Optimizacija maksimalne k-zadovoljivosti populacionim metaheuristikama

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet, Beograd

David Dimić, Zorana Gajić
daviddimic@hotmail.com, zokaaa_gajich@bk.ru

Maj 2019.

Sažetak

U ovom radu biće objedinjene, implementirane i upoređene dve grupe algoritama za rešavanje optimizacionog problema maksimalne k-zadovoljivosti zasnovane na populacijama - evolucioni algoritmi razvijeni u nekoliko različitih varijanti SAWEA, RFEA, FlipGA, ASAP i jedan novi algoritam MASAP, kao i grupa rojevih čestica PSO sa verzijama PSO-LS, PSOSAT i WPSOSAT. Biće iznete njihove osobine, prednosti, mane i unapređenja, da bi se na kraju, na konkretnim rezultatima implementiranih algoritama pokazale njihove razlike i mogućnosti.

Sadržaj

1 Uvod

Na početku, potrebno je jasno formulisati problem da bi se pristupilo njegovom rešavanju. Problem maksimalne k-zadovoljivosti može se definisati na sledeći način: Neka je data je formula F u KNF obliku sa n promenjivih $(x_1, x_2, ..., x_n)$ i m klauza. Klauza C_i dužine k je disjunkcija k literala: $C_i = (x_1 \vee x_2 ... \vee x_k)$, gde je svaki literal promenljiva ili njegova negacija i može se pojavljivati više puta u izrazu. Cilj je pronaći istinitosne vrednosti promenljivih, valuaciju koja je vektor $\vec{v} = (x_1, x_2, ..., x_n) \in \{0, 1\}^n$ tako da ova valuacija maksimizuje broj zadovoljenih klauza u formuli F.

Ako valuacija zadovoljava formulu, onda se ona naziva modelom formule F. Max k-SAT problem može se definisati parom (Ω, SC), gde je Ω skup svih potencijalnih rešenja iz $\{0, 1\}^n$, vektor n promenljivih, a $SC : \Omega \to \mathbb{N}$, skor valuacije koji je jednak broju zadovoljenih klauza. Shodno ovome, problem max k-SAT je naći $\vec{v} \in \Omega$ za koje je SC maksimalno:

$$\max_{\vec{v} \in \Omega} \{SC(\vec{v})\}$$

Očigledno, ima 2^n potencijalnih rešenja koji zadovoljavaju formulu F. Dokazano je da je problem max k-SAT je NP-kompletan za svako $k > 2$ [4].

Kroz dalja poglavlja pristupiće se rešavanju ovako formulisanog problema prvo evolucionim algoritmima (EA) 2, gde će biti reči o osnovnim osobinama, biće dat njegov pseudokod i objašnjenja svake od akcija koje se preduzimaju. Potom će biti iznete nekoliko verzija algoritma [1] i njihova unapređenja u delu 2.8, kao i jedan originalni doprinos. Mogućnosti EA biće dati tabelarno i grafički u rezultatima 2.9. U drugom delu biće predstavljena optimizacija rojem čestica (PSO) 3 u njegovoj diskretnoj formi sa nekoliko različitih verzija 3.2 koji će biti upoređeni u rezultatima 3.3. U poslednjem poglavlju 4 će se uporediti EA nasuprot PSO i doneti krajnji zaključci.

2 Evolucioni algoritmi (EA)

Evolucioni algoritmi [11] bazirani su na Darvinovoj teoriji evolucije, u kojoj unutar jedne populacije najčešće opstaju najbolje prilagođene jedinke. Reprezentacija jedinke naziva se hromozom ili genotip. Cilj je naći vrednost za koju zadata funkcija cilja dostiže svoj ekstremum ili vrednost koja je dovoljno blizu ekstremuma. Hromozomi su obično predstavljeni nizovima nula i jedinica, ali moguće su i druge reprezentacije. Postupak se odvija kroz generacije.

Tokom izvršavanja algoritma u svakoj generaciji postoji isti broj jedinki i za svaku od njih izračunava se njihov kvalitet, odnosno funkcija prilagođenosti. Ova funkcija naziva se i fitnes funkcija. Iz jedne generacije se na osnovu vrednosti fitnes funkcije, kroz proces selekcije biraju jedinke koje će biti iskorišćene za stvaranje novih jedinki. One kvalitetnije se biraju sa većom verovatnoćom. Videćemo različite tehnike selekcija kao što su turnirska i ruletska.

Nad izabranim jedinkama primenjuju se genetski operatori ukrštanja i tako se dobijaju nove jedinke. Ukrštanjem se od dve jedinke dobija nova (ili dve nove) sa genetskim materijalom koji je dobijen neposredno od roditelja, tj. od polaznih jedinki. U ovom radu korišćeno je samo uniformno ukrštanje.

Operatorom mutacije može da se modifikuje deo polazne jedinke i dobije nov genetski materijal. U svakoj generaciji može da dođe do rekombinacije gena zbog koje se javlja sličnost ali i različitost između jedinki iste generacije. Biće izložene različite

[1] Sve implementacije dostupne su na GitHub nalogu autora:
https://github.com/zokaaagajich/max-k-sat

vrste mutacija: slučajna, zasnovana na znanju, jednog bita i ostale.

Politika zamene generacija određuje kako se od postojećih jedinki i njihovog potomstva kreira nova generacija. Neke jedinke u novoj generaciji mogu biti bolje, a neke lošije od jedinki u prethodnoj generaciji.

Kriterijumi zaustavljanja mogu biti razni. U ovom radu korišćeno je zaustavljanje kada se dostigne maksimalan broj unapred zadatih iteracija, ili kada sve klauze formule F postanu zadovoljene. Tada je formula u potpunosti zadovoljena pa nema potrebe za daljom pretragom.

Naredni pseudokôd 1 sumira objašnjene faze osnovnog EA na kojem se dalje zasnivaju ostale varijante. U narednim poglavljima biće opisan svaki od pojedinačnih delova koji čini ovaj algoritam, a koji će se nadalje koristiti: fitnes funkcija, selekcija, ukrštanje i mutacija, kao i neke od njihovih adaptacija.

Input : Formula F u KNF-u, n i m
Output: Najbolja procenjena valuacija i broj zadovoljenih klauza

```
inicijalizacija populacije;
t = 0; // tekuća iteracija
while nije zadovoljen uslov zaustavljanja do
    t = t + 1;
    Određivanje funkcije prilagođenosti svake jedinke;
    Selekcija jedinki za primenu genetskih operatora;
    Ukrštanje za izabrane parove jedinki;
    Mutacija izabranih jedinki;
end
```

Algoritam 1: Osnovni evolutivni algoritam

2.1 Inicijalizacija rešenja

Populaciju jedinki jedne generacije čini skup hromozoma, odnosno rešenja problema. Hromozome možemo predstaviti na različite načine, ali će ovde biti korišćeno prirodno predstavljanje nizom binarnih cifara. Potrebno je da inicijalizujemo hromozome pseudoslučajnim brojevima $\{0, 1\}$. Veoma važnu ulogu igra odabir parametara. Parametri mogu biti fiksni ili im se vrednosti mogu na određen način menjati iz generacije u generaciju.

2.2 Fitnes funkcija

U različitim verzijama EA koriste se različite funkcije prilagođenosti, koja je od suštinske važnosti za dobro rešavanje problema. Prva fitness funkcija koja se sama nameće jeste broj zadovoljenih klauza, koja će biti upotrebljena u FlipGA i ASAP algoritmima.

$$f_{MAXSAT}(x) = \sum_{k=1}^{N} f(x, c_k)$$

gde je N ukupan broj klauza, x predstavlja jedan hromozom i

$$f(x, c_k) = \begin{cases} 1, & \text{ako je klauza } c_k \text{ zadovoljena hromozomom } x \\ 0, & \text{inače.} \end{cases}$$

Nešto komplikovanije funkcije prilagođenosti su SAW i REF koje se koriste u SAWEA i RFEA algoritmima.

2.2.1 SAW

Eiben i van der Hauw su razvili evolutivni algoritam koji koristi stepenasto prilagođavanje težina (eng. *stepwise adaptation of weights*) [7]. Svakoj klauzi c_i pridružuje se težina w_i, u početku inicijalizovana sa jedan. Ove težine se potom, posle određenog vremena, ažuriraju po pravilu:

$$w_i = w_i + \Delta w$$

gde je Δw:

$$\Delta w = 1 - f(x^*, c_i)$$

gde f označava da li je klauza c_i zadovoljena trenutno najboljim hromozomom x^*. Stoga, Δw može biti samo 1, ako c_i nije zadovoljena, ili 0, ako jeste. Iz ovoga se vidi da se težine nezadovoljenih klauza vremenom uvećavaju i time se pretraga usmerava ka njima.

