Heurističko rešavanje problema minimalnog broja zadovoljivih formula

Aleksa Papić Aleksandar Stefanović

17. septembar 2022.

Sadržaj

1 Uvod

Problem minimalne zadovoljivosti iskazne formule f (eng. MIN-SAT) je optimizaciona varijanta problema zadovoljivosti (eng. SAT) u kojoj se traži valuacija v takva da je broj klauza formule f tačnih u valuaciji v minimalan. Poznato je da je ovaj problem NP-težak [1].

U ovom radu ćemo posmatrati naredno modifikaciju ovog problema datu u [2]:

Definicija 1.1 (Problem minimalnog broja zadovoljivih formula)**.** Neka je dat par (U, C) gde je U skup iskaznih promenljivih, a C skup iskaznih formula u 3KNF (eng. 3CNF). Rešenje problema minimalnog broja zadovoljivih formula nad (U, C) je valuacija v za promenljive iz skupa U takva da je broj formula iz skupa C zadovoljenih tom valuacijom minimalan.

Iz definicije se može zaključiti da svaka instanca MIN-SAT problema odgovara nekoj instanci problema 1.1 u kojoj je broj klauza svake formule iz C jednak jedan.

Razmotrićemo i uporediti performanse jednog genetskog algoritma i više varijanti memetičkih algoritama za rešavanje problema 1.1.

2 Opis algoritama

U ovom poglavlju ćemo dati opis nekoliko pristupa u rešavanju problema 1.1.

2.1 Kodiranje jedinki

Pre razmatranja konkretnih algoritama, opisaćemo način kodiranja jedinki, tj. način predstavljanja konkretnih valuacija u okviru problema 1.1.

Definicija 2.1 (Kodiranje jedinki)**.** Neka je dat par (U, C) kao u 1.1 i neka je skup promenljivih $U = \{p_1, ..., p_n\}$. Tada je valuacija v nad skupom promenljivih U niz binarnih brojeva $(x_1, ..., x_n) \in \{0, 1\}^n$ takav da x_i odgovara konkretizovanoj vrednosti promenljive p_i.

2.2 Ocena kvaliteta jedinki

Ocenu kvaliteta jedinki u okviru problema 1.1 ćemo zadati preko tzv. fitnes funkcije jedinke.

Definicija 2.2 (Fitnes funkcija)**.** Neka je v neka jedinka, tj. konkretna valuacija (2.1) za par (U, C) definisan kao u problemu 1.1. Fitnes funkcija

$fitness : \{0,1\}^n \to (0,1]$ je zadata sa $fitness(v) = \frac{1}{sat(v)+1}$, gde je $sat(v)$ broj iskaznih formula iz C zadovoljenih u valuaciji v.

Iz date definicije se može zaključiti da je jedinka v_1 bolja od jedinke v_2 u kontekstu problema 1.1 ako i samo ako je $fitness(v_1) > fitness(v_2)$.

Broj zadovoljenih formula u valuaciji v se može dobiti kao $sat(v) = \frac{1}{fitness(v)} - 1$, ali se zbog računa u pokretnom zarezu predlaže zaokruživanje na najbliži ceo broj, tj. $sat(v) = round(\frac{1}{fitness(v)} - 1)$.

2.3 Rešavanje algoritmom grube sile

Naivni algoritam kojim se problem rešava grubom silom proverava sve moguće valuacije u problemu. Ukoliko je U skup promenljivih u problemu 1.1, takvih valuacija je $2^{|U|}$. Iako je egzaktan, ovaj algoritam je praktično neprimenljiv za sve osim najmanje probleme zbog svoje eksponencijalne složenosti.

2.4 Rešavanje genetskim algoritmom

Prvi heuristički algoritam koji ćemo razmotriti je genetski algoritam. Opisaćemo operatore koje ćemo koristiti, kao i uslov zaustavljanja. Detaljan opis algoritma i mogućih modifikacija se može videti u [3].

Algoritam 1: Genetski algoritam

```
t ← 0;
P_0 ← generisi populaciju();
while nije ispunjen uslov zaustavljanja do
    P_sel ← selekcija(P_t);
    P_{t+1} ← ukrstanje(P_sel);
    P_{t+1} ← mutacija(P_{t+1});
    t ← t + 1;
end
```

2.4.1 Operator selekcije

Za selekciju jedinki za reprodukciju koristićemo ruletsku selekciju. Verovatnoća izbora neke jedinke v jednaka je $\frac{fitness(v)}{\sum_{u \in P_t} fitness(u)}$, gde je P_t populacija jedinki u tekućoj iteraciji algoritma 1.

