mane. Tako će traženje lokalnog optimuma bolje istražiti prostor rješenje, no konvergencija će biti sporija.

PSO sustav iskorištava metode lokalnog i globalnog pretraživanja.

PSO dijeli mnogo sličnosti sa tehnikama evolucijskog računanja poput genetičkih algoritama (GA). Pomoću PSO-a optimum se traži obnavljanjem generacija. Za razliku od GA, u PSO algoritmu ne postoje evolucijski operatori kao što su krosoveri ili mutacije. [3]

2. Algoritam

2.1. Matematička formulacija

Roj čestica je u prirodi stohastičan; on iskorištava vektor brzine kako bi ažurirao trenutnu poziciju svake čestice u roju. Vektor brzine se ažurira na temelju pamćenja svake čestice, što koncepcijski odgovara autobiografskoj memoriji, kao i na temelju znanja koje je stekao roj kao cjelina. Pozicija čestice u jatu je ažurirana na temelju socijalnog ponašanja roja koji se prilagođava svom okruženju stalnim traženjem boljih pozicija tokom vremena.

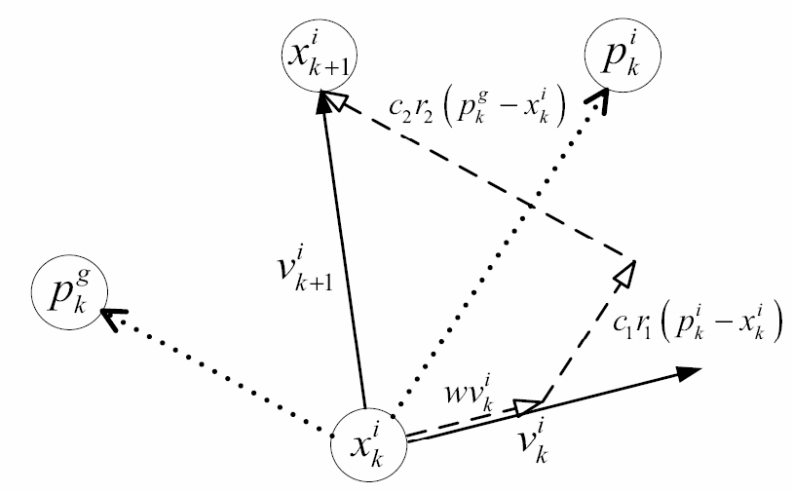
Numerički, pozicija *x* čestice *i* u iteraciji *k+1* je ažurirana na sljedeći način:

$$\boldsymbol{x}_{k+1}^{i} = \boldsymbol{x}_{k}^{i} + \boldsymbol{v}_{k+1}^{i}\Delta t \tag{2.1.1}$$

pri čemu je $\boldsymbol{v}_{k+1}^{i}$ pripadajući ažurirani vektor brzine, a Δt je vrijednost vremenske step funkcije. Vektor brzine svake čestice se računa kao:

$$v_{k+1}^{i} = \omega v_{k}^{i} + c_1 r_1 \frac{\left(p_k^i - x_k^i\right)}{\Delta t} + c_2 r_2 \frac{\left(p_k^g - x_k^i\right)}{\Delta t} \tag{2.1.2}$$

Pri čemu je v_k^i vektor brzine u iteraciji *k*, p_k^i i p_k^g su najbolja ikad pozicija čestice i globalna najbolja pozicija čitavog roja sve do trenutne iteracije *k*, dok *r* predstavlja nasumični broj iz intervala [0, 1]. Preostali članovi su konfiguracijski parametri koji igraju važnu ulogu u konvergencijskom ponašanju PSO-a. Član c_1 (kognitivni, samospoznajni parametar) predstavlja stupanj povjerenja u najbolje rješenje do kojeg je došla pojedina čestica dok član c_2 (socijalni, društveni parametar) predstavlja stupanj povjerenja u globalno najbolje rješenje (najbolje pronađeno rješenje od jata kao cjeline). Uglavnom se uzima $1.8 < c_1 = c_2 < 2.2$. Posljednji član ω je inercijska varijabla koja je iskorištena za kontroliranje istraživačkih sposobnosti roja tako da skalira vrijednost trenutne brzine te na taj način utječe na iznos ažuriranog vektora brzine. Većim vrijednostima inercijske varijable vršimo globalno pretraživanje zbog toga što se ažurirani vektor brzine brže povećava dok zadavanjem manje vrijednosti inercijske varijable vrijednost ažuriranog vektora brzine postaje manja pa se tako novi položaj čestice ograničava na manje područje prostora istraživanja tj. omogućujemo lokalno pretraživanje.



Slika 2.1.1 Promjena položaja

Na slici se vidi pozicija čestice i ažuriranje vektora brzine na prethodno opisan način u dvodimenzionalnom prostoru. Isto tako je vidljivo da će ažurirani položaj čestice ovisiti ne samo o najboljim pozicijama roja i čestice samo, već i o veličini konfiguracijskih parametara. [4]

2.2. Kanonski algoritam

Predloženi algoritam upotrebljava globalno i lokalno najprikladnije rješenje, no ne i susjedno najprikladnije rješenje. Susjedna prikladna rješenja dopuštaju paralelizam u istraživanju prostora i smanjuju mogućnost upadanja u lokalni minimum, no smanjuju brzinu konvergencije.

Samostalno čestica ne može postići ništa, za uspjeh je potrebna suradnja s ostalim česticama roja.

Neka je $f:\square^m \to \square$ funkcija prikladnosti koja uzima rješenje čestice iz višedimenzionalnog prostora i prepisuje ju u jednodimenzionalni prostor. Neka postoji *n* čestica i neka je svakoj dodijeljena pozicija $x_i \in \square^m$ i brzina $v_i \in \square^m, i = 1,...,n$. Neka je $\hat{x}_i$ trenutno najprikladnije lokalno rješenje svake čestice, a $\hat{g}$ neka predstavlja najprikladnije globalno rješenje.

Kao i ostali numeričko bazirani optimizacijski pristupi, PSO je po svojoj prirodi iterativan, a njegov osnovni algoritam je konstruiran na sljedeći način:

- Inicijaliziraj x_i i v_i za sve *i*.
- $\hat{x}_i \leftarrow x_i \wedge \arg\min_{x_i} f(x_i), i = 1,...,n$
- Dok nije došlo do konvergencije
  - Za svaku česticu $1 \le i \le n$
    - Stvori nasumične vektore r_1, r_2
    - Ažuriraj poziciju čestice: $x_i \leftarrow x_i + v_i$
    - Ažuriraj brzinu čestice: $v_i \leftarrow \omega v_i + c_1 r_1 \circ (\hat{x}_i - x_i) + c_2 r_2 \circ (\hat{g} - x_i)$
    - Ažuriraj najprikladnije lokalno rješenje: ako je $f(x_i) < f(\hat{x}_i), \hat{x}_i \leftarrow x_i$
    - Ažuriraj najprikladnije globalno rješenje: ako je $f(x_i) < f(\hat{g}), \hat{g} \leftarrow x_i$
- $\hat{g}$ je najbolje rješenje s prikladnošću $f(\hat{g})$

Promotrimo sljedeće parametre navedenog algoritma:

- ω je inercijalna konstanta čije su vrijednosti najčešće manje od 1.
- c_1 je kognitivni parametar koji predstavlja vrijednost samospoznaje čestice, odnosno najboljeg lokalnog rješenja, dok je c_2 društveni parametar koji predstavlja važnost društva, odnosno globalno najprikladnijeg rješenja. (c_1, $c_2 \approx 2$). Ova dva koeficijenta se nazivaju još i koeficijenti povjerenja.
- r_1 i r_2 su dva nasumična vektora čije komponente poprimaju vrijednosti iz intervala [0, 1]. [3]

2.3. Pseudokod

```
// inicijaliziraj pozicije čestica i njihovih brzina

 za (I = 1 do broj čestica n){
   za (J = 1 do broja dimenzija prostora m){

      X[I][J] = donja_granica + (gornja_granica – donja_granica) * homogeni
                nasumični broj
      V[I][J] = 0

   }
 }

 // inicijaliziraj globalnu i lokalnu vrijednost prikladnosti na najgoru
    moguću

 prikladnost_gbest = inf;
 za (I = 1 do broj čestica n){

    prikladnost_lbest[I] = inf

 }

 // ponavljaj do konvergencije (u ovom primjeru do konačnog, ranije
    izabranog, broj iteracija)

 za (k = 1 do broj iteracija t){

   // izračunaj prikladnost svake čestice
   prikladnost_X = izracunaj_prikladnost(X)

   // ažuriraj lokalno najbolje rješenje i njegovu prikladnost
   za (I = 1 do broj čestica n){

      ako (prikladnost_X[I] < prikladnost_lbest[I])
       prikladnost_lbest[I] = prikladnost_X[I]
       za (J = 1 do broj dimenzija m){

         X_lbest[I][J] = X[I][J]

       }
     }
   }

   // ažuriraj globalno najbolje rješenje i njegovu prikladnost

   [min_prikladnost, min_prikladnost_index] = min(prikladnost_X)
   ako (min_prikladnost < prikladnost_gbest){
       prikladnost_gbest = min_prikladnost
       za (J = 1 do broj dimenzija m){

         X_gbest[J] = X(min_prikladnost_index,J)

       }
   }
```

```
  // ažuriraj brzinu i položaj čestice
  za (I = 1 do broj čestica n){
    za (J = 1 do broj dimenzija m){

      R1 = nasumični broj
      R2 = nasumični broj
      V[I][J] = w*V[I][J]
             + C1*R1*(X_lbest[I][J] - X[I][J])
             + C2*R2*(X_gbest[J] - X[I][J])
      X[I][J] = X[I][J] + V[I][J]

    }
  }
}
```