Finalno za jedan hromozom i definišemo funkciju prilagođenosti kao:

$$f_{SAW}(i) = w_1 f(i, c_1) + ... + w_N f(i, c_N) = \sum_{k=1}^{N} w_k f(i, c_k)$$

2.2.2 REF

Gotlieb i Vos [6] predstavili su koncept funkcija prerada (eng. *refining functions*) koje su nastale iz želje da se prevaziđe mana obične funckije prilagođenosti koja često vraća isti rezultat za različite hromozome. Definišemo je koristeci f_{MAXSAT} za hromozom x na sledeći način:

$$f_{REF}(x) = f_{MAXSAT}(x) + \alpha r(x)$$

gde je $\alpha > 0$ predstavlja nivo uticanja funkcije r, $r : \{0,1\}^n \to [0,1]$. Za $\alpha = 0$ f_{REF} se ponaša kao f_{MAXSAT}.

Funkcija r za hromozom x se definiše na sledeći način:

$$r(x) = \frac{1}{2}(1 + \frac{\sum_{i=1}^{N} K(x_i)v_i}{1 + \sum_{i=1}^{N} |v_i|})$$

gde $K(0) = -1, K(1) = 1$. Autori su predložili da ažuriranje v_i izgleda ovako:

$$v_i = v_i - K(x^*) \sum_{k \in U_i(x^*)} w_k$$

gde je x^* trenutno najbolja jedinka, $U_i(x^*)$ je skup nezadovoljenih klauza u kojima se pojavljuje promenljiva i, w_k je definisano na isti način kao i u 2.2.1. v_i predstavlja težine promenljivih. Visoke pozitivne težine ukazuju da se za odgovarajuće promenljive preferira da budu 1, dok za negativne 0. Inicijalno, sve težine promenljivih su postavljene na vrednost 0.

2.3 Selekcija

Selekcija predstavlja izbor jedinki iz trenutne populacije koje će biti korišćene za dobijanje naredne generacije. Ona obezbeđuje čuvanje i prenošenje dobrih osobina populacije. U ovom radu koriste se dve najpopularnije strategije selekcije: ruletska i turnirska.

2.3.1 Ruletska selekcija

Ruletska selekcija (eng. *roulette wheel selection*) [11] je proces selekcije u kojem veće šanse da učestvuju u reprodukciji imaju prilagođenije jedinke. Ako $f(i)$ vrednost funkcije prilagođenosti za jedinku i , a N broj jedinki u populaciji, verovatnoća da će jedinka i biti izabrana da učestvuje u reprodukciji je:

$$p_i = \frac{f(i)}{\sum_j^N f(j)}$$

Ovako definisana ruletska selekcija imaće primenu u FlipGA algoritmu.

2.3.2 Turnirska selekcija

U turnirskoj selekciji [11] jedinke odigravaju turnire u kojima veće šansu za pobedu imaju one sa boljom prilagođenošću. Za jedan turnir bira se slučajno k jedinki iz populacije. Pobednikom se smatra jedinka sa najvećom prilagođenošću. Turnirska selekcija primenjuje se u RFEA.

2.4 Ukrštanje

Ukrštanje [11] je proces u kome učestvuju dve jedinke koje se nazivaju roditelji. Rezultat ukrštanja je jedna nova jedinka ili dve nove jedinke koje nazivamo decom, koje za cilj imaju kombinovanje gena oba roditelja radi postizanja boljih osobina. U FlipGA algoritmu biće korišćeno uniformno ukrštanje. Uniformno ukrštanje daje dva deteta. Kod ovog ukrštanja svaki bit prvog roditelja se sa verovatnoćom p prenosi na prvo dete, a sa verovatnoćom $1 - p$ na drugo dete. Za verovatnoću p uzeta je vrednost 0.5.

Tabela 1: Uniformno ukrštanje

Dva roditelja

1	0	0	0	1	1
0	0	1	1	0	1

Dva deteta

1	0	1	0	0	1
0	0	0	1	1	1

2.5 Mutacija

Mutacija [11] se primenjuje nakon operatora ukrštanja (ako se on uopšte i primenjuje). To je operator koji sa određenom verovatnoćom menja jedan deo jedinke na određeni način. Uloga mutacije je da spreči da jedinke u populaciji postanu suviše slične i da pomogne u obnavljanju izgubljenog genetskog materijala. U narednom delu biće izložene neke od mutacija koje će se koristiti u različitim varijantama EA algoritama, koji će biti objašnjeni u poglavlju 2.8.

2.5.1 Slučajna mutacija

Slučajna mutacija (eng. *random mutation*) primenjuje se nad celim hromozomom tako što za svaki gen na slučajan način bira vrednost $t \in [0, 1]$ i ako je ta vrednost manja od zadatog parametra mutiranja *mutation_rate* tj. $t < mutation\_rate$ onda se vrši invertovanje tog gena. Ovakav operator mutiranja biće korišćen u FlipGA algoritmu 2.8.3.

2.5.2 Mutacija jednog bita

Za razliku od slučajne mutacije koja je prolazila kroz ceo hromozom i menjala gene u zavisnosti od slučajnog broja, u mutaciji jednog bita (eng. *mutation one operator*) bira se proizvoljno tačno jedan gen koji se potom menja. Ova mutacija koristiće se u SAWEA verziji 2.8.1.

2.5.3 Slučajno-prilagodljiva mutacija

Slučajno-prilagodljiva mutacija (eng. *random-adaptive mutation*) [1] je u svojoj srži zapravo slučajna mutacija sa jednom izmenom: neki geni hromozoma mogu biti ,,zamrznuti" radi očuvanja dobrog rešenja do kog se došlo, dok se ostali, za koje se utvrdilo

da ne vode do rešenja, mogu menjati.

Ova ideja uvodi se u ASAP algoritmu 2.8.4 gde će biti detaljnije obrađena. Dakle, ova mutacija radi isto kao slučajna mutacija, osim kada naiđe na ,,zamrznuti" gen. U tom slučaju ne radi ništa i prelazi na sledeći gen hromozoma.

2.5.4 Mutacija zasnovana na znanju

Kod mutacije zasnovane na znanju (eng. *knowledge-based mutation*) [5] pokušavamo da iskoristimo nešto što znamo o trenutnom rešenju i da ne vršimo mutaciju na potpuno slučajan način. Mutaciju ćemo vršiti nad onim hromozomima koje nismo zadovoljili, jer će možda, posle mutacija nad njima, postati zadovoljene i time uvećati broj ukupno zadovoljenih klauza.

Za dati hromozom, koji predstavlja tekuće rešenje, možemo naći skup nezadovoljenih klauza. Iz skupa nezadovoljenih klauza na slučajan način biramo tačno jednu klauzu, a potom iz nje, ponovo, biramo jednu njenu promenljivu koju ćemu mutirati. Primena ove mutacije nalazi se u RFEA algoritmu 2.8.2.

2.5.5 Lamarckian SEA-SAW mutacija

Da bi se poboljšali SAWEA i RFEA algoritmi koristi se Lamarckian mutacija koju su predstavili de Jong i Kosters [3, 5]. Ovakav operator mutacije liči na lokalnu pretragu, naime, nakon dobijanja dece za novu populaciju bira se jedno dete c na slučajan način. Takođe, kreiramo skup slučajno odabranih klauza. Ako je svaka klauza u ovakvom skupu zadovoljena hromozomom c onda se ne radi ništa. Inače, izabere se jedna slučajna promenljiva u nezadovoljenoj klauzi i mutira se.

2.6 Politika zamene generacija

Jedna od podgrupa EA su evolucione strategije ES (eng. *Evolution Strategy*) koje koriste selekciju, mutaciju i neku od politika zamene. Politika zamene generacije [11, 3] opisuje kako se od tekuće generacije dobija nova. Osnovna podela po ovom kriterijumu je na generacijske genetske algoritme (μ, λ) (eng. *generational genetic algorithm*) i genetske algoritme stabilnog stanja $(\mu + \lambda)$ (eng. *steady state genetic algorithm*), gde μ predstavlja broj pojedinaca u populaciji, a λ broj dece.

U slučaju generacijskih genetskih algoritama, nova generacija se dobija tako što se selekcijom bira dovoljno jedinki iz tekuće generacije da se napravi cela nova generacija. Izabrane jedinke se ukrštaju i mutiraju i tako dobijena generacija zamenjuje staru. (μ, λ) - od μ roditelja kreira se λ dece i za novu generaciju uzimaju se najbolji među decom.

U slučaju genetskih algoritama stabilnog stanja, čim se izabere par roditelja, vrši se ukštanje i mutacija, a potom umetanje potomaka u populaciju u skladu sa nekom politikom zamene. $(\mu + \lambda)$ - od μ roditelja kreira se λ dece i biraju se najbolje jedinke iz skupa roditelja i dece zajedno.