2.4.2 Operator ukrštanja

Za ukrštanje jedinki prilikom reprodukcije koristićemo jednopoziciono ukrštanje. Dve jedinke, $r_1 = (x_1, ..., x_n)$ i $r_2 = (y_1, ..., y_n)$, izabrane za roditelje u fazi selekcije kreiraće dva potomka, p_1 i p_2, izborom *tačke preseka* k iz diskretne uniformne raspodele nad vrednostima $\{1, ..., n\}$. Tada će važiti $p_1 = (x_1, ..., x_k, y_{k+1}, ..., y_n)$ i $p_2 = (y_1, ..., y_k, x_{k+1}, ..., x_n)$.

2.4.3 Operator mutacije

Operator mutacije koji ćemo koristiti sa nekom verovatnoćom $p \in U(0, 1)$, koja se zadaje kao parametar algoritma 1, invertuje jedan od bitova jedinke nad kojom se sprovodi mutacija.

2.4.4 Uslov zaustavljanja

Kao uslov zaustavljanja koristićemo maksimalni broj iteracija algoritma 1, kao i maksimalni broj iteracija bez promene u najboljoj jedinki. Obe vrednosti se zadaju kao parametri algoritma.

2.4.5 Elitizam

U algoritmu ćemo koristiti elitizam, tj. određeni broj najboljih jedinki u svakoj generaciji neće biti zamenjen novonastalom generacijom. Procenat elitizma se zadaje kao parametar algoritma.

2.5 Rešavanje memetskim algoritmom

Još jedan tip algoritama koje ćemo razmotriti su memetski algoritmi. Ovi algoritmi predstavljaju kombinaciju više različitih heurističkih pristupa rešavanju problema [4].

Konkretna implementacija koju ćemo razmotriti kombinuje genetski algoritam opisan u prethodnom poglavlju sa nekom S-metaheuristikom, a mi ćemo ih obraditi tri. Sledi uopšteni algoritam:

Jedina razlika u odnosu na genetski algoritam 1 je novouvedeni operator optimizacije. Svi ostali operatori su implementirani identično kao u prethodnom poglavlju.

2.5.1 Operator optimizacije

Uloga operatora optimizacije je da se potencijalno poboljšaju pojedinačne jedinke iz novonastale populacije. Ovo se suštinski postiže pozivanjem

Algoritam 2: Memetski algoritam

$t \leftarrow 0$;
$P_0 \leftarrow$ *generisi populaciju*();
while *nije ispunjen uslov zaustavljanja* **do**
  $P_{sel} \leftarrow selekcija(P_t)$;
  $P_{t+1} \leftarrow ukrstanje(P_{sel})$;
  $P_{t+1} \leftarrow mutacija(P_{t+1})$;
  $P_{t+1} \leftarrow optimizacija(P_{t+1})$;
  $t \leftarrow t + 1$;
end

neke S-metaheuristike, koja je zadata kao parametar algoritma, nad svakom jedinkom ili nad određenim brojem njih. U ovom radu ćemo koristiti dodatan parametar frekvencije optimizacije k kojim se uspostavlja optimizacija jedinki iz svake $k - te$ reprodukcije.

Lokalna pretraga kao operator optimizacije Najjednostavnija S-metaheuristika koju ćemo koristiti je lokalna pretraga. Kriterijum zaustavljanja je broj iteracija koji se prosleđuje kao parametar algoritmu.

Algoritam 3: Lokalna pretraga

Data: *pocetna jedinka*
Result: *najbolja jedinka*
trenutna jedinka $\leftarrow$ *pocetna jedinka*;
while *nije ispunjen uslov zaustavljanja* **do**
  nova jedinka $\leftarrow$ *invertuj*(*trenutna jedinka*);
  if *fitness(nova jedinka)* $>$ *fitness(trenutna jedinka)* **then**
    trenutna jedinka $\leftarrow$ *nova jedinka*;
  end
end
najbolja jedinka $\leftarrow$ *trenutna jedinka*;

Simulirano kaljenje kao operator optimizacije Verovatnoća kaljenja se računa po formuli $\frac{1}{t^s}$, gde se broj s prosleđuje kao parametar algoritma. Kriterijum zaustavljanja je broj iteracija koji se takođe prosleđuje kao parametar.