Slika 2.3.1 Pseudokod PSO algoritma [3]

3. Primjene

Vrlo popularno područje primjene PSOa je dizajn antena – kontrola i dizajn faznih polja, dizajniranje i modeliranje širokopojasnih antena, ispravljanje grešaka u polju, dizajniranje ugradbenih antena…

Sljedeće važno područje primjene nalazimo u biomedicini gdje se PSO upotrebljava u svrhu detekcije Parkinsonove bolesti na temelju drhtavice, optimizacije biomehaničkog ljudskog pokreta, klasifikacije raka i predviđanja ostatka života, dizajniranje lijekova…

U području komunikacijskih mreži, PSO se koristi u dizajniranju bluetooth mreža, za usmjeravanje, za izgradnju radarskih mreža, rezervaciju pojasa…

U području kombinatoričkih problema, PSO služi za rješavanje problema trgovačkog putnika, optimizaciju puta, za rješavanje problema naprtnjače i N-kraljica…

Dizajn je jedno popularno područje koje je u IEEE zastupljeno s 5% bibliografije, a ono uključuje: dizajniranje motora, antena, filtera, dizajn kuhala, konceptualni dizajn…

Dizajniranje i restrukturiranje električnih mreža zauzima 7% bibliografije, a u ovo područje primjene spadaju širenje i rekonfiguracija mreža, distribuirano stvaranje te regulacija napona…

U kombinaciji s neuralnim mrežama, PSO se koristi za inverziju neuralnih mreža, kontrolu neuralnih mreža za nelinearne procese, za kontrolu mobilnih neuralnih mreža i za izgradnju neuralnih kontrolora.

PSO se koristi i u funkciji predviđanja kvalitete i klasificiranja vode, u predviđanju ponašanja kaotičnih sustava, u ekološkim modelima, u meteorološkim predviđanjima, za predviđanje migracije slonova te gradskog prometa.

Uz navedena, PSO se upotrebljava i u sljedećim područjima: enegteski sustavi, robotika, raspoređivanje, sigurnost i vojska, mreže senzora, procesiranje signala, modeliranje, metalurgija, zabava, financije, grafika i vizualizacija, motori te elektrotehnika.

S obzirom da PSO algoritam ima više svojih inačica te da se iz njega izrodilo već nekoliko novih podvrsta (npr. RPSO), u nastavku ću opisati jednu, u literaturi ne tako često spominjanu verziju – plemena. [11]

3.1. Plemena (TRIBES)

Ovaj model PSO algoritma pronalazi rješenje problema tako da krene od jedne jedine čestice, a ostale dodaje ili miče inteligentno, po potrebi.

Klasičnom PSO-u obično trebamo zadati parametre – numeričke koeficijente, veličinu roja, veličinu susjedstva i topologiju. Važnost je očita, program ne može sam znati o kojem je problemu riječ ili koji se stupanj točnosti od njega zahtijeva. Pomoću raznih pravila dajemo do znanja programu kako bi on trebao raditi, no takva pravila najčešće nisu robusna jer za neke probleme daju sjajna rješenja, dok za ostale ne.

Plemena su verzija PSO algoritma koji pronalazi parametre sam i dalje *dovoljno dobra* rješenja (pojam *dovoljno dobra* predstavlja inženjerski pogled na rješenje – rješenje nije izvrsno, ali nije ni poražavajuće; razumna strategija dat će kvalitetnija rješenja). Nažalost, ovim se postupkom ipak gubi na efikasnosti jer ovaj algoritam parametre mora pronaći sam i to u samo jednom izvođenju što ipak neće dati jednako dobra rješenja kao kakako bi se parametri zadali ručno, a odredili kroz niz testova.

Plemena se služe dvama tehnikama koja omogućuju njihovo ispravno izvođenje: koriste se hipersfere umjesto hiperparalelepipeda u prostoru leta čestica i vrši se adaptacija veličine roja na temelju odnosa između čestica.

3.1.1. Opis plemena

1. Što su plemena?

Doušnik čestice A je čestica B čiju dosad najprikladniju pronađenu poziciju može pročitati čestica A. Ova definicija insinuira da je svaka čestica doušnik samoj sebi. Doušnik je „vanjski“ ako ne pripada plemenu čestica. Kakako bismo čestice promatrali kao da su vrhovi nekog grafa, bridovi između njih predstavljali bi informacijske kanale. Svaki će čvor imati i petlju jer je svaka čestica doušnik samoj sebi.

Pleme je podskup čestica u kojem svaka čestica može informirati svaku česticu unutar podskupa. Pleme je metafora za grupe različitih veličina koje se kreću u nepoznatom okružju tražeći neko pogodno mjesto.

2. Kakva je povezanost unutar i između plemena?

Čak i ako pojedino pleme nađe lokalni minimum, oni moraju komunicirati međusobno kako bi odlučili koja je od tih vrijednosti globalni minimum što znači da informacijski putevi između plemena cijelo vrijeme trebaju biti aktivni. Zaključujemo da globalna struktura izgleda na sljedeći način: svako je pleme gusta mreža čestica i između plemena mostoji mreža koja osigurava međusobnu komunikaciju.

- Kvaliteta čestice

Kao i u klasičnom PSO-u čestica ima trenutnu poziciju kao i dosad najprikladniju poziciju koja se memorira. Na tom nivo se utvrđuje da li je došlo do poboljšanja čestice odnosno čestica je *dobra* ako je popravila svoju zadnju zapamćenu najprikladniju poziciju, inače se smatra neutralnom.

Po definiciji, najprikladnija pozicija čestice se ne može pokvariti i zato se ne definira apsolutno *loša* čestica. Do saznanja da je čestica *loša* dolazimo isključivo usporedbom. U skladu s tim, mogu se odrediti čestice koje imaju najlošije i najbolje značajke unutar plemena.

Nadalje, memorija čestice se povećava u odnosu na klasični PSO pa tako ona sada pamti i posljednje dvije varijacije u performansi te na taj način stvara kratku povijest svog kretanja. U skladu s time se definira i treći status čestice – *odlična*, u slučaju da su te zadnje dvije varijacije poboljšanja. Treći status je uveden kako bi se lakše odabrala strategija kretanja čestice.

- Kvaliteta plemena

Što je više dobrih čestica u plemenu, to je i pleme bolje. U primjeni se status plemena računa na slijedeći način: neka je T veličina plemena (broj čestica u plemenu) i neka je G broj dobrih čestica najviše jednak T. Broj p je nasumično generiran prema uniformnoj distribuciji od 0 do T. Ako je G manji ili jednak p, pleme je loše, inače je dobro.

- Evolucija plemena

    - Micanje čestice

S obzirom da se optimum želi naći u što manjem broju ponavljanja, čim se ukaže prilika za uklanjanjem čestice iz plemena trebamo je iskoristiti. Brisati se smiju samo loše čestice, dakle bolje je zabunom sačuvati česticu nego ju zabunom obrisati. Česticu može izbrisati samo dobro pleme s tim da briše najgoru česticu jer informacija koju ona prenosi (svoju njabolju performansu) i nije najpotrebnija. Podsjetimo se, čestica predstavlja potencijalno rješenje pa uklanjanjem loše čestice uklanjamo loše rješenje.

    - Generiranje čestice

Nova se čestica generira na najjednostavniji mogući način – potpuno nasumično, prema uniformnoj distribuciji unutar prostora pretraživanja i generira ju loše pleme i ostaje s njom u kontaktu.

    - Frekvencija adaptacije

Niti je poželjno niti potrebno raditi ovakve strukturalne adaptacije nakon svake iteracije, potrebno je dati čestici šansu za propagacijom. Ako je ukupan broj informacijskih puteva (komunikacijski put koji spaja dvije čestice) L (nakon jedne strukturalne adaptacije), iduća će strukturalna adaptacija nastupiti nakon L/2 iteracija.

- Kretanje roja

Na početku postoji samo jedna čestica koja predstavlja jedno pleme. Nakon prve iteracije, ako se situacija ne popravi, što je vrlo vjerojatno, generira se sljedeća čestica koja formira novo pleme.

U sljedećoj iteraciji ako se situacija između dviju čestica ne popravi, svako će pleme stvoriti novu česticu simultano, a one će formirati novo dvočlano pleme i proces će se nastaviti. Kada se rješenje ne nazire, stvaraju se sve brojnija plemena, moć pretraživanja se povećava. Primijetimo da se stvaranje novih plemena odvija sve rjeđe zbog povećanja broja informacijskih putova što je povezano s ranije spomenutim pravilom (ako je ukupan broj informacijskih puteva (komunikacijski put koji spaja dvije čestice) L (nakon jedne strukturalne adaptacije), iduća će strukturalna adaptacija nastupiti nakon L/2 iteracija). Između dviju adaptacija, roju se povećavaju šanse da pronađe ispravno rješenje.

- Strategije kretanja

Čestica preuzima onu metodu kretanja koja ovisi o njenoj prošlosti. Ako je primjerice, čestica napredovala dvaput, ona se ne treba previše odvajati od prostora u kojem se sada nalazi.

Ako je čestica izvrsna koristimo metodu – jednostavan pivot (simple pivot method). U protivnom koristimo metodu živahnog pivota (noisy pivot method).