2.7 Lokalna pretraga flip heuristikom

Da bi se unapredili standardni evolutivni algoritmi uvodi se stohastička lokalna pretraga heuristikom okretanja bitova (eng. *flip heuristic*) [2]. Ova heuristika zasniva se na izmeni pojedinačnih bitova u tekućem rešenju. Za svaki bit u hromozomu pokušava se njegovo okretanje. Ako sa tom izmenom ima dobiti (eng. *gain*), u smislu ne pogoršavanja funkcije cilja, onda se čuva nova vrednosti okrenutog bita, dok se u suprotnom vraća na njegovu staru vrednost. Čitav proces se ponavlja dokle god ima napretka, odnosno dok postoji bar jedna izmena koja je poboljšala tekuće rešenje, ili dok se ne

dostigne maksimalni dozvoljeni broj okretanja bitova *maxFlip*.

```
Input  : hromozom p, Formula F u KNF-u, maxFlip
Output: novi hromozom
improvement = 1;
numFlip = 0;
while improvement > 0 and numFlip < maxFlip do
    improvement = 0;
    for i ← 0 to n do
        flip p[i];
        numFlip += 1;
        Izračunaj dobit: gain;
        if gain >= 0 then
            prihvati flip;
            improvement += gain;
        end
        else
            odbaci flip, vrati na staru vrednost p[i];
        end
    end
end
```

Algoritam 2: Funkcija lokalne pretrage

Ovakva lokalna pretraga usporava celokupno izvršavanje algoritma, ali sa druge strane, umnogo ubrzava konvergenciju ka lokalnom optimumu. Ovu lokalnost razbijaju slučajni faktori ukrštanja i mutacije evolutivnih algoritama, u čijem se prisustvu obično koristi. Njena primena prvi put predstavljena je u FlipGA algoritmu o kome će biti reči u sledećem poglavlju.

2.8 Varijante EA algoritma

Da bismo dobili što bolje rešenje pokušavamo da kombinujemo različite tehnike operatora selekcije, ukrštanja i mutacije, koje su opisane u prethodnim poglavljima. Negde čak nećemo imate sve navedene operatore, već samo neke od njih. U tabeli 2 možemo da vidimo osnovna svojstva varijanti evolutivnog algoritma koji su implementirani, odnosno šta koristi svaki od pobrojanih algoritama. Zajedničko za sve vrste EA je reprezentacija rešenja kao niz bitova i inicijalizacija koja se vrši na slučajan način.

Tabela 2: Varijante EA algoritma

Svojstvo	SAWEA	RFEA	FlipGA	ASAP	MASAP
politika zamene	$(1, \lambda^*)$	stabilno stanje	generacijski	$(1+1)$	$(1+1)$
selekcija	-	turnirska	ruletska	-	-
fitnes	f_{SAW}	f_{REF}	f_{MAXSAT}	f_{MAXSAT}	f_{SAW}
ukrštanje	-	-	uniformno	-	-
mutacija	jednog bita	zasnovana na znanju	slučajna	slučajno-prilagodljiva	Lamarckian
lokalna pretraga	-	-	flip heuristika	flip heuristika	flip heuristika
adaptacija	fitnes	fitnes	-	tabu lista	tabu lista

U narednom delu biće opisani svaki od navedenih algoritama iz tabele 2. Obratiće se pažnja na njihove prednosti i mane, a praktični rezultati implementacija biće prikazani u poglavlju 2.9.

2.8.1 SAWEA

SAWEA je evolutivni algoritam koji koristi stepenasto prilagođavanje težina (eng. *Stepwise Adaptation of Weights*) [5, 3] koji su razvili Eiben i van der Hauw. Prema preporuci autora koristi se politika zamene generacije $(1, \lambda^*)$ 2.6 i mutacija jednog bita 2.5.2.

S obzirom da je politika zamene $(1, \lambda^*)$ znamo da je populacija veličine jedan, zbog čega nije moguće vršiti selekciju i ukrštanje, već samo mutaciju. Jedan od parametara ovog algoritma je i λ^* koji podrazumevano ima vrednost 10. Kreiranje nove generacije vrši se na sledeći način: uzima se jedan jedini mogući roditelj, λ^* puta primenjuje se operator mutacije da bi se dobilo najviše λ^* dece. Za novu generaciju bira se najbolji od njih, pri čemu je funkcija prilagođenosti f_{SAW}.

Klauzama su pridružene težine w_i čija je inicijalna vrednost 1 (primetimo da je tada $f_{SAW} = f_{MAXSAT}$). Potom se se, nakon određenog broja iteracija (uzeto je 10), ažuriraju, čime se akcenat stavlja na ,,teže" klauze.

Dalje poboljšanje ovog SAWEA algoritma izveli su Jong i Kosters [10]. Oni su uveli novi operator mutacije poznat kao Lamarckian-ova SEA-SAW mutacija koja je opisana u delu 2.5.5. Ovo unapređene pokazalo se kao bolje rešenje za SAWEA algoritam.

2.8.2 RFEA

Gotlieb i Vos [6, 5, 3] predstavili su koncept funkcija prerada (eng. *refining functions*) i uveli f_{REF} funkciju prilagođenosti opisanu u delu 2.2.2. RFEA koristi populaciju veličine četiri, turnirsku selekciju 2.3.2 veličine dva i politiku zamene generacije stabilnog stanja 2.6 sa eliminacijom najgore jedinke. Takođe je korišćena eliminacija duplikata, tj. dete je odbijeno ukoliko je već u populaciji. Jedini variacioni operator je mutacijski operator zasnovan na znanju 2.5.4. Pošto se koriste težine potrebno je nakon određenog vremena izvršiti njihovo ažuriranje.

2.8.3 FlipGA

FlipGA (eng. *Flipping Evolutionary Algorithm*) predstavili su Marchiori i Rossi 1999. godine [2, 5]. Ovo je zapravo klasičan evolutivni algoritam sa ukrštanjem i mutacijom koji je unapređen lokalnom pretragom 2.7, pa se zbog ovoga često naziva i Hibridni-EA. Koristi se funkcija prilagođenosti f_{MAXSAT} i ruletska selekcija 2.3.1.

FlipGA koristi generacijsku politiku zamene sa populacijom veličine 10. Primenjuje se strategija elitizma koja ostavlja najbolje dve jedinke iz trenutne generacije u narednu. Ovo je jedini algoritam u kojem se uvek vrši uniformno ukrštanje i klasičan operator mutacije. Da bi se izbegli lokalni optimumi parametar mutiranja ima visoku verovatnoću od 0.9. Ovaj algoritam, sa svojom lokalnom pretragom putem flip heuristike, dalje je unapređen u ASAP algoritam.

2.8.4 ASAP

ASAP (eng. *Adaptive ea for the SAtisfiability Problem*) [1, 5] nastaje kao dalje unapređenje FlipGA u cilju izbegavanja lokalnih optimuma i vremenskom ubrzanju algoritma koji su dali Rossi, Marchiori i Kok. Osnovna ideja je promeniti prostor pretrage prilagodljivim mehanizmom kada se utvrdi da je došlo do zaglavljivanja u lokalnom optimumu.

Ovo se izvodi pomoću tabu liste (zapravo matrica, jer je lista listi) u koju se smeštaju slični hromozomi koji imaju istu fitnes funkciju, a do kojih se došlo lokalnom pretragom. Očigledno da oni vode u jedan lokalni optimum, ali i među njima ima razlike. Kada se lista napuni vrši se poređenje gena duž cele tabu liste. Oni geni koji se ne menjaju verovatno su nas doveli do lokalnog optimuma i njih je potrebno menjati, dok ostale

treba zadržati tako što će njihova promena biti zabranjena. Takvi geni, koji nisu isti duž cele tabu liste, biće „zamrznuti". Zamrznute gene ne mogu izmeniti ni operator mutacije ni lokalna pretraga. Veličina tabu liste po preporuci autora je 10.

U tabeli 3 geni duž prve i četvrte kolone su svuda isti geni i te se kolone neće zamrznuti, odnosno u listu zamrznutih gena (eng. *frozen*) na pozicije 1 i 4 upisuje se nula, dok su na ostalim mestima, gde se geni menjaju, jedinice. Ova lista je inicijalno popunjena nulama.

Tabela 3: Tabu lista i zamrznuti geni

1	1	0	0	...	0
1	0	0	0	...	1
1	0	1	0	...	1
1	1	0	0	...	0
1	...	...	0	...	...
1	0	0	0	...	1
Lista zamrznutih gena					
0	1	1	0	...	1

Kada se fitnes najboljeg hromozoma poveća, tabu lista se prazni i svi geni se odmrzavaju. Takođe, hromozomi u listi su grupisani po klasama ekvivalencije, gde svaka klasa sadrži iste hromozome (sva rešenja u tabu listi imaju isti fitnes, ali ne moraju biti ista po genima). Kada je broj klasa vrlo mali, manji ili jednak od dva, to znači da je pretraga u potpunosti usmerena prema tim rešenjima, i potrebno je restartovati pretragu od novog slučajno generisanog hromozoma.