Algoritam 4: Simulirano kaljenje

```
Data: pocetna jedinka
Result: najbolja jedinka
trenutna jedinka ← pocetna jedinka;
najbolja jedinka ← pocetna jedinka;
t ← 1;
while nije ispunjen uslov zaustavljanja do
    nova jedinka ← invertuj(trenutna jedinka);
    if fitness(nova jedinka) > fitness(trenutna jedinka) then
        trenutna jedinka ← nova jedinka;
        if fitness(nova jedinka) > fitness(najbolja jedinka) then
            najbolja jedinka ← nova jedinka;
        end
    else
        p ← 1/t^s;
        if q ∈ U(0,1) < p then
            trenutna jedinka ← nova jedinka;
        end
    end
    t ← t + 1;
end
```

Redukovana metoda promenljivih okolina kao operator optimizacije Okolina veličine k neke jedinke v predstavlja skup jedinki koje se mogu dobiti invertovanjem tačno k bitova u reprezentaciji jedinke v. Maksimalna veličina okoline se prosleđuje kao parametar algoritma, kao i broj iteracija algoritma koji predstavlja kriterijum zaustavljanja.

Algoritam 5: Redukovana metoda promenljivih okolina

```
Data: pocetna jedinka
Result: najbolja jedinka
trenutna jedinka ← pocetna jedinka;
while nije ispunjen uslov zaustavljanja do
    for k ← 1 to maks. okolina do
        nova jedinka ← invertuj(trenutna jedinka, k);
        if fitness(nova jedinka) > fitness(trenutna jedinka) then
            trenutna jedinka ← nova jedinka;
            break;
        end
    end
end
najbolja jedinka ← trenutna jedinka;
```

3 Rezultati

Svi testovi su izvršeni implementacijom algoritama u programskom jeziku Python na Intel Xeon procesoru sa radnim taktom 2.20GHz. Rešavanje algoritmom grube sile se vrši samo nad skupom malih instanci.

3.1 Male instance

Testovi iz skupa malih instanci su izvršeni 100 puta sa istim parametrima. Maksimalan broj generacija svih heurističkih algoritama je 100, maksimalan broj generacija bez poboljšanja najbolje jedinke je 30 i veličina populacije je 20. Za elitizam je uzeto 10% populacije, verovatnoća mutacije je 0.05, broj iteracija optimizacije memetskih algoritama je 10, frekvencija 3, a maksimalna veličina okoline u redukovanoj metodi promenljivih okolina 3.

Instanca	small1-50-7-1	small2-50-7-2	small3-50-10-1	small4-50-10-3	small5-50-15-1	small6-50-15-4
|U|	7	7	10	10	15	15
|C|	50	50	50	50	50	50
BF opt.	40	29	37	23	34	17
GA najbolje	40	29	37	23	34	17
GA prosek	40.0	29.0	37.17	23.1	34.38	18.37
GA najgore	40	29	39	27	36	21
MA(LS) najbolje	40	29	37	23	34	17
MA(LS) prosek	40	29.0	37.03	23.0	34.43	18.19
MA(LS) najgore	40	29	38	23	36	20
MA(SA) najbolje	40	29	37	23	34	17
MA(SA) prosek	40.0	29.0	37.01	23.0	34.56	18.17
MA(SA) najgore	40	29	38	23	36	20
MA(RVNS) najbolje	40	29	37	23	34	17
MA(RVNS) prosek	40.0	29.0	37.0	23.0	34.34	17.83
MA(RVNS) najgore	40	29	37	23	35	19
t BF	< 0.01	0.01	0.04	0.07	1.18	2.70
t GA	0.03	0.06	0.05	0.08	0.05	0.11
t MA(LS)	0.11	0.16	0.13	0.24	0.15	0.34
t MA(SA)	0.11	0.16	0.14	0.24	0.16	0.34
t MA(RVNS)	0.24	0.35	0.26	0.46	0.32	0.72

Tabela 1: Rezultati nad malim instancama

3.2 Srednje instance

Testovi iz skupa srednjih instanci su izvršeni 20 puta sa istim parametrima. Maksimalan broj generacija svih heurističkih algoritama je 500, maksimalan broj generacija bez poboljšanja najbolje jedinke je 50 i veličina populacije je 70. Za elitizam je uzeto 20% populacije, verovatnoća mutacije je 0.05, broj iteracija optimizacije memetskih algoritama je 15, frekvencija 3, a maksimalna veličina okoline u redukovanoj metodi promenljivih okolina 3.