3.1.2. Algoritam

- generiraj jato (jedna čestica – jedno pleme)
- za svaki vremenski korak
    - doušnik trenutne čestice je najbolja čestica plemena (šaman)
    - ako je čestica šaman, izaberi nasumičnog doušnika između drugih šamana ili u arhivi
    - primijeni strategiju kretanja na temelju prošlosti čestice
    - s vremena na vrijeme
        - provjeri da li je pleme dobro ili loše
        - ako loše, generiraj novu česticu
        - ako su čestice dobre, ukloni najgoru česticu
    - s vremena na vrijeme
        - provjeri je li roj dobar ili loš
        - ako loš, dodaj novo pleme
        - ako dobar i plemena ima dovoljno, ukloni najgore pleme

Slika 3.1.2.1 Pseudokod plemena

3.1.3. Primjena

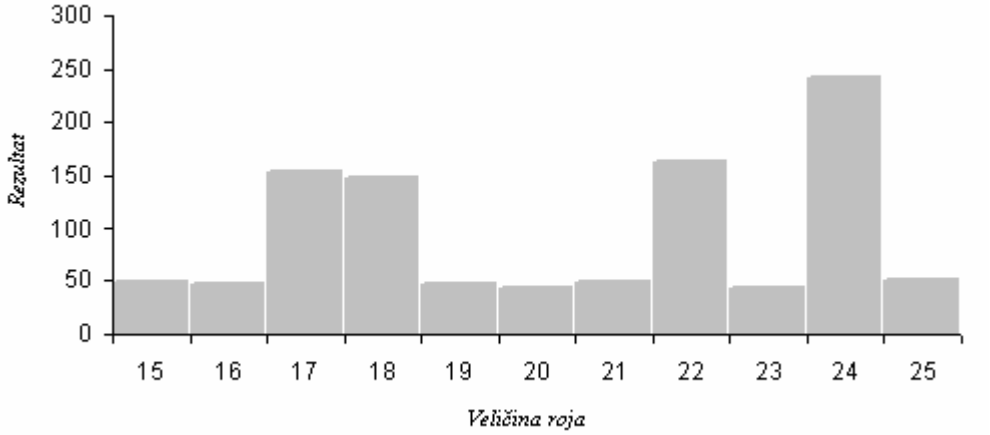
Pretraga s konstantnim brojem evaluacija

Započnimo pitanjem – ako na raspolaganju imam limit od 40000 evaluacija funkcije, koji je najbolji rezultat koji mogu očekivati? Koja je vjerojatnost uspjeha? Ako koristimo CPSO najbolju veličinu roja moramo naći pokretanjem programa dovoljno mnogo puta kako bismo imali ideju koje bi rješenje uopće moglo doći u obzir, no kako doći do rješenja koristeći plemena?

Promotrimo Rosenbrock funkciju na intervalu $[-10,10]^{30}$:

$$f(x) = f((x_1,...,x_d,...x_D)) = \sum_{d=1}^{D-1}(1 - x_d)^2 + 100\left(x_d^2 - x_{d+1}\right)^2 \qquad (3.1.3.1)$$

Za svaku veličinu roja napravili smo stotinu pokretanja programa i srednja vrijednost rezultata je zabilježena na ordinati. Optimalna veličina roja je 20, no nema nekog posebnog pravila kako bismo to odredili. Rojevi veličine 15, 16 i 23 su gotovo jednako dobri.



Slika 3.1.3.1 30D Rosenbrock. Ograničeni smo na 40000 evaluacija. Sa klasičnim PSO-om rezultat je očito varijabilan, ovisi o veličini roja, bez nekog posebnog pravila. Za svaku veličinu roja, napravili smo stotinu pokretanja programa. Najbolji rezultat postigao se za roj veličine 20 čestica.

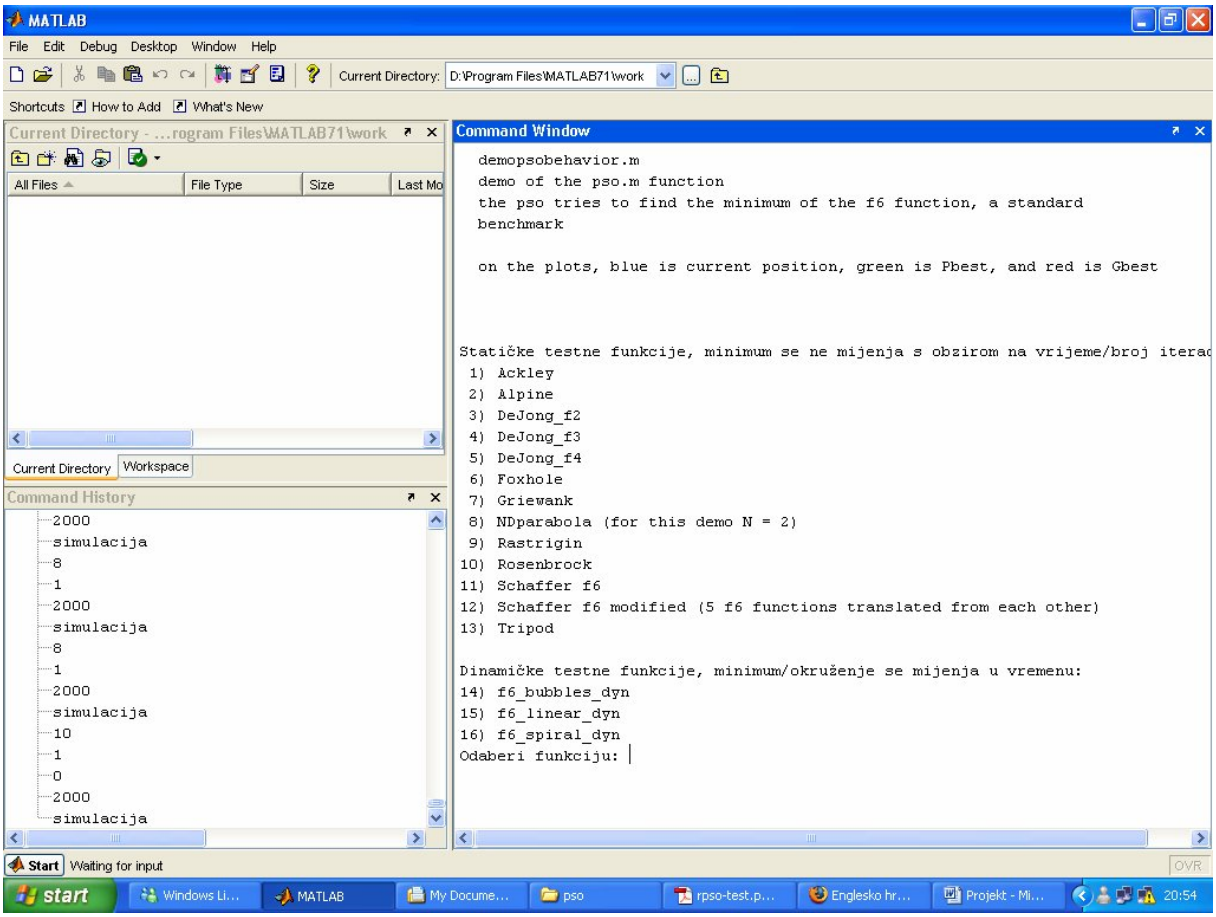
Kako bismo vidjeli efikasnost plemena, pokrećemo CPSO I plemena istovremeno. U svrhu dobivanja boljih vjerojatnosti, program pokrećemo 500 puta. Sa CPSO srednja vrijednost iznosi 49,6 (standardna devijacija= 49,6), dok sa plemenima dobivamo rezultat 42,6 (standardna devijacija= 39,6). Dakle, plemena su bolja od CPSO-a jer češće pronalaze vrijednosti manje od 25 nakon 40000 evaluacija. [1]

4. Praktični rad

akos am u teoretskom dijelu ovog rada stekao neka znanja o PSO-o, u praktičnom sam dijelu odlučio ta znanja potvrditi testiranjem. U testiranju sam se služio programom PSO Toolbox.

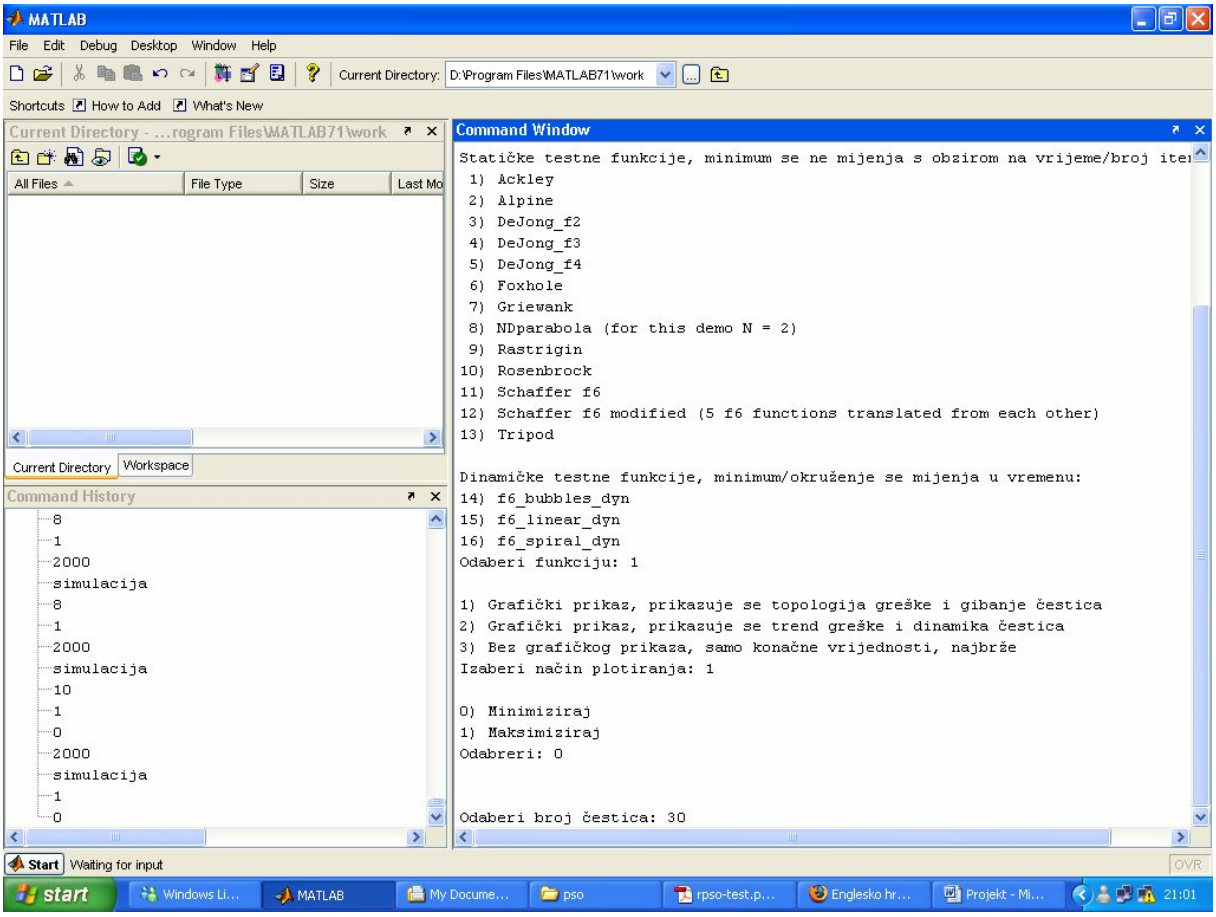
PSO Toolbox je kolekcija Matlab datoteka koje se upotrebljavaju u svrhu optimiziranja sustava. Dosad su izdanje dvije alfa i dvije beta verzije. Posljednja beta verzija objavljena je 2004. godine.