Input : otac $c0$, dete c, tabu lista T, lista zamrznutih gena
Output: lista zamrznutih gena

```
odmrzni sve gene;
if fitness(c0) > fitness(c) then
    //odbaci dete jer je roditelj bolji
    c = c0;
end
else
    if fitness(c) > fitness(c0) then
        isprazni T;
        dodaj c u T;
    end
    else
        //isti fitnes
        dodaj c u T;
        if T puna then
            izračunaj zamrznute gene;
            ažuriraj mutation_rate;
            izračunaj klase ekvivalencije;
            if brojKlasa <= 2 then
                RESTART;
            end
            isprazni T;
        end
    end
end
```

Algoritam 3: Ažuriranje tabu liste

Sa tehničke strane promenjena je politika zamene i koristi se (1 + 1) strategija

(jedan roditelj proizvodi jedno dete), pa je populacija veličine jedan. Ovo u startu dosta ubrzava algoritam i lokalna pretraga sada ne usporava rad. Kako je populacija veličine jedan, nije moguća selekcija ni ukrštanje, pa se kreiranje nove generacije vrši na sledeći način: uzima se jedini mogući roditelj, potom se slučajno-prilagodljivom mutacijom 2.5.3 dobija jedno dete. Na njega se primenjuje lokalna pretraga sa flip heuristikom (koja je takođe adaptivna - ne menja zamrznute gene), a potom se ažurira tabu lista. Pošto je $(1 + 1)$ u pitanju, nova generacija je ili roditelj ili dobijeno dete u zavisnosti od prilagođenosti izračunate sa f_{MAXSAT}.

2.8.5 MASAP

MASAP (eng. *Modified ASAP*) uveli su autori ovog rada kao svoj originalni doprinos i jedno dalje unapređenje ASAP algoritma. Kako se kasnije u rezultatima 2.9 videlo, ASAP se može kategorisati kao najbolji algoritam iz kategorije EA i PSO algoritama koji su prezentovani u ovom radu, pa je zato dalja modifikacija pokušana upravo na njemu.

Modifikovani ASAP zadržava populaciju veličine jedan i korišćenje lokalne pretrage i tabu liste - elemente koji su se pokazali kao ključni u efikasnosti ovog algoritma. Ali, uvodi drugu fitnes funkciju i mutaciju. Umesto f_{MAX} koristi težinsku funkciju f_{SAW} koja se zbog ažuriranja težina u nekim slučajevima pokazala kao značajno unapređenje. Takođe, umesto obične slučajne mutacije, MASAP koristi Lamarckian SEA-SAW mutaciju, koja je jedno unapređenje mutacije zasnovane na znanju. Ovako modifikovani algoritam koristi najbolje elemente od svih prethodno izloženih evolutivnih algoritama, pa je moguće očekivati poboljšanje u odnosu na ostale. Njegov rad biće prikazan u sledećem poglavlju.

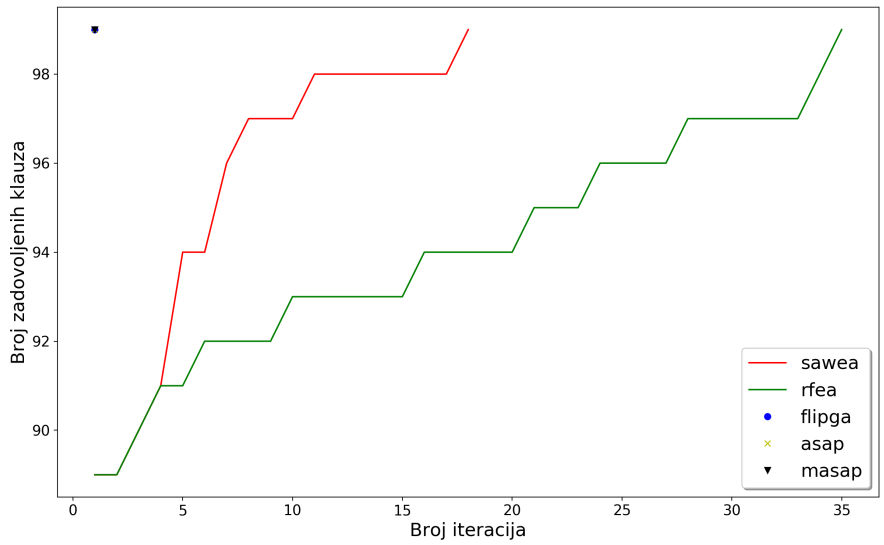
2.9 Rezultati

Svi algoritmi pokretani su pet puta sa istim parametrima i beležen je prosečan broj zadovoljenih klauza, pri čemu podebljan rezultat označava da se do rešenja dolazilo u prvoj iteraciji. Maksimalan broj iteracija postavljen je na 1000, a vrednost za $maxFlip$ u lokalnoj pretrazi FlipGA algoritma na 30000.

Prvo su pokrenute nezadovoljene instance gde su testovi takvi da je broj zadovoljenih klauza za jedan manji od ukupnog broja klauza. Pri ovim instancama u tabeli 4 prikazani su rezultati izvršavanja programa. Vidi se da su SAWEA i RFEA retko dolazili do rešenja, pri čemu se RFEA, očekivano, pokazao kao nešto bolji, sa cenom sporijeg izvršavanja. Dalje su, FlipGA, ASAP i MASAP u potpunosti ispunili očekivanja i rešili sve instance, pri čemu prednjače ASAP i MASAP gledajući vremenski faktor, kako je bilo najavljeno u opisu ovih algoritma 2.8.4. ASAP i MASAP koriste populaciju veličine jedan, u čemu leži njihovo ubrzanje i lokalnu pretragu u kombinaciji sa tabu listom za izbegavanje lokalnih optimuma. Zanimljivo je primetiti da se sa ova tri algoritma uglavnom u prvoj iteraciji dolazilo do rešenja. U proseku, iteracije MASAP-a su vremenski kraće, a dolazilo se do rešenja u kraćem broju iteracija.

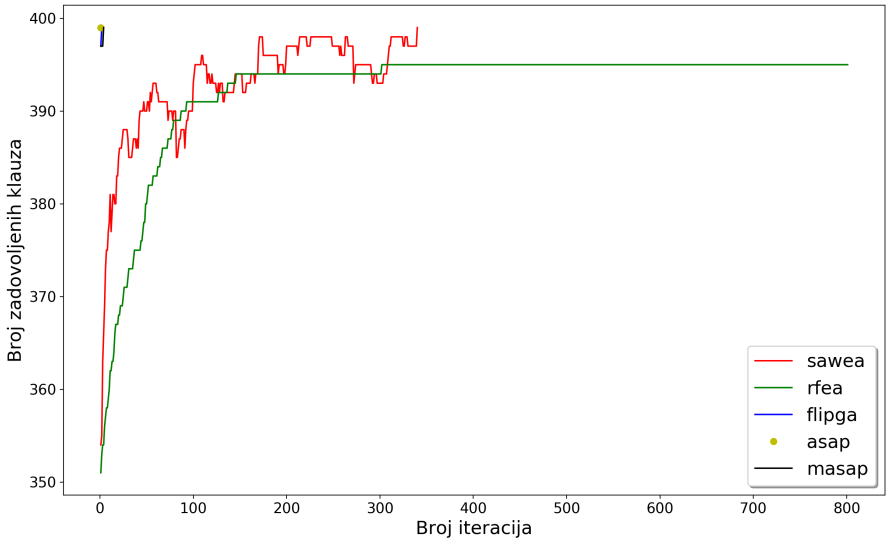
Tabela 4: AIM nezadovoljivi testovi

Instanca	Broj literala	Broj klauza	SAWEA	RFEA	FlipGA	ASAP	MASAP
aim-50-1_6-no	50	80	79	79	**79**	**79**	**79**
aim-50-2_0-no	50	100	98.6	99	**99**	**99**	**99**
aim-100-1_6-no	100	160	158.8	159	**159**	**159**	**159**
aim-100-2_0-no	100	200	197.8	199	**199**	**199**	**199**
aim-200-2_0-no	200	400	396	397.8	**399**	399	**399**



Slika 1: Broj zadovoljenih klauza kroz iteracije za EA aim-50-2_0-no

Za pokrenutu jednu nezadovoljivu instancu na slici 1 jasno se vidi razlika između ovih algoritama i njihov put ka pronalaženju rešenja. Najduži put imao je RFEA (1,48s) u 35 iteracija, potom SAWEA (0,08s) koji je vremenski najbrži. Najefikasniji su MASAP (0,07s) i ASAP (0,08s) koji su za jednu iteraciju došli do rešenja. FlipGA (0,19s) je nešto sporiji što je cena pozivanja lokalne pretrage nad većom populacijom.



