Rešavanje problema maksimalne k-zadovoljivosti optimizacijom rojem čestica

David Dimić
Zorana Gajić

Mart 2019.

Sadržaj

1 Uvod

2 Formulacija problema

U ovom radu, n će predstavljati broj promenjivih, a m broj klauza u formuli. 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 promenjiva ili njegova negacija i može se pojavljivati više puta u izrazu. Cilj je pronaći valuaciju, odnosno vektor $v = (x_1, x_2, ..., x_n) \in \{0, 1\}^n$ koja maksimizuje broj zadovoljenih klauza u formuli F.

3 Optimizacija rojem čestica (PSO)

Optimizacija rojem čestica (Particle swarm optimization – PSO) je jedna od tehnika pretraživanja zasnovana na populaciji kao što je genetski algoritam, ali ne koriste evolutivne algoritme kao što su mutacija i ukrštanje. PSO algoritmi su 1995. godine uveli Kenedi i Eberhart kao alternativu standardnim genetskim algoritmima.

Optimizacija rojem čestica je algoritam zasnovan na ponašanju pojedinačnih jedinki unutar određene grupe (na primer, jata ptica ili roja insekata). Ukoliko se, vođeno instiktom, jato prica uputi u određenom smeru u potrazi za hranom, očekivanje je da će čitavo jato slediti upravo onu pticu koja je pronašla izvor hrane. Međutim, i svaka ptica ponaosob može biti vođena sopstvenim instiktom i time na trenutak u potrazi za hranom napustiti jato. Tada se verovatno može desiti da, ukoliko pronađe bolji izvor hrane, čitavo jato upravo krene da sledi tu pticu.

PSO pripada skupu algoritama koji se zasnivaju na inteligenciji roja (swarm intelligence). Algoritam radi nad skupom jedinki, koji se naziva rojem. Elementi ovog skupa se nazivaju česticama. Svaka čestica predstavlja kandidatsko rešenje optimizacionog problema. Čestice se na unapred definisan način kreću po prostoru pretraživanja. Njihovo kretanje se usmerava imajući u vidu njihovu trenutnu poziciju, njihovu do sada najbolju poziciju, kao i do sada najbolju poziciju čitavog roja. Pod najboljom pozicijom čitavog roja se podrazumeva do sada najbolja pozicija, uzimajući u obzir sva njegova rešenja. Proces se ponavlja dok ne bude zadovoljen kriterijum zaustavljanja, a u svakoj iteraciji se ažurira najbolja vrednost rešenja za svaku česticu, kao i za roj u celini.

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

- Pozicija u prostoru za pretragu $\vec{x_i}$
- Brzina, vektor $\vec{v_i}$
- 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 [1], 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

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

```
inicijalizacija populacije: pozicije i brzine;
t = 0; // tekuca iteracija
while nije zadovoljen uslov zaustavljanja do
    t = t + 1;
    for i ← 0 to broj cestica u roju do
        Izračunaj Fitness(P_i^t);
        Primeni mutaciju;
        Sačuvaj individualni najbolji rezultat kao globalni P_g;
        Ažuriraj brzine na osnovu P_i i P_g;
        Ažuriraj pozicije v_i^t;
        Ažuriraj individualni najbolji rezultat P_i;
        Ažuriraj globalni najbolji rezultat P_g;
    end
end
```

Algorithm 1: Uopšteni PSO algoritam

3.2 Reprezentacija rešenja

Za dobar algoritam je takođe važno dobro predstavljanje rešenja. Postoje više načina reprezentacija, ali je ovde odabran prirodna binarna reprezentacija. Svaka čestica je predstavljena binarnim nizom dužine broja literala, odnosno n.

3.3 Inicijalizacija

Inicijalizacija je krajnje jednostavna. Vektor pozicije svake čestice inicijalizujemo slučajnim odabirom vrednosti 0 i 1. Vektor brzine može uzimati vrednosti od $[-1, 1]$.

```
def __init__(self, num_literals):
        self.num_literals = num_literals
        self.position = [randint(0,1) for x in range(num_literals)]
        self.best = self.position
        self.velocity = [2*random()-1 for x in range(num_literals)]
        self.fitness = float("-inf")
```

3.4 Fitnes funkcija

Fitnes funkcija je veoma važna za perfomanse algoritma. Prva fitnes funkcija koja se sama nameće jeste broj zadovoljenih klauza. Ali se ona nije pokazala kao dobra, pa je korišćen mehanizam stepenastog ažuriranja težina (stepwise adaptation weights) uvedena od strane Eiben-a [2]. Fitnes funkcija je data sledećom formulama:

$$Fitness(x) = \sum_{i=1}^{m} W_i C_i(x)$$

$$W_{i+1} = W_i + 1 - C_i(x^*)$$

Svaka klauza C_i ima težinu W_i. Ova funkcija ima za cilj identifikovanje težih klauza u procesu učenja koja je predstavljena većom vrednošću W_i. Na početku su težine inicijalizovane na 1, pa se potom ažuriraju. x^* je tekuće najbolje rešenje.

3.5 Lokalna pretraga i flip heuristika

3.6 Ažuriranje brzine i pozicije čestica

```
    def update_velocity(self, global_best, w, c1, c2):
        r1 = random()
        r2 = random()
        new_velocity = []
        for i in range(self.num_literals):
            new_velocity.append( w*self.velocity[i] + c1*r1*(self.best[i] - self.posi
        self.velocity = new_velocity

    def update_position(self):
        r = random()
        new_position = []
        for i in range(self.num_literals):
            position_i = 1 if r < self.sigmoid(self.velocity[i]) else 0
            new_position.append(position_i)
        self.position = new_position
```

3.7 Kriterijum zaustavljanja

Korišćena je kombinacija 2 kriterijuma: ako je dostignut maksimalan broj unapred zadatih iteracija ili ako je broj zadovoljenih klauza maksimalan, odnosno broj m.

3.8 Rezultati

Literatura

[1] Kennedy J, Eberhart RC. "A discrete binary version of the particle swarm algorithm," In: Proceedings of IEEE conference on systems, man and cybernetics , p. 4104–4109, 1997.

[2] Gottlieb J, and Voss N. "Adaptive fitness functions for the satisfiability problem". In: Parallel Problem Solving from Nature - PPSN VI 6th International Conference, Paris, France. Springer Verlag. LNCS 1917, 2000.

[3] Abdesslem Layeb, "A particle swarm algorithm for solving the maximum satisfiability problem", MISC Lab., Computer science department, University mentouri of Constantine, Algeria