PSO Toolbox je osmišljen za istraživače na području računalne inteligencije. Dovoljno je robustan da je pomoću njega napisano već nekoliko znanstvenih radova, no istovremeno se konstantno razvija. Jednostavan je za upotrebu i lagan za osobnu prilagodbu.



Slika 4.1. Početni zaslon PSO Toolboxa u Matlabu

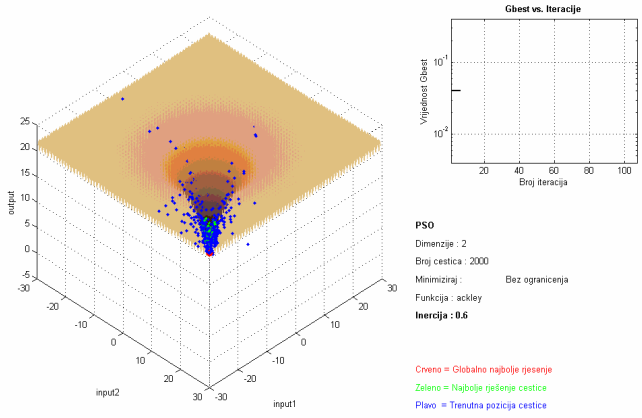
Program se pokreće pozivom simulacije. Program razlikuje 13 statičkih i 3 dinamičke funkcije. Kod statičkih se funkcija minimum ne mijenja s obzirom na vrijeme, odnosno na broj iteracija, dok se kod dinamičkih funkcija okruženje, odnosno minimum funkcije ne mijenja u vremenu. Nakon što prikaže menu s funkcijama, program traži unos rednog broja funkcije koju smo namjeravali testirati.



Slika 4.2. Izbor u PSO Toolboxu

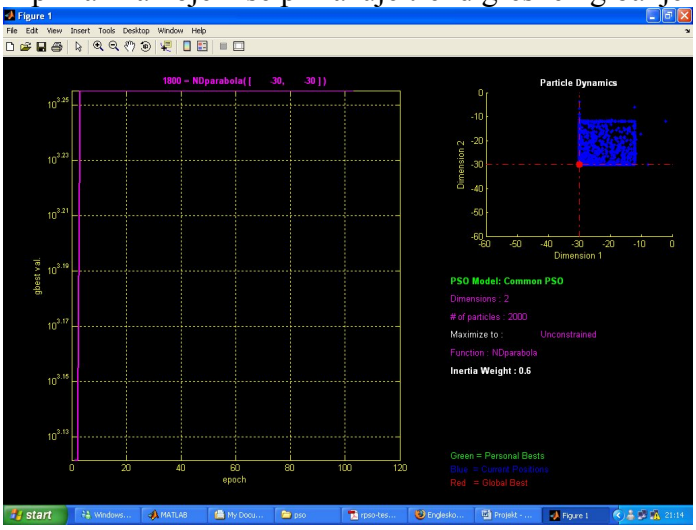
Nakon što smo odlučili koju funkciju želimo testirati, program nas traži da odaberemo jedan od tri načina prikaza rezultata:

- grafički prikaz na kojem se prikazuje topologija i gibanje čestica. Najsporiji je jer zahtijeva bolje računalne značajke



Slika 4.3. Grafički prikaz 1.

grafički prikaz na kojem se prikazuje trend greške i gibanje čestice



Slika 4.4. Grafički prikaz 2

- tekstualni prikaz koji je ujedno i najbrži jer prikazuje samo konačne vrijednosti

```
Naprikladniji parametri:
 GBest = ackley( [ input1, input2 ] )
--------------------------------
      input1 = 0
      input2 = 0
      GBest = -8.8818e-016
      Avg = 0.0022568
      # = 107
```

Slika 4.5. Tekstualni prikaz

Nakon odabira načina ispisa rezultata, preostaje nam još odabrati optimizacijski problem. Za svaku funkciju PSO Toolbox može naći njen minimum ili maksimum.

Za kraj, biramo broj čestica koji optimiziraju funkciju.

Parametri poput inercije, kognitivnih koeficijenata te broja iteracija mijenjaju se unutar same datoteke pomoću koje pokrećemo simulaciju i to zbog dobivanja na performansama i jednostavnosti testiranja. [12]

5. Definicija optimizacijskog problema

Kao što je spomenuto u prethodnom poglavlju, pomoću PSO Toolboxa rješavamo optimizacijske probleme odabranih funkcija dakle tražimo njihov minimum odnosno maksimum.

Minimum i maksimum (ekstremi) su najmanja (minimum) ili najveća (maksimum) vrijednost koju funkcija poprima unutar zadanog susjedstva (lokalni ekstrem) ili na čitavoj domeni funkcije (globalni ekstrem).

Matematički: neka je $f:\Omega\to$ realna funkcija od više varijabli. Kažemo da f ima u točki $P_0\in\Omega$:

- maksimum, ako $f(P)\le f(P_0),\forall P\in\Omega$,
- strogi maksimum, ako $f(P)\le f(P_0),\forall P\in\Omega, P\neq P_0$,
- minimum, ako $f(P)\ge f(P_0),\forall P\in\Omega$¨,
- strogi minimum, ako $f(P)\ge f(P_0),\forall P\in\Omega, P\neq P_0$.

Ako je funkcija neprekinuta na zatvorenom intervalu tada, tada prema teoremu o ekstremnim vrijednostima, minimum i maksimum postoje. Nadalje, globalni maksimum (minimum) mora biti ili lokalni maksimum (minimum) u unutrašnjosti domene ili mora ležati na granicama domene. Dakle, globalni maksimum (minimum) ćemo pronaći tako da tražimo lokalni maksimum (minimum) u unutrašnjosti domene i da istu vrijednost pokušamo pronaći na rubovima domene te da uzmemo najveću (najmanju) od njih.

Lokalne ekstreme pronalazimo preko Fermatova teorema koji kaže da se lokalni ekstremi pojavljuju u stacionarnim točkama. Da li je u stacionarnoj točci riječ o minimumu ili maksimumu određujemo preko testa s prvom i drugom derivacijom funckije čije stacionarne točke promatramo. [9][10]

5.1. Pretpostavke

Prije početka testiranja, iznijet ću tezu čiju ispravnost želim provjeriti. Prvi korak testiranja je provjeriti na koji način broj čestica u roju utječe na brzinu pronalaska ekstrema te kako broj čestica utječe na preciznost dobivenog rješenja. Može li se jednako precizno rješenje dobiti s bilo kojim brojem čestica ili je potrebno ranije procijeniti koliko je čestica potrebno za optimizaciju neke funkcije?

Pretpostavljam kako bi veći broj čestica usporio značajke jer se vektor brzine treba ažurirati više puta u jednoj iteraciji, no isto tako mislim kako bi veći broj čestica u konačnici trebao dati preciznije rješenje jer više čestica pretražuje isti prostor.

6. Rezultati eksperimenata

Cilj ovog eksperimenta je istražiti na koji način broj čestica utječe na brzinu pronalaska ekstrema te koliko je vrijednost tog ekstrema precizna.