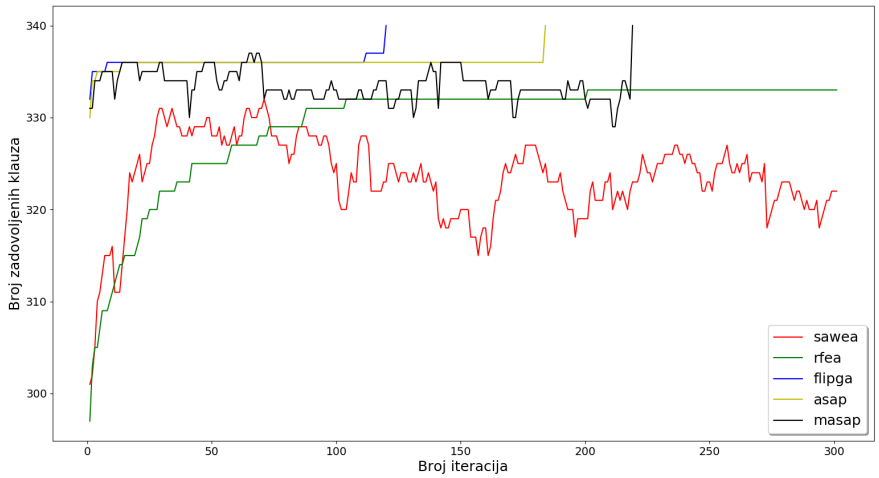
Slika 2: EA aim-200-2_0-no

Na slici 2 prikazano je rešavanje nešto teže nezadovoljive instance. I dalje su za čitavu klasu razlike FlipGA (5,50s), ASAP (0,50s) i MASAP (0,70s) pri čemu se još jasnije vidi značaj osnovnog ASAP unapređenja. RFEA (463,38s) pokazao se još gori za teže instance. On, istina, monotono raste pri uspehu u pretrazi, za razliku od SAWEA (0,93s) koji više nasumično luta, ali kada se pogleda vreme izvršavanja, to definitivno nije vredno. Takođe, SAWEA u svom lutanju uspeva da dođe do rešenja, za razliku od RFEA koji je očigledno dostigao svoj lokalni optimum.

Potom je vršeno poređenje za zadovoljive testove, gde su rezultati u tabeli 5. Iz rezultata vidi se slična situacija kao i u prethodnoj grupi, samo ovoga puta SAWEA i RFEA daleko odstupaju od preostala dva unapređenja lokalnom pretragom. RFEA se po rezultatima čini nešto bolji, ali kada je vreme izvršavanja u pitanju, to apsolutno nije opravdano. Slično je i pri poređenju FlipGA i ASAP gde su oba postigli iste rezultate u svim instancama, pri čemu je ASAP neuporedivo brži. Jedini algoritam koji je rešio čak dve instance više od ostalih je MASAP, sa prosečnim vremenom izvršavanja čak kraćim nego ASAP, pa se stoga, značajno izdvaja u odnosu na ostale.

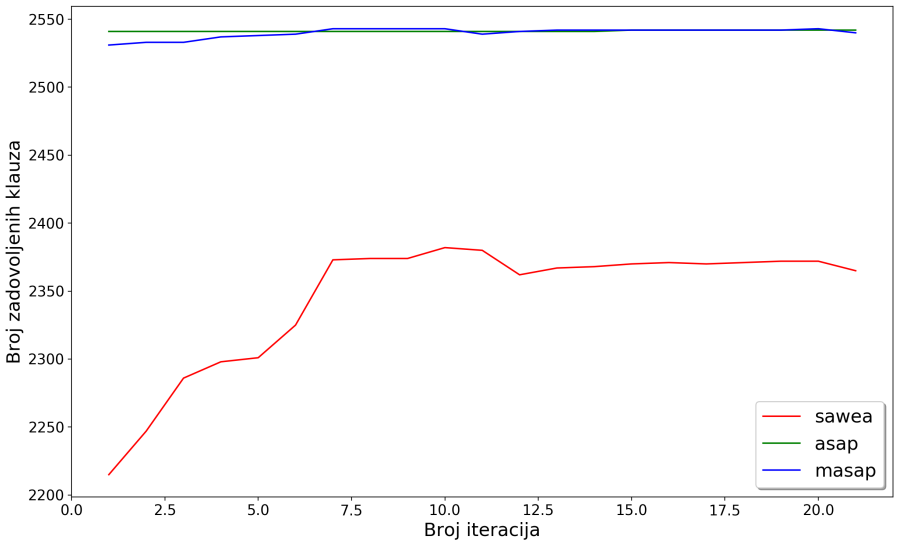
Tabela 5: AIM zadovoljivi testovi

Instanca	Broj literala	Broj klauza	SAWEA	RFEA	FlipGA	ASAP	MASAP
aim-50-1_6-yes	50	80	79	79	79	79	79.5
aim-50-2_0-yes		100	98.4	99	99	99	100
aim-50-3_4-yes		170	167	167.2	170	170	170
aim-50-6_0-yes		300	300	290	300	300	300
aim-100-1_6-yes	100	160	158.2	158	159	159	159
aim-100-2_0-yes		200	195.8	197.8	199	199	199
aim-100-3_4-yes		340	324.6	332.4	340	340	340
aim-100-6_0-yes		600	586.8	576	**600**	600	**600**
aim-200-2_0-yes	200	400	392.2	396	399	399	399
aim-200-6_0-yes		1200	1133	1173	1200	1200	1200



Slika 3: EA aim-100-3_4-yes

Jedna instanca zadovoljivih problema prikazana je na slici 3. U skladu sa uočenom monotonošću RFEA (93,06s) koja se dobija zahvaljujući komplikovanijem izračunavanju funkcije prilagođenosti f_{REF} uspela je da ga istakne pre SAWEA (0,80s), ali po brzini izvršavanja i dalje ostaje najgore rangiran. Problem rešavaju FlipGA (140,47s) prvi dolazi do rešenja, ali vremenski su bolji i dalje ASAP (27,42s) i MASAP (21,54s).



Slika 4: EA f600 za 20 iteracija

Zaključujemo da su SAWEA sa jedne strane, a sa druge ASAP i MASAP iako različiti po efikasnosti, najbrži algoritmi, pa ih za razliku od ostalih možemo pokrenuti i na nekim još težim instancama, kao što je formula f600 3-SAT koja se sastoji od 600 promenljivih i 2550 klauza. U pitanju je zadovoljiva formula, pa je očekivano rešenje baš 2550. Na slici 4 jasno se vidi velika razlika u efikasnosti između SAWEA (0,43s) i ASAP (230,40s)/MASAP (115,30s). ASAP i MASAP su u samo 20 iteracija skoro došli do tačnog rešenja, dok je SAWEA dosta dalji, polako konvergirao. Naravno, ovakva drastična razlika plaćena je vremenom izvršavanja, pa je SAWEA znatno brži. Na težoj instanci se ovoga puta videla i vremenska prednost kao i veća tačnost MASAP-a u odnosu na njegovu nemodifikovanu verziju.

3 Optimizacija rojem čestica (PSO)

Optimizacija rojem čestica (Particle swarm optimization – PSO) [12] je jedna od tehnika pretraživanja zasnovana na populaciji, kao što su i evolutivni algoritmi, ali ne koristi evolutivne tehnike mutacije i ukrštanja. PSO algoritam su 1995. godine uveli Kenedi i Eberhart kao alternativu standardnim genetskim algoritmima.

Optimizacija rojem čestica je algoritam koji se zasniva na ponašanju pojedinih jedinki unutar neke grupe, na primer, jata ptica ili roja insekata. Ako neka jedinka unutar jata pronađe izvor hrane čitavo jato će krenuti njenim putem. Naravno, svaka jedinka može, vođena sopstvenim instinktom napustiti jato, u potrazi za hranom. U slučaju da nađe hranu, čitavo jato će je slediti.

PSO pripada grupi algoritama koji se zasnivaju na inteligenciji roja (eng. *swarm intelligence*). Algoritam radi nad skupom jedinki koji se naziva rojem. Elementi ovog skupa, odnosno jedinke, se nazivaju česticama. Svaka čestica predstavlja jedno potencijalno rešenje problema koji se rešava. Čestice se prema zadatim pravilima kreću po prostoru pretraživanja. Kretanje čestica se usmerava u zavisnosti od njihove trenutne pozicije, njihove do sada najbolje pozicije, kao i do sada najbolje pozicije u čitavom roju. Ažuriranje pozicija se nastavlja dok ne bude zadovoljen neki od kriterijuma zaustavljanja. Takođe se, u svakoj iteraciji, ažurira i najbolje rešenje čitavog roja

Neka je dat roj sa $\vec{S}$ čestica. Svaka čestica se sastoji od tri elementa:

- Pozicija u prostoru za pretragu $\vec{x_i}$
- Brzina $\vec{v_i}$ kojom se čestica pomera na sledeću poziciju
- Sećanje, koje se koristi za skladištenje elitnih čestica globalne pretrage $\vec{P_g}$, kao i najboljih individualnih rešenja $\vec{P_i}$ koja su do sada pronašle zasebne čestice

Nije neophodno da se u budućim populacijama nalazi bilo koji elitni pojedinac, iako svaka čestica u populaciji pokušava da bude blizu svog najboljeg rešenja i globalnog najboljeg rešenja.