Instanca	medium1-100-30-5	medium2-150-50-5	medium3-150-75-5
|U|	30	50	75
|C|	100	150	150
BF opt.	-	-	-
GA najbolje	23	34	21
GA prosek	26.1	38.8	30.9
GA najgore	31	43	37
MA(LS) najbolje	23	37	29
MA(LS) prosek	26.45	43.05	36.2
MA(LS) najgore	32	51	49
MA(SA) najbolje	23	36	28
MA(SA) prosek	26.4	41.85	36.55
MA(SA) najgore	30	45	46
MA(RVNS) najbolje	23	40	27
MA(RVNS) prosek	27.7	45.05	38.95
MA(RVNS) najgore	31	52	49
t BF	-	-	-
t GA	1.47	2.66	3.50
t MA(LS)	7.75	15.26	18.95
t MA(SA)	8.08	16.99	21.19
t MA(RVNS)	17.49	35.20	44.72

Tabela 2: Rezultati nad srednjim instancama

3.3 Velike instance

Testovi iz skupa velikih instanci su izvršeni 10 puta sa istim parametrima. Maksimalan broj generacija svih heurističkih algoritama u slučaju instance large1-500-200-5 je 200, a u slučaju instance large2-300-300-5 je 350, maksimalan broj generacija bez poboljšanja najbolje jedinke je 70 i veličina populacije je 100. Za elitizam je uzeto 30% populacije, verovatnoća mutacije je 0.05, broj iteracija optimizacije memetskih algoritama je 15, frekvencija 3, a maksimalna veličina okoline u redukovanoj metodi promenljivih okolina 3.

Instanca	large1-500-200-5	large2-300-300-5
|U|	200	300
|C|	500	300
BF opt.	-	-
GA najbolje	126	34
GA prosek	132.1	44.2
GA najgore	139	52
MA(LS) najbolje	145	46
MA(LS) prosek	154.8	51.3
MA(LS) najgore	161	61
MA(SA) najbolje	142	39
MA(SA) prosek	150.4	51.4
MA(SA) najgore	167	58
MA(RVNS) najbolje	150	51
MA(RVNS) prosek	167.8	59.4
MA(RVNS) najgore	178	66
t BF	-	-
t GA	19.59	18.79
t MA(LS)	86.62	89.68
t MA(SA)	87.14	89.86
t MA(RVNS)	206.92	227.04

Tabela 3: Rezultati nad velikim instancama

4 Zaključak

Iako je ideja bila obećavajuća, u ovom slučaju su se memetski algoritmi pokazali loše u odnosu na klasičan genetski algoritam. U slučaju malih instanci, svi heuristički algoritmi su uspevali da nađu optimalno rešenje u skoro svakom pokretanju. Za srednje i velike instance, genetski algoritam je uvek nadmašivao sve tri varijante memetskih algoritama, kako u nađenom rešenju, tako i u vremenu izvršavanja koje je višestruko manje.

Što se tiče međusobnog poređenja tri varijante memetskih algoritama, najgore se pokazuje algoritam sa optimizacijom baziranom na redukovanoj

metodi promenljivih okolina, dok su varijante bazirane na simuliranom kaljenju i na lokalnoj pretrazi dale slične rezultate.

Potencijalan razlog za loše ponašanje memetskih algoritama u ovakvoj implementaciji je prebrza konvergencija ka nekom lokalnom optimumu usled dodatne optimizacije jedinki. Moguće rešenje za ovaj problem je upotreba naprednijih memetskih algoritama, poput onog opisanog u [5].

Reference

[1] Rajeev Kohlit, Ramesh Krishnamurti, Prakash Mirchandani, *The minimum satisfiability problem.* SIAM J. Discrete Math. Vol. 7, No. 2, pp. 275-283, May 1994.

[2] Viggo Kann, *Polynomially bounded minimization problems that are hard to approximate.* Nordic Journal of Computing 1(1994), 317–331.

[3] Engelbrecht, Andries P. *Computational intelligence : an introduction / Andries P. Engelbrecht. – 2nd ed.*

[4] Pablo Moscato, Carlos Cotta, Alexandre Mendes, *Memetic Algorithms.*

[5] Muzaffar Eusuff, Kevin Lansey, Fayzul Pasha, *Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization.* Engineering Optimization, volume 38, pages 129-154.