6.1. Utjecaj broja čestica na preciznost globalnog ekstrema uz konstantne parametre

Budući se testira samo ovisnost rješenja samo o promjeni broja čestica, u nastavku su navedene vrijednosti (parametri) koji su postavljeni kao konstantni za vrijeme cijelog ispitivanja. Njihove vrijednosti odgovaraju najčešćim vrijednostima koje se koriste kod testiranja PSO algoritma:

- inercija = 0,9
- kognitivni parametri = 2,1
- broj iteracija = 400

Rješenja su upisana u tablice:

- BROJ ČESTICA predstavlja broj čestica koje su sudjelovale u optimizaciji
- GBEST je vrijednost globalnog ekstrema nakon svih iteracija
- AVG je srednja vrijednost svih pronađenih ekstrema tokom evolucije rješenja
- # je broj iteracija u koliko je program završio s optimizacijom. Program je podešen na način da se, ako se vrijednost ekstrema nije promijenila u proteklih 100 iteracija, ta zadnja vrijednost uzima kao konačna i program zaustavlja daljnju pretragu.
- KOORDINATE su x i y pozicija ekstrema

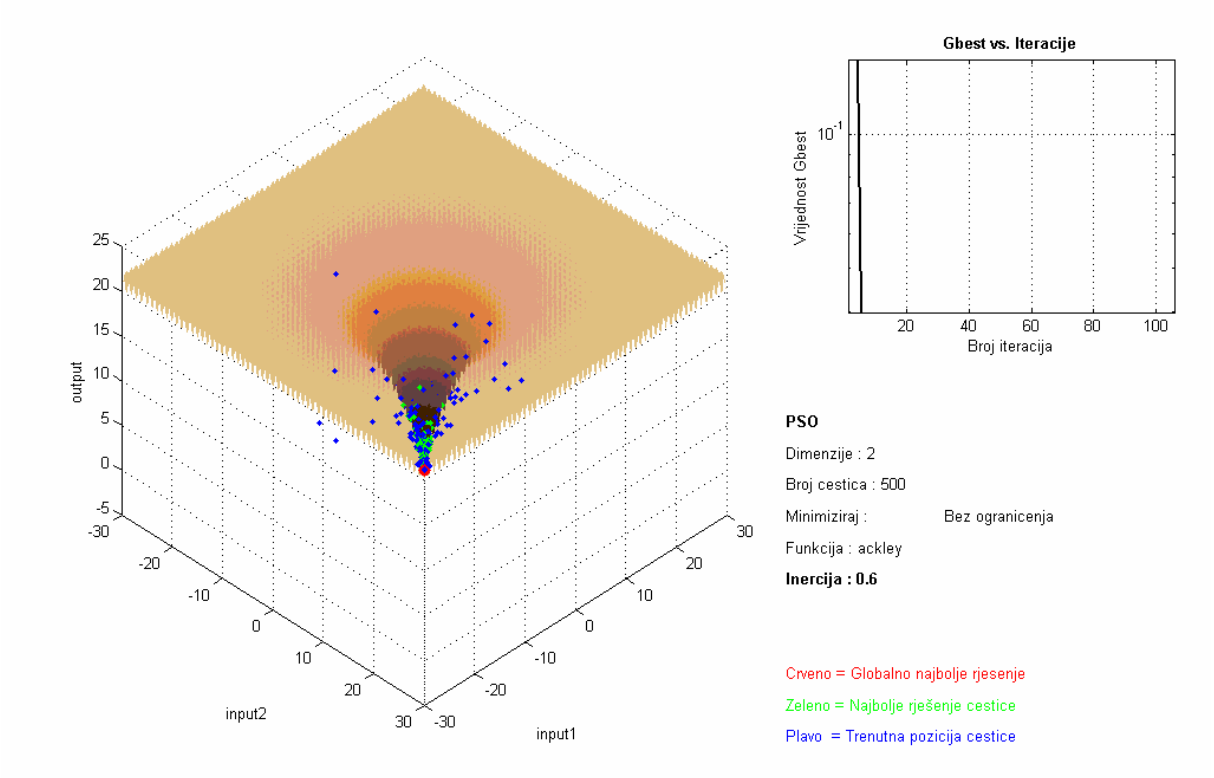
6.1.1. Minimizacija

STATIČKE FUNKCIJE:

- ACKLEY

Broj čestica	30	500	2000
GBest	*3.8192e-014*	*-8.8818e-016*	*-8.8818e-016*
Avg	*0.17071*	*0.0098266*	*0.0020801*
#	*400*	*107*	*105*
Koordinate	*(8.0774e-016, -1.3055e-014)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.1 Ovisnost GBest o broju čestica za funkciju Ackley



Slika 6.1.1.1 Prikaz topologije greške i dinamike čestica za minimum Ackley funkcije

Za ovu je funkciju vidljivo da je 30 premali broj čestica kako bi se unutar 400 iteracija dobio ispravan minimum. Već se za 500 on postiže, a povećanjem broja čestica na 2000 smanjuje se broj iteracija čime se proces optimizacije ubrzava.

- ALPINE

Broj čestica	30	500	2000
GBest	*0*	*0*	*0*
Avg	*0.015292*	*0.00072024*	*5.3998e-005*
#	*112*	*105*	*105*
Koordinate	*(0, 0)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.2 Ovisnost GBest o broju čestica za funkciju Alpine

Povećanjem broja čestica, smanjuje se trajanje izvođenja i preciznost raste što je vidljivo iz prosječnih vrijednosti ekstrema tokom iteracija koja se smanjuje s povećanjem broja čestica.

- DEJONG_F2

Broj čestica	30	500	2000
GBest	*7.727e-007*	*4.1352e-016*	*2.3742e-018*
Avg	*0.45553*	*0.055444*	*0.018087*
#	*400*	*400*	*400*
Koordinate	*(0.99928, 0.99862)*	*(1, 1)*	*(1, 1)*

Tablica 6.1.1.3 Ovisnost GBest o broju čestica za funkciju Dejong_f2

400 iteracija je premalo kako bi se dobio ispravan minimum, a iz dobivenih vrijednosti može se zaključiti da preciznost raste porastom broja čestica.

- DEJONG_F3

Broj čestica	30	500	2000
GBest	*-60*	*-60*	*-60*
Avg	*-59.5481*	*-59.5146*	*-59.5728*
#	*104*	*104*	*103*
Koordinate	*(-30, -30)*	*(-30, -30)*	*(-30, -30)*

Tablica 6.1.1.4 Ovisnost GBest o broju čestica za funkciju Dejong_f3

Preciznost rješenja raste porastom broja čestica.

- DEJONG_F4

Broj čestica	30	500	2000
GBest	*0*	*0*	*0*
Avg	*0.00014226*	*2.0934e-009*	*9.0289e-010*
#	*104*	*108*	*105*
Koordinate	*(0, 0)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.5 Ovisnost GBest o broju čestica za funkciju Dejong_f4

Preciznost rješenja raste porastom broja čestica.

- FOXHOLE

Broj čestica	30	500	2000
GBest	*0.0020077*	*0.0020077*	*0.0020077*
Avg	*0.0020077*	*0.0020077*	*0.0020077*
#	*337*	*284*	*254*
Koordinate	*(-24.0085, 24.0086)*	*(-24.0085, -24.0085)*	*(24.0086, -24.0085)*

Tablica 6.1.1.6 Ovisnost GBest o broju čestica za funkciju Foxhole

Nije potrebno mnogo čestica da se dođe do ekstrema. Preciznost je ista s porastom broja čestica, jedino se vrijeme izvođenja smanjuje.

- GRIEWANK

Broj čestica	30	500	2000
GBest	*2.6826*	*2.6826*	*2.6826*
Avg	*2.7309*	*2.6905*	*2.6922*
#	*400*	*394*	*400*
Koordinate	*(24.6395, 28.985)*	*(24.6395, 28.985)*	*(24.6395, 28.985)*

Tablica 6.1.1.7 Ovisnost GBest o broju čestica za funkciju Griewank

Postoji mogućnost da je 400 iteracija premalo kako bi se dobio ispravan minimum te je bolje povećati broj iteracija za ovu funkciju.

- 2D PARABOLA

Broj čestica	30	500	2000
GBest	*3.5899e-032*	*0*	*0*
Avg	*0.24906*	*0.00067473*	*4.1064e-006*
#	*400*	*107*	*106*
Koordinate	*(-1.8874e-016, -1.6559e-017)*	*(0, 0)*	*(0, 0)*

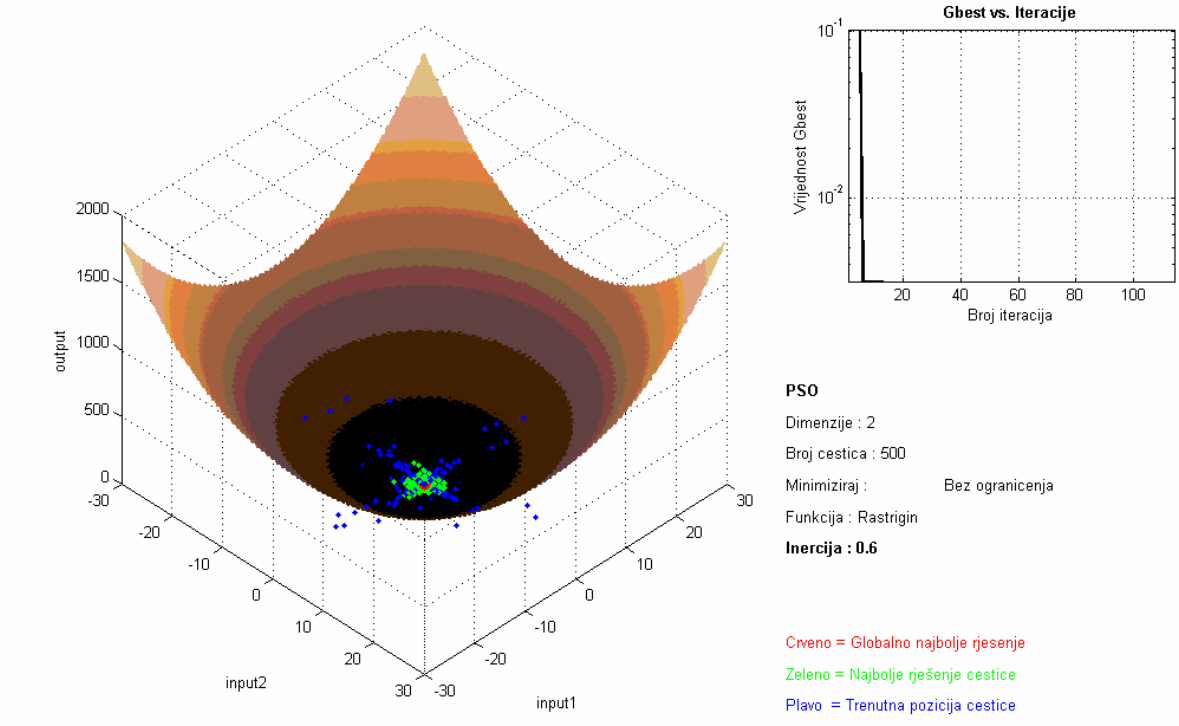
Tablica 6.1.1.8 Ovisnost GBest o broju čestica za funkciju 2D parabola

Preciznost rješenja i brzina izvođenja rastu s porastom broja čestica.