Osnovni oblik PSO algoritma dat je sledećim samoažurirajućim jednačinama:

$$\vec{v_i}^{t+1} = w \cdot \vec{v_i}^{t} + c_1 \cdot \vec{r_1} \times (\vec{P_i}^{t} - \vec{x_i}^{t}) + c_2 \cdot \vec{r_2} \times (\vec{P_g}^{t} - \vec{x_i}^{t}) \tag{1}$$

$$\vec{x_i}^{t+1} = \vec{x_i}^{t} + \vec{v_i}^{t+1} \tag{2}$$

Jednačina 1 opisuje kako se ažurira brzina i-te čestice, a 2 koja je sledeća pozicija i-te čestice, pri čemu je:

- w - faktor inercije
- c_1, c_2 - faktori učenja: kognitivna i socijalna
- $\vec{v_{id}}^{t}$ - brzina i-te čestice u iteraciji t
- $\vec{x_{id}}^{t}$ - pozicija i-te čestice u iteraciji t
- $\vec{r_1}, \vec{r_2}$ - pseudoslučajni brojevi iz uniformnog intervala $[0, 1]$
- $\vec{P_i}$ - najbolje individualno rešenje čestice i
- $\vec{P_g}$ - trenutno najbolje globalno rešenje

Kako je max k-SAT problem diskretan potrebno je prilagoditi jednačinu 2. Izračunata brzina $\vec{v_i}$ je iz $\mathbb{R}^n$, pa je potrebno da se svede na $\{0, 1\}^n$. Jedan predlog za ažuriranje položaja čestice, izložen u radu [8], dat je sigmoidnom transformacijom. Sada ${v_i}^{t}$ predstavlja verovatnoću da bit x_i^t uzme vrednost 1.

$$x_i^t = \begin{cases} 1, rand(0,1) < sigmoid(v_i^t) \\ 0, inace \end{cases} \tag{3}$$

$$sigmoid(v_i^t) = \frac{1}{1 + e^{-v_i^t}} \tag{4}$$

3.1 Pseudokod PSO

U ovom poglavlju dat je osnovni oblik algoritma na kojem se zasnivaju ostale varijante i izmene koje će biti detaljnije izložene. Jednu populaciju, odnosno roj, čini unapred određen broj čestica, lista potencijalnih rešenja kao i dodeljene brzine za svaku od njih. Kroz iteracije računa se fitnes, ažuriraju se brzine i pozicije čestica dok se ne zadovolji kriterijum zaustavljanja. Dve varijante (PSOSAT i WPSOSAT) uvode i lokalnu pretragu koja se izvodi umesto ažuriranja pozicija, odnosno jednačine 3. Kriterijumi zaustavljanja koji se mogu koristiti u opštem slučaju su: da li je dostignut maksimalan broj unapred zadatih iteracija ili, da li u poslednjih nekoliko iteracija nema značajnog napretka. U test primerima za koje unapred znamo da je formula zadovoljiva, ili koliko je klauza zadovoljivo, možemo koristiti kriterijum da se dostigao ukupan broj zadovoljivih klauza.

Input : Formula F u KNF-u, n i m
Output: Najbolja procenjena valuacija i broj zadovoljenih klauza

inicijalizacija populacije: pozicije i brzine;
t = 0; `// tekuća iteracija`
while *nije zadovoljen uslov zaustavljanja* **do**
    t = t + 1;
    for $i \leftarrow 0$ **to** *broj cestica u roju* **do**
        Izračunaj $fitness(\vec{P_i^t})$;
        Sačuvaj individualni najbolji rezultat kao globalni $\vec{P_g}$;
        Ažuriraj brzine na osnovu $\vec{P_i}$ i $\vec{P_g}$;
        Ažuriraj pozicije $\vec{v_i^t}$;
        Ažuriraj individualni najbolji rezultat $\vec{P_i}$;
        Ažuriraj globalni najbolji rezultat $\vec{P_g}$;
    end
end

Algoritam 4: Osnovni PSO algoritam

Prvo je potrebno inicijalizovati pozicije čestica i vektora brzine. Pozicije su inicijalizovane pseudo-slučajnim brojevima $\{0, 1\}$, a vektor brzine realnim brojevima iz intervala $[-V_{min}, V_{max}]$, gde su granice intervala jedan od parametara PSO algoritma.

Koriste se iste fitnes funkcije kao i kod evolutivnih algoritama, opisane u delu 2.2. Biće korišćene f_{MAXSAT} i f_{SAW} funkcije prilagođenosti.

3.2 Varijante PSO algoritma

Da bismo uporedili kombinaciju lokalne pretrage, SAW fitnes funkcije i klasičnog PSO algoritma implementirani su i isprobane sledeće tri verzije: PSO-LS, PSOSAT i WPSOSAT [9].

3.2.1 PSO-LS

PSO-LS je osnovna varijanta diskretnog PSO algoritma koji ne koristi lokalnu pretragu, već sigmoidnu transformaciju, jednačine 1 i 3 za ažuriranje brzina i pozicije

čestica. Korišćena fitnes funkcija je f_{SAW}, sa upotrebom težina nad klauzama. Karakteristike ovog algoritma su: sporija konvergencija do globalnog optimuma, ali pojedinačne iteracije se izvršavaju brže.

3.2.2 PSOSAT

PSOSAT koristi lokalnu pretragu, ali ne i f_{SAW} funkciju prilagođenosti, pa je njegova funkcija broj zadovoljenih klauza. Mana ovog algoritma je teško izlaženje iz lokalnih optimuma zbog korišćenja fitnes funkcije koja ne razaznaje težinu klauza.

3.2.3 WPSOSAT

WPSOSAT - Modifikovan PSO algoritam sa korišćenjem flip heuristike i f_{SAW} fitnes funkcije sa težinama klauza. Značaj lokalne pretrage ogleda se u poređenju sa PSO-LS, a korišćenje F_{SAW} u poređenju sa PSOSAT, koji će biti prikazani u rezultatima 3.3.

3.3 Rezultati

Svi algoritmi pokretani su pet puta sa istim parametrima radi mogućnosti njihovog poređenja (parametri predloženi u radu [9]) datim u tabeli 6. Beležen je prosečan broj zadovoljenih klauza u maksimalno 1000 iteracija, pri čemu podebljan rezultat označava da se do rešenja dolazilo u prvoj iteraciji.

Tabela 6: Parametri PSO algoritma

Parametar	Vrednost
Broj iteracija	1000
w	1
c1	1.7
c2	2.1
Broj čestica	20
max flip	30000
v_{min}	-1
v_{max}	1

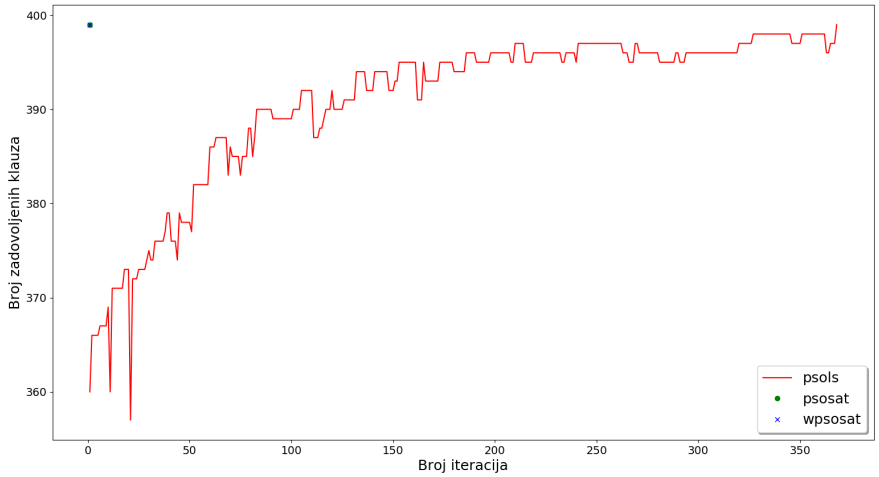
U tabeli 7 pokretane su instance nezadovoljivih testova. Ovde su, skoro svi algoritmi uspeli da brzo nađu rešenje. Za sada između PSOSAT i WPSOSAT nema značajne razlike, ali, već za poslednji test vidi se slabost ne korišćenja lokalne pretrage.