- RASTRIGIN

Broj čestica	30	500	2000
GBest	*0*	*0*	*0*
Avg	*0.44972*	*0.0052651*	*0.00028679*
#	*383*	*105*	*105*
Koordinate	*(1.9726e-010 , -9.9367e-010)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.9 Ovisnost GBest o broju čestica za funkciju Rastrigin



Slika 6.1.1.2 Prikaz topologije greške i dinamike čestica za minimum Rastrigin funkcije

Preciznost rješenja i brzina izvođenja rastu s porastom broja čestica.

- ROSENBROCK

Broj čestica	30	500	2000
GBest	*2.9426e-008*	*5.2817e-015*	*3.0385e-018*
Avg	*0.30904*	*0.072248*	*0.012545*
#	*400*	*400*	*400*
Koordinate	*(0.99994, 0.99987)*	*(1, 1)*	*(1, 1)*

Tablica 6.1.1.10 Ovisnost GBest o broju čestica za funkciju Rosenbrock

Preciznost rješenja i brzina izvođenja rastu s porastom broja čestica.

- SCHAFFER_F6

Broj čestica	30	500	2000
GBest	*0*	*0*	*0*
Avg	*0.0076856*	*1.8619e-005*	*2.7954e-009*
#	*393*	*105*	*105*
Koordinate	*(-9.1259e-010 , -4.2099e-010)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.11 Ovisnost GBest o broju čestica za funkciju Schaffer_f6

Preciznost rješenja i brzina izvođenja rastu s porastom broja čestica.

- SCHAFFER_F6 MODIFIED

Broj čestica	30	500	2000
GBest	*0.39506*	*0.39506*	*0.39506*
Avg	*0.39682*	*0.39506*	*0.39507*
#	*105*	*105*	*105*
Koordinate	*(0, 0)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.1.12 Ovisnost GBest o broju čestica za funkciju Schaffer_f6 modified

Preciznost rješenja raste s porastom broja čestica.

- TRIPOD

Broj čestica	30	500	2000
GBest	*20*	*20*	*20*
Avg	*20.0333*	*20*	*20*
#	*105*	*101*	*101*
Koordinate	*(0, -30)*	*(0, -30)*	*(0, -30)*

Tablica 6.1.1.13 Ovisnost GBest o broju čestica za funkciju Tripod

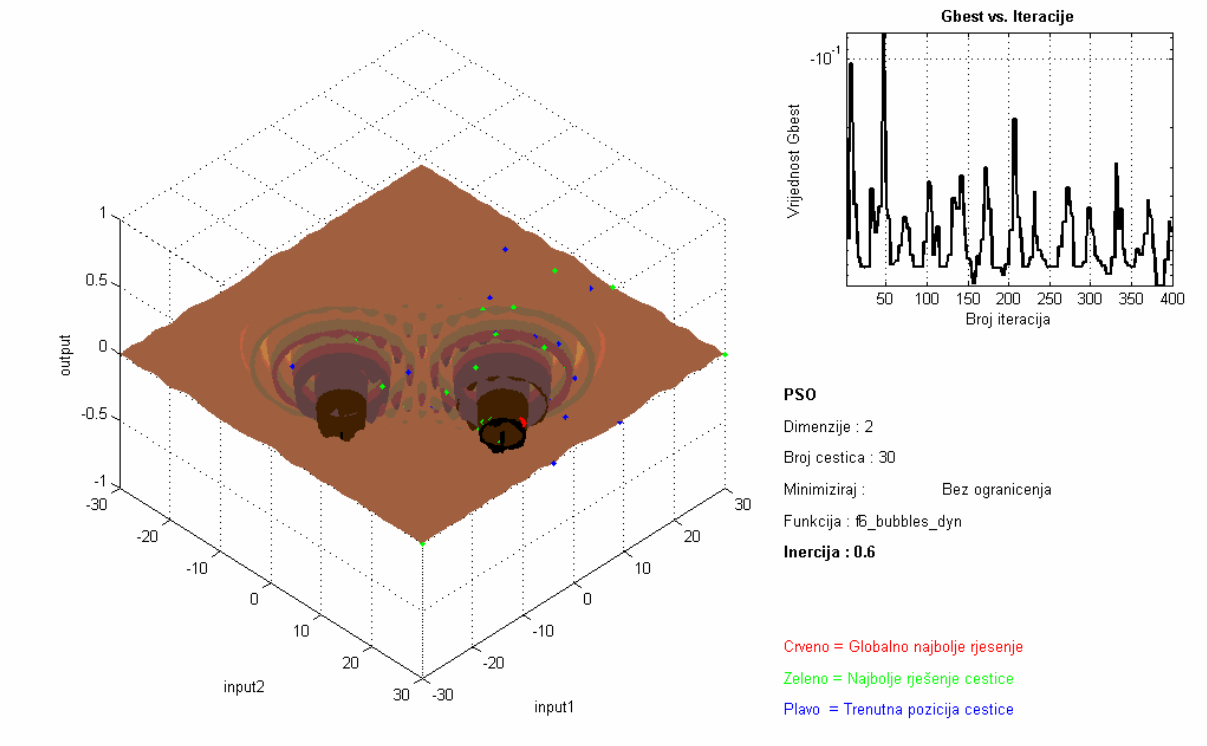
Preciznost rješenja i brzina izvođenja rastu s porastom broja čestica.

DINAMIČKE FUNKCIJE:

- F6_BUBBLES_DYN

Broj čestica	30	500	2000
GBest	*-0.99966*	*-0.90075*	*-0.87177*
Avg	*-0.93891*	*-0.8692*	*-0.88436*
#	*234*	*400*	*226*
Koordinate	*(-7.998, -8.0004)*	*(-7.9955, -7.9955)*	*(7.9919, 7.9966)*

Tablica 6.1.1.14 Ovisnost GBest o broju čestica za funkciju F6_Bubbles



Slika 6.1.1.3 Prikaz topologije greške i dinamike čestica za minimum f6_bubbles_dyn funkcije

- F6_LINEAR_DYN

Broj čestica	30	500	2000
GBest	*0.00012288*	*0.037177*	*0.016897*
Avg	*0.011221*	*0.015065*	*0.017505*
#	*400*	*400*	*400*
Koordinate	*(0.2263, 0.22686)*	*(0.6436, 0.59364)*	*(1.4651, 1.6249)*

Tablica 6.1.1.15 Ovisnost GBest o broju čestica za funkciju F6_linear

- F6_SPIRAL_DYN

Broj čestica	30	500	2000
GBest	*0.004821*	*1.0101e-005*	*0.012033*
Avg	*0.015722*	*0.0054117*	*0.0057456*
#	*400*	*400*	*400*
Koordinate	*(0.069726 , -0.0047467)*	*(0.051837 , 0.089657)*	*(-0.19582 , 0.062389)*

Tablica 6.1.1.16 Ovisnost GBest o broju čestica za funkciju F6_spiral

Dobiveni rezultati mogu se uspoređivati po grupacijama funkcija koje su promatrane. Tako za većinu statičkih funkcija vrijedi da preciznost globalno najprikladnijeg rješenja raste s brojem čestica (vrijednost Avg u tablici pada). Razlog tome je da više čestica može istražiti isti prostor detaljnije nego manji broj čestica. Iz tog razloga funkcije nađu ekstrem i prije isteka početno zadanih 400 iteracija pa zaključujemo da je vrijeme izvođenja manje nego s manjim brojem čestica (vrijednost # u tablici pada).

S druge strane za dinamičke funkcije ne možemo tvrditi isto. Naime dinamičkim se funkcijama u svakoj iteraciji promijeni okruženje. Teoretski se u svakoj iteraciji može promijeniti i vrijednost rješenja, tako da dinamičke funkcije gotovo nikada neće završiti nalaženje ekstrema prije isteka početno zadanih 400 iteracija. S obzirom da se okruženje stalno mijenja, možemo tvrditi da se prostor povećenjem broja čestica detaljnije pretražuje, no ne možemo reći da je postupak brži kao što to možemo tvrditi kod statičkih funkcija.

6.1.2. Maksimizacija

STATIČKE FUNKCIJE:

- ACKLEY

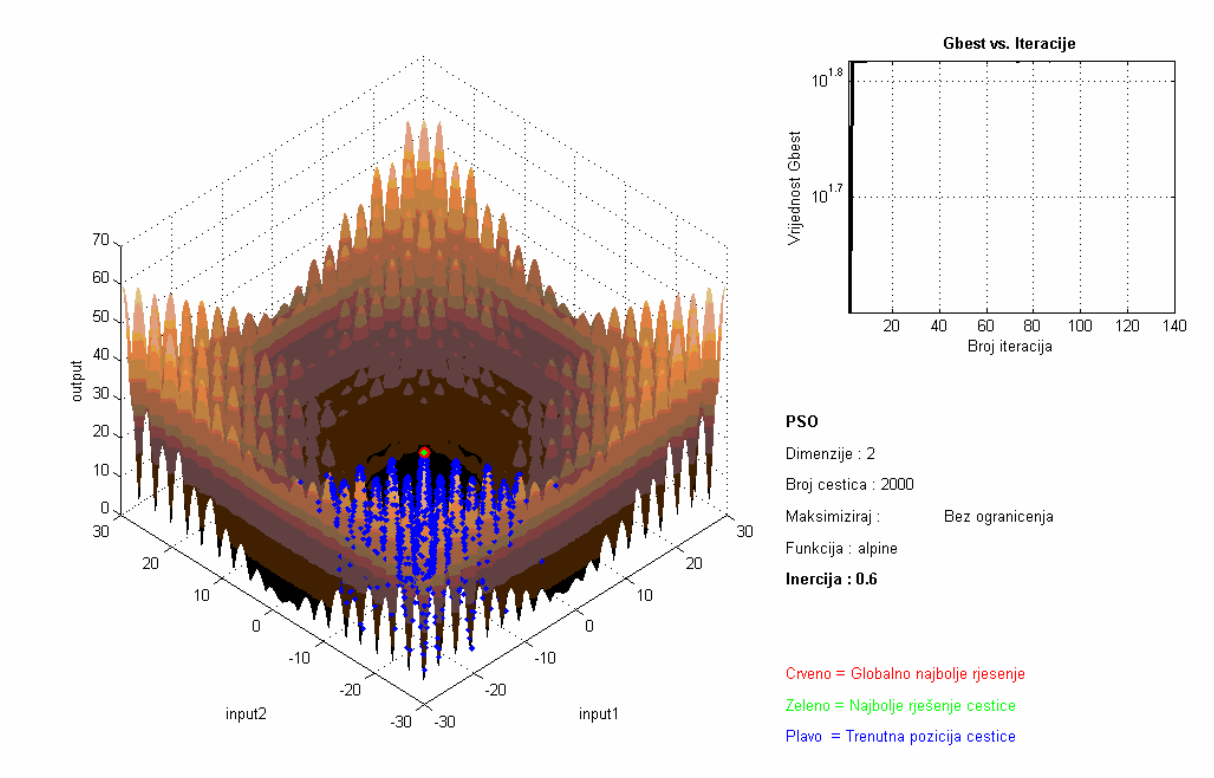
Broj čestica	30	500	2000
GBest	*22.2828*	*22.2956*	*22.2956*
Avg	*22.2235*	*22.2829*	*22.2891*
#	*201*	*400*	*398*
Koordinate	*(29.4416, 29.4963)*	*(29.5008, -29.5008)*	*(-29.5008, 29.5008)*

Tablica 6.1.2.1 Ovisnost GBest o broju čestica za funkciju Ackley

- ALPINE

Broj čestica	30	500	2000
GBest	*65.4726*	*65.6998*	*65.6998*
Avg	*64.9643*	*65.4869*	*65.5138*
#	*120*	*385*	*400*
Koordinate	*(-29.847, -30)*	*(-29.8819, -29.8819)*	*(-29.8819, -29.8819)*

Tablica 6.1.2.2 Ovisnost GBest o broju čestica za funkciju Alpine



Slika 6.1.2.1 Prikaz topologije greške i dinamike čestica za maksimum Alpine funkcije

- DEJONG_F2

Broj čestica	30	500	2000
GBest	*86490961*	*86490961*	*86490961*
Avg	*85653050.5756*	*86026843.2679*	*86026843.2679*
#	*105*	*103*	*103*
Koordinate	*(-30, -30)*	*(-30, -30)*	*(-30, -30)*

Tablica 6.1.2.3 Ovisnost GBest o broju čestica za funkciju Dejong_f2

- DEJONG_F3

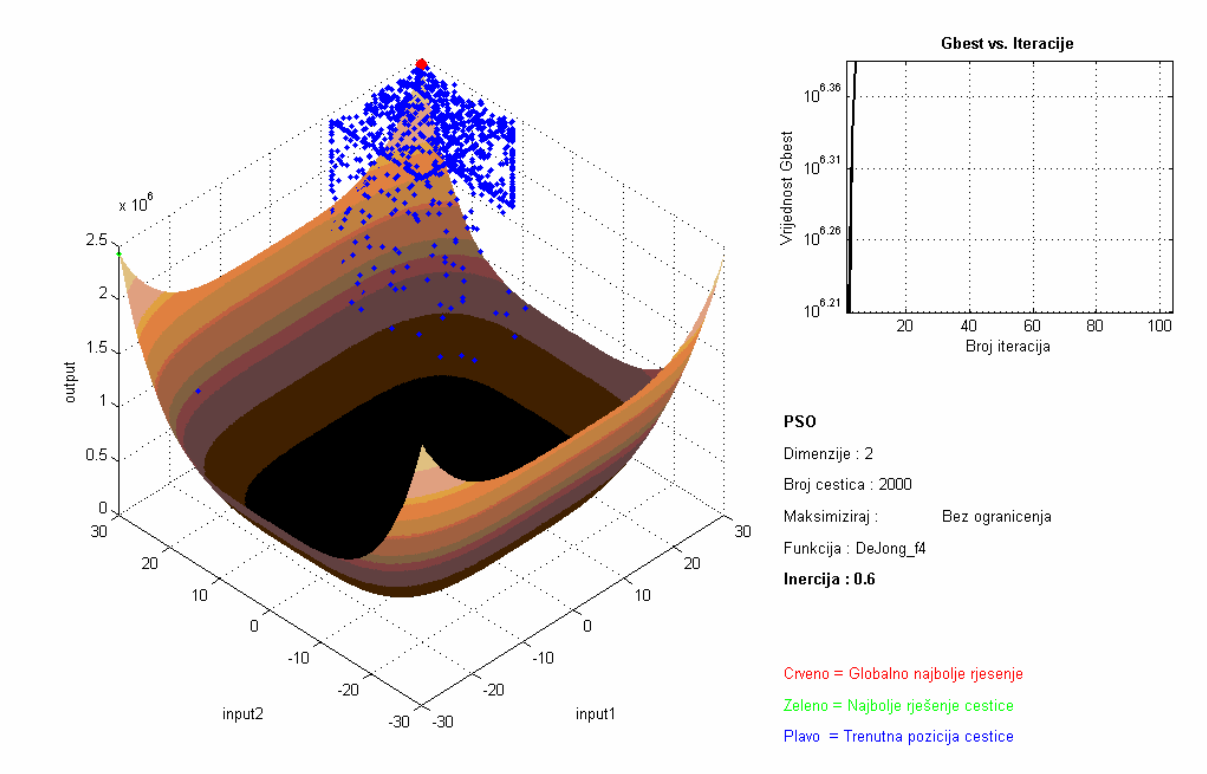
Broj čestica	30	500	2000
GBest	*86490961*	*86490961*	*86490961*
Avg	*85653050.5756*	*86026843.2679*	*86026843.2679*
#	*105*	*103*	*103*
Koordinate	*(-30, -30)*	*(-30, -30)*	*(-30, -30)*

Tablica 6.1.2.4 Ovisnost GBest o broju čestica za funkciju Dejong_f3

- DEJONG_F4

Broj čestica	30	500	2000
GBest	*2430000*	*2430000*	*2430000*
Avg	*2418026.0502*	*2413042.7986*	*2411744.625*
#	*103*	*104*	*104*
Koordinate	*(30, 30)*	*(30, -30)*	*(30, -30)*

Tablica 6.1.2.5 Ovisnost GBest o broju čestica za funkciju Dejong_f4



Slika 6.1.2.2 Prikaz topologije greške i dinamike čestica za maksimum Dejong_f4 funkcije

- FOXHOLE

Broj čestica	30	500	2000
GBest	*0.14486*	*0.14486*	*0.14486*
Avg	*0.14486*	*0.1444*	*0.14448*
#	*363*	*387*	*313*
Koordinate	*(-15.9753, -15.9753)*	*(-15.9753, -15.9753)*	*(-15.9753, -15.9753)*

Tablica 6.1.2.6 Ovisnost GBest o broju čestica za funkciju Foxhole

- GRIEWANK

Broj čestica	30	500	2000
GBest	*10.1042*	*10.1042*	*10.1042*
Avg	*10.0254*	*10.0954*	*10.091*
#	*400*	*366*	*331*
Koordinate	*(-25.7269, -28.973)*	*(-25.7269, -28.973)*	*(-25.7269, -28.973)*

Tablica 6.1.2.7 Ovisnost GBest o broju čestica za funkciju Griewank

- 2D PARABOLA

Broj čestica	30	500	2000
GBest	*1800*	*1800*	*1800*
Avg	*1790.8163*	*1785.2147*	*1788.6676*
#	*103*	*103*	*103*
Koordinate	*(30, 30)*	*(30, 30)*	*(-30, 30)*

Tablica 6.1.2.8 Ovisnost GBest o broju čestica za funkciju 2D parabola

- RASTRIGIN

Broj čestica	30	500	2000
GBest	*1800*	*1800*	*1800*
Avg	*1799.6372*	*1789.9038*	*1793.0504*
#	*103*	*103*	*103*
Koordinate	*(-30, 30)*	*(-30, -30)*	*(30, 30)*

Tablica 6.1.2.9 Ovisnost GBest o broju čestica za funkciju Rastrigin

- ROSENBROCK

Broj čestica	30	500	2000
GBest	*86490961*	*86490961*	*86490841*
Avg	*86081117.0139*	*86041188.0614*	*86080129.1946*
#	*104*	*103*	*103*
Koordinate	*(-30, -30)*	*(-30, -30)*	*(30, -30)*

Tablica 6.1.2.10 Ovisnost GBest o broju čestica za funkciju Rosenbrock

- SCHAFFER_F6

Broj čestica	30	500	2000
GBest	*0.97643*	*0.97643*	*0.97643*
Avg	*0.97604*	*0.97641*	*0.97643*
#	*383*	*343*	*400*
Koordinate	*(-1.2555, -0.91855)*	*(0.16329, -1.547)*	*(0.47132, 1.4825)*

Tablica 6.1.2.11 Ovisnost GBest o broju čestica za funkciju Schaffer_f6

- SCHAFFER_F6 MODIFIED

Broj čestica	30	500	2000
GBest	*0.59821*	*0.59821*	*0.59821*
Avg	*0.59758*	*0.5982*	*0.59821*
#	*327*	*288*	*386*
Koordinate	*(-7.0287e-008, -1.5702)*	*(-4.3322e-0, 1.5702)*	*(-1.6455e-008, 1.5702)*