Tabela 7: AIM nezadovoljivi testovi

Instanca	Broj literala	Broj klauza	PSO-LS	PSOSAT	WPSOSAT
aim-50-1_6-no	50	80	79	**79**	**79**
aim-50-2_0-no	50	100	99	**99**	**99**
aim-100-1_6-no	100	160	159	**159**	**159**
aim-100-2_0-no	100	200	199	**199**	**199**
aim-200-2_0-no	200	400	398.6	**399**	**399**

Na slici 5 vidi se rešavanje jedne instance nezadovoljive formule. PSO-LS (6,60s), koji se zasniva na ažuriranju brzina i pozicija, dostiže globalni optimum uz nešto lokalnog lutanja što je posledica kretanja u hiperprostoru rešenja. S vrlo malom vremenskom razlikom, samo zahvaljujući korišćenju lokalne pretrage, PSOSAT (6,73s) dolazi

do rešenja u jednoj iteraciji, kao i WPSOSAT (8,20s).



Slika 5: PSO aim-200-2_0-no

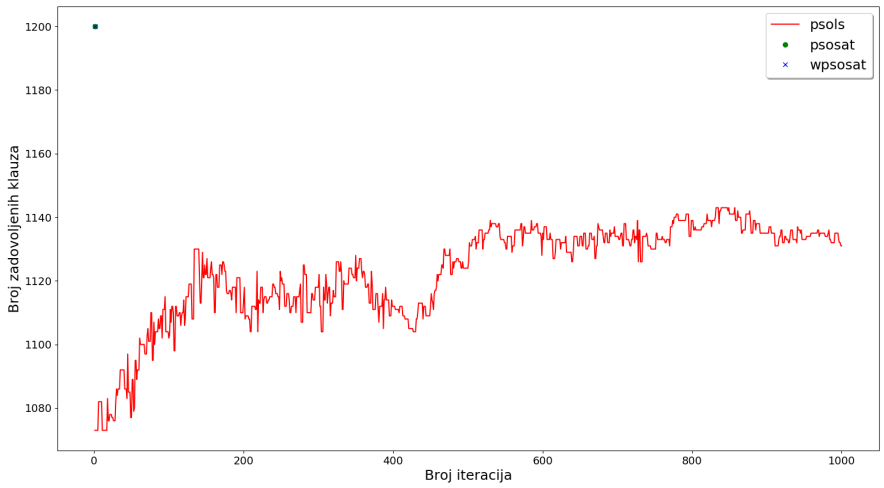
Potom su pokrenuti zadovoljivi testovi i njivoh rezultat beležen je u tabelu 8. Sada PSO-LS nije mogao da se uporedi sa ostala dva algoritma. Za neke instance do globalnog optimuma došao je jedino WPSOSAT odakle se vidi značaj SAW funkcije, i naravno, lokalne pretrage.

Tabela 8: AIM zadovoljivi testovi

Instanca	Broj literala	Broj klauza	PSO-LS	PSOSAT	WPSOSAT
aim-50-1_6-yes	50	80	79	79.2	80
aim-50-2_0-yes		100	99	99	100
aim-50-3_4-yes		170	168.6	170	170
aim-50-6_0-yes		300	300	300	300
aim-100-1_6-yes	100	160	158.6	159	159
aim-100-2_0-yes		200	198	199	200
aim-100-3_4-yes		340	328	339.4	340
aim-100-6_0-yes		600	580.2	**600**	**600**
aim-200-2_0-yes	200	400	395.4	399	399
aim-200-6_0-yes		1200	1135.6	**1200**	**1200**

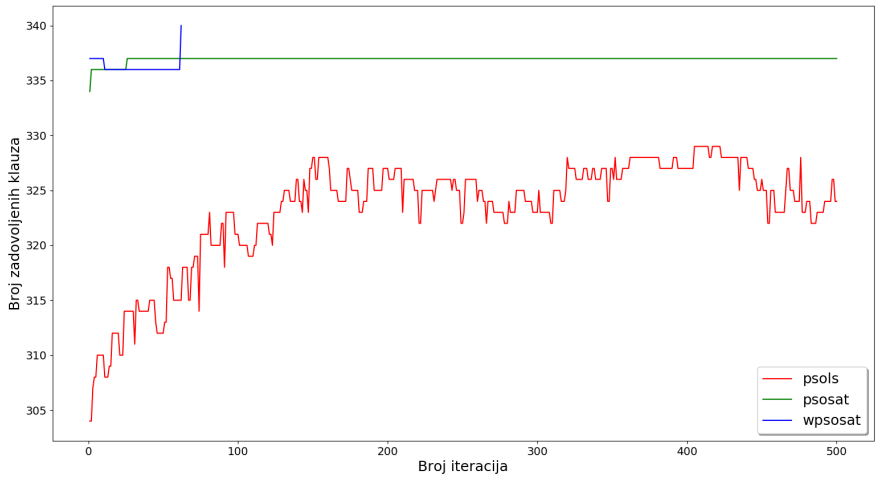
Na slici 6 vidi se da sa složenošću instance koja se rešava, običan PSO-LS (44,44s) koji nije uspeo da dođe do rešenja, ne može da se poredi sa PSOSAT (34,60s) i WPSOSAT (40,80s) koji su opet došli do rešenja u prvoj iteraciji. Gledajući zbirno ove tri vezije, vremena izvršavanja ne razlikuju se toliko drastično, kako je bio slučaj kod EA, gde je jedan jednostavniji algoritam bio znatno brži od ostalih.

Sledeća instanca prikazana na slici 7 ograničena je na 500 iteracija, kako posle nije bilo značajnog napretka. PSO-LS (6,18s) ponaša se očekivano s prethodnim zaključcima, ali se sada vidi razlika između preostala dva algoritma. PSOSAT (303,84s) zbog loše fitnes funckije i usled nedostatka ažuriranja težina, iako blizu rešenja upada



Slika 6: PSO aim-200-6_0-yes

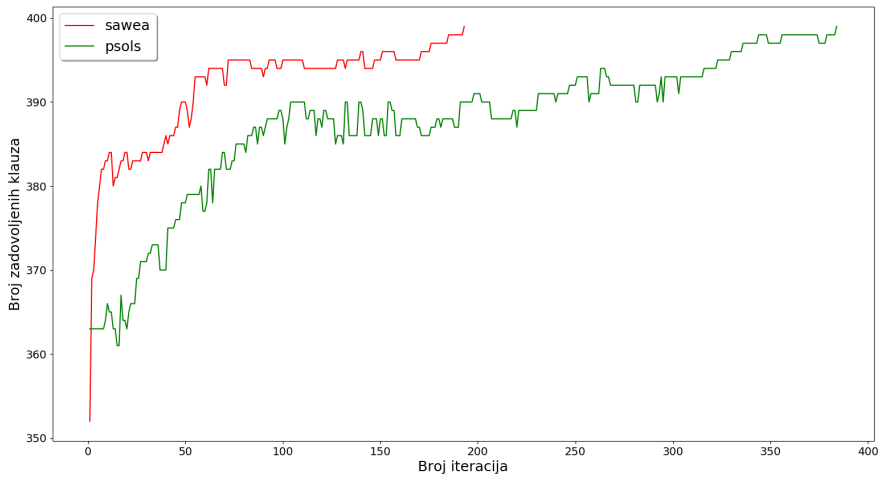
u jedan lokalni optimum iz koga ne može da se izvuče. Za razliku od njega WPSOSAT (51,65s) u svega nekoliko iteracija jedini dolazi do rešenja. U sledećem poglavlju biće prikazan odnos u efikasnosti prethodno dve izložene grupe algoritama EA i PSO, po odgovarajućim kategorijama.



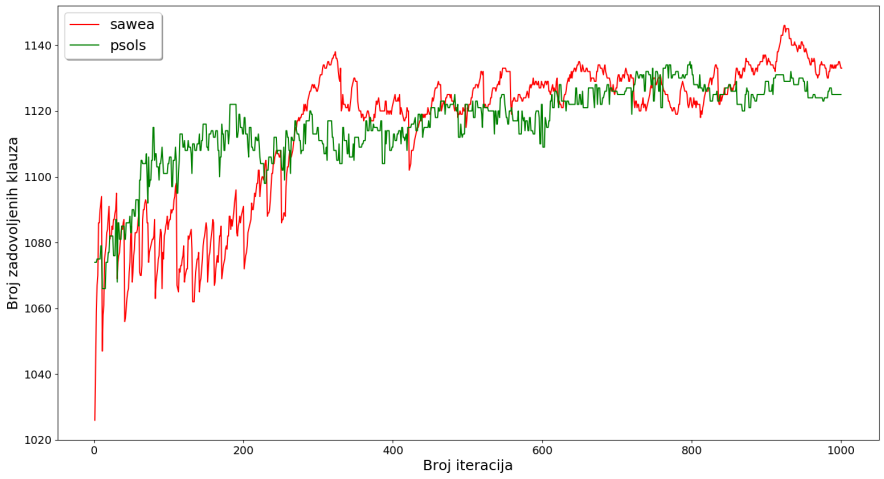
Slika 7: PSO aim-100-3_4-yes

4 EA nasuprot PSO

Do sada su bili razmatrani zasebno evolutivni algoritmi u poglavlju 2, za koje su rezultati bili prikazani tabelarno i grafički u 2.9, kao i optimizacioni algoritmi rojem čestica, u delu 3 sa pratećim rezultatima u 3.3. Kada su obrađeni zasebno i rangirani pojedinačni algoritmi iz svake grupe, sada se mogu objediniti slični iz EA i iz PSO i međusobno uporediti.



Slika 8: Laki EA vs PSO aim-200-2_0-no

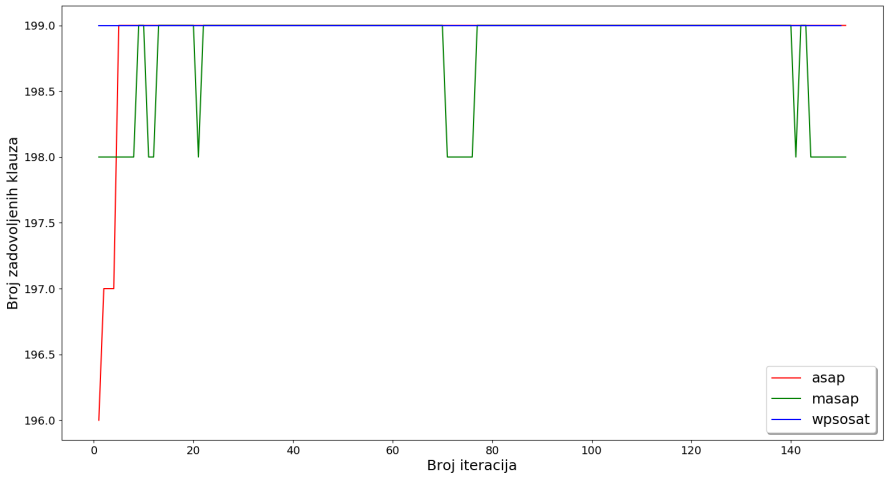


Slika 9: Laki EA vs PSO aim-200-6_0-yes

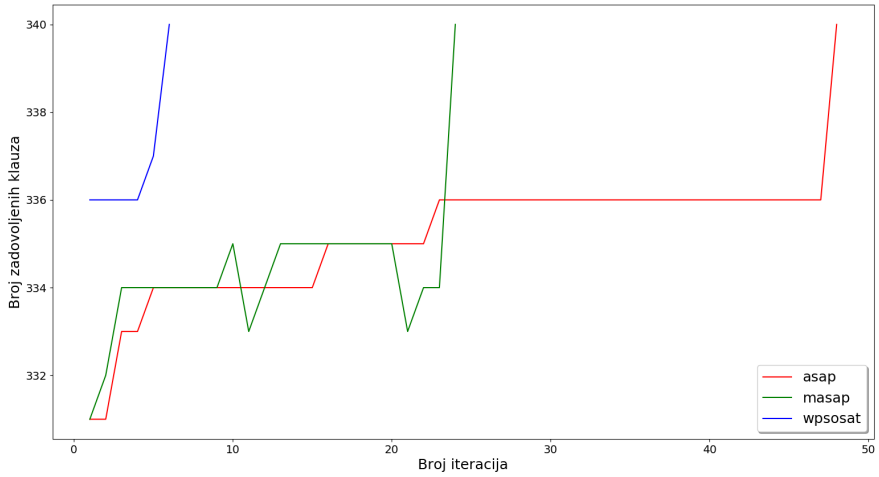
Prvo se mogu uporediti algoritmi ,,lake kategorije" koji su se pokazali kao vremenski brzi sa sporom konvergencijom: SAWEA i PSO-LS. Za jednu nezadovoljivu formulu 8, kao i za zadovoljivu 9 vidi se slično ponašanje. SAWEA (0,60 i 9,06s) pokazao se kao

brži algoritam koji pritom u manje iteracija dolazi do rešenja za razliku od PSO-LS (6,89 i 46,97s).

Može se zaključiti da su oba algoritma dosta slična, ali se, na ovim testovima, predstavnik evolutnivnih algoritama SAWEA pokazao kao brži, gledajući i vremenski, i po broju iteracija, u odnosu na prestavnika optimizacije rojem čestica PSO-LS.



Slika 10: Teži EA vs PSO aim-100-2_0-yes



Slika 11: Teži EA vs PSO aim-100-3_4-yes

Dalje su poređeni najbolji algoritmi iz kategorije efikasnih po tačnosti rešenja, ali nešto sporijih zbog komplikovanije implikacije - ASAP i MASAP protiv WPSOSAT. Na slici 9 vidi se da i u ovoj kategoriji evolucioni algoritmi prednjače u odnosu na

optimizacije rojem čestica, pa je i ovde najbolji MASAP (4,50s) u odnosu na svog prethodnika ASAP (11,38s) i WPSOSAT (71,36s). Kako ni jedan od njih nije došao do tačnog rešenja ograničeno je na 150 iteracija, pa su data vremena realan prikaz trajanja jedne iteracije.

Za instancu aim-200-2_0-no skoro svi algoritmi dolazili su do rešenja u prvoj iteraciji. Po vremenu izvršavanja MASAP (0,75s) i ASAP (0,92s) su se opet pokazali kao brži u odnosu na WPSOSAT (8,27s).

Još jedan primer ove kategorije prikazan je na slici 11 gde su opet svi algoritmi došli do rešenja, i gde se MASAP (2,48s) rangirao kao najbolji, za kojim je ASAP (7,13s) i WPSOSAT (8,50) koji je u najmanje iteracija došao do rešenja. Već je utvrđeno da njegove pojedinačne iteracije traju duže u odnosu na preostala dva algoritma.

5 Zaključak

U ovom radu objedinjeni su i poređeni razni neegzaktni algoritmi za rešavanje optimizacionog problema maksimalne k-zadovoljivosti, a koji su tesno povezani sa evolucionim algoritmima. Ispričan je nastanak evolucionih algoritama, njihove osnovne karakteristike i nekoliko unapređenja. Dat je osvrt i na algoritam optimizacije rojem čestica, koji je po prirodi blizak sa EA, ali suštinski nije podoban za rešavanje diskretnih problema kakav je max k-SAT, što se videlo i u rezultatima. U zavisnosti od potrebe, razni algoritmi su se pokazali kao podobno rešenje, pa su s toga podeljene u dve kategorije. Ako je potrebno brzo odrediti broj zadovoljenih klauza, pri čemu nije od velikog značaja tačno rešenje, već je dovoljno i približno, a posebno ako su veliki problemi u pitanju, zbog brzine koristiće se SAWEA ili PSO-LS. Ako je ipak važna brža konvergencija, najbolje se pokazao MASAP, novi algoritam predstavljen u ovom radu, a potom i njemu blizak ASAP. Oni su zahvaljujući svojim unapređenjima nadmašili FlipGA, kao i sve varijante PSO algoritama.

Literatura

[1] J. Kok C. Rossi, E. Marchiori. An adaptive evolutionary algorithm for the satisfiability problem. *Proceedings of the 2000 ACM Symposium on Applied Computing*, 2000.

[2] C. Rossi E. Marchiori. A flipping genetic algorithm for hard 3-sat problems. 1999. In Proceedings of the Genetic and Evolutionary Computation Conference.

[3] B. Stein H. K. Buning. A study of evolutionary algorithms for the satisfiability problem, 2004.

[4] H. Shen H. Zhang. Exact algorithms for max-sat. *Electronic Notes in Theoretical Computer Science Vol. 86*, 2003.

[5] C. Rossi J. Gottlieb, E. Marchiori. Evolutionary algorithms for the satisfiability problem.

[6] N. Voss J. Gottlieb. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. *Parallel Problem Solving from Nature*, 1998.

[7] N. Voss J. Gottlieb. Adaptive fitness functions for the satisfiability problem. 2000. In Parallel Problem Solving from Nature.

[8] R.C. Eberhart J. Kennedy. A discrete binary version of the particle swarm algorithm, 2007.

[9] Abdesslem Layeb. A particle swarm algorithm for solving the maximum satisfiability problem.

[10] W. Kosters M. de Jong. Solving 3–sat using adaptive sampling. 1998. In Proceedings of the Tenth Dutch/Belgian Artificial Intelligence Conference.

[11] Mladen Nikolić Predrag Janičič. *Veštačka inteligencija*. Beograd, 2019.

[12] J. Kennedy R. Eberhart. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 2006.