Tablica 6.1.2.12 Ovisnost GBest o broju čestica za funkciju Schaffer_f6 modified

- TRIPOD

Broj čestica	30	500	2000
GBest	*102*	*102*	*102*
Avg	*101.9499*	*101.9982*	*101.9991*
#	*105*	*105*	*105*
Koordinate	*(0, 0)*	*(0, 0)*	*(0, 0)*

Tablica 6.1.2.13 Ovisnost GBest o broju čestica za funkciju Tripod

DINAMIČKE FUNKCIJE:

- F6_BUBBLES_DYN

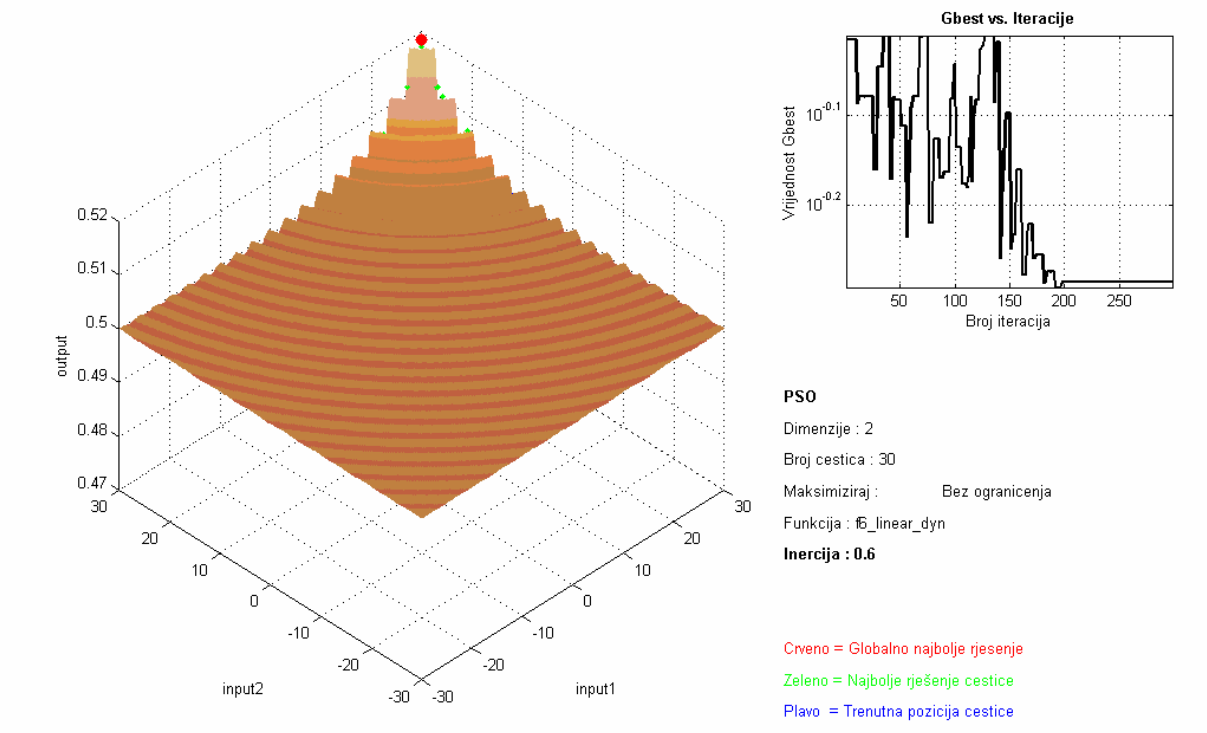
Broj čestica	30	500	2000
GBest	*0.91923*	*0.85675*	*0.87135*
Avg	*0.89887*	*0.83234*	*0.80118*
#	*400*	*400*	*400*
Koordinate	*(7.695, 9.5254)*	*(-9.5254, -7.695)*	*(-9.5254, -7.695)*

Tablica 6.1.2.14 Ovisnost GBest o broju čestica za funkciju f6_bubbles

- F6_LINEAR_DYN

Broj čestica	30	500	2000
GBest	*0.97643*	*0.97643*	*0.97643*
Avg	*0.97073*	*0.97643*	*0.97643*
#	*166*	*280*	*400*
Koordinate	*(1.5893, -0.094597)*	*(1.3677, 1.1668)*	*(2.0422, -0.15109)*

Tablica 6.1.2.15 Ovisnost GBest o broju čestica za funkciju f6_linear



Slika 6.1.2.3 Prikaz topologije greške i dinamike čestica za maksimum f6_linear_dyn funkcije

- F6_SPIRAL_DYN

Broj čestica	30	500	2000
GBest	*0.97643*	*0.97643*	*0.97643*
Avg	*0.97457*	*0.97642*	*0.97643*
#	*346*	*260*	*201*
Koordinate	*(0.97857, -1.2218)*	*(-1.308, -0.78751)*	*(-1.0297, 1.153)*

Tablica 6.1.2.15 Ovisnost GBest o broju čestica za funkciju f6_spiral

Kod pronalaženja maksimuma, vidljivo je da se ne može generalizirati kao što je to bio slučaj kod pronalaženja minimuma. Naime kod traženja maksimuma statičkih funkcija imamo slučaj da se sa svim testnim uzorcima broja čestica dobe jednake maksimumi i to gotovo jednake preciznosti. Čak je i brzina izvođenja slična i nema drastične razlike, ne može se čak ni generalizirati i reći da brzina izvođenja isključivo raste ili isključivo pada, nego uvijek ustaje u krugu nekoliko desetaka iteracija.

6.2. Buduća testiranja

Iako su rezulati kod traženja minimuma očekivani, kod traženja maksimuma očivao sam veće razlike. U budućnosti je potrebno provesti još nekoliko mjerenja za svaku funkciju i proučiti ponašanje preciznosti i brzine izvođenja.

Također izvesti više mjerenja za dinamičke funkcije, bolje proučiti način mijenjanja okoliša te testiranjem otkriti koja je optimalna količina čestica za optimizaciju za pojedinu funkciju.

Osim tih potrebno je još promotriti i kako se ponaša sustav ako se mijenja inercija na niže i više vrijednosti nego sada (0.9; 0.6) te koji je utjecaj kognitivnih parametara na sustav prilikom optimizacije.

Nakon tih mjerenja, provesti testiranje u kojem bi se svi parametri mijenjali te na taj način proučiti međusobnu ovisnost svih parametara.

7. Zaključak

Od kada se PSO razvio prije otprilike 13 godina, autori vjerojatno nisu ni mogli zamisliti koliki će utjecaj taj algoritam imati na današnji računarski svijet. Idejno maštovit i kreativan, gotovo multidisciplinaran, no istovremeno, u svojoj suštini, prirodan te, najbitnije, programski lagan za implementaciju, PSO oduševljava svojim mogućnostima implementacije u najrazličitijim područjima ljudskog djelovanja, od biologije i medicine, preko elektronike, elektromagnetizma, kombinatoričkih problema, analize sustava do robotike. Ono što je posebno zanimljivo je trend rasta primjena ovog algoritma. U posljednjih nekoliko godina primjene ovog algoritma narasle su eksponencijalno i ne čini se da trend trenutno pada.

Zbog svoje jednostavnosti i jednostavne mogućnosti prilagodbe, PSO je lako testirati te ostvariti novu verziju koja više odgovara specifičnom optimizacijskom problemu.

8. Literatura

[1] Maurice Clerk, TRIBES a Parameter Free Particle Swarm Optimizer, 2.10.2003., *TRIBES a Parameter Free Particle Swarm Optimizer*, http://clerc.maurice.free.fr/pso/Tribes/Tribes_doc.zip, 28.10.2008.

[2] Particle swarm optimization - Wikipedia, the free encyclopedia, 16.10.2008., *Particle swarm optimization - Wikipedia, the free encyclopedia,* http://en.wikipedia.org/wiki/Particle_Swarm_Optimization, 28.10.2008.

[3] Goran radanović, Pregled heurističkih algoritama, *Pregled heurističkih algoritama*, http://www.zemris.fer.hr/~golub/ga/studenti/seminari/2007_radanovic/index.html, 16.10.2008.

[4] C. K. Mohan, E. Ozcan, Particle Swarm Optimization, *Particle Swarm Optimization Homepage*, http://www.cis.syr.edu/~mohan/pso/, 17.10.2008.

[5] Xiaohui Hu, Particle Swarm Optimization, 1.1.2005., *Particle Swarm Optimization*, http://www.swarmintelligence.org/index.php, 17.10.2008.

[6] Maurice Clerk, pso, 31.10.2008., *pso*, http://clerc.maurice.free.fr/pso/, 3.11.2008.

[7] Kennedy, J.; C. Eberhart, R.; Shi, Y. Swarm intelligence.

[8] E. Perez, R.; Behdinan, K. Particle Swarm Optimization in Structural Design

[9] Ekstremi funkcije više varijabli, *Ekstremi funkcije više varijabli*, http://www.grad.hr/nastava/matematika/mat2/node8.html, 13.12.2008.

[10] Minima and Maxima - Wikipedia, the free encyclopedia, *Minima and Maxima*, http://en.wikipedia.org/wiki/Maxima_and_minima, 10.12.2008.

[11] Technical reports, *Technical reports*, http://cswww.essex.ac.uk/technical-reports/2007/tr-csm469.pdf, 2.12.2008.

[12] Main, 30.4.2005., *PSO Toolbox*, http://psotoolbox.sourceforge.net/, 2.12.2008.

U radu je testiran program PSO Toolbox preuzet sa stranice http://psotoolbox.sourceforge.net/. [12]