Maximum satisfiability

Pavle Veličković

Uvod

MAXSAT, problem maksimalne zadovoljivosti, je NP-težak problem koji za skup U iskaznih promenljivih i kolekciju C klauza literala, gde je literal promenljiva ili negirana promenljiva iz skupa U, određuje valuaciju za skup U tako da je zadovoljen najveći mogući broj klauzula.

Ovaj problem se za manje instance može rešiti tako da se dobije tačno rešenje, brute force algoritmom ili nekim drugim rešavačem, ali je za veće instance potrebno aproksimirati rešenje ili razviti optimizacioni algoritam koji u većini slučajeva daje dovoljno dobro rešenje.

MAXSAT je aproksimabilan u 1.2987, što znači da se aproksimacijama može doći do rešenja koje je od optimalnog rešenja najviše manje ili veće za faktor 1.2987.

MAXSAT je takođe APX-kompletan, što znači da je moguće na njega redukovati ostale APX probleme.

Opis rešenja

Za rešenje problema koristio sam optimizacioni algoritam simuliranog kaljenja, koji prima broj promenljivih i listu klauza, a vraća najbolju pronađenu valuaciju i broj koji predstavlja podatak o tome koliko je klauza zadovoljeno. Rešenje je pisano u jeziku python, koristeći jupyter notebook.

U, skup iskaznih promenljivih, je lista boolean promenljivih.

C, lista klauza, je lista čiji su elementi liste celobrojnih vrednosti, gde pozitivan broj kao literal predstavlja promenljivu iz U, a negativan broj negiranu promenljivu iz U.

Funkcija cilja je evaluacija valuacije koja prima kao argumente neku valuaciju i listu klauza, a vraća broj zadovoljenih klauza tom varijacijom.

Algoritam počinje tako što se slučajno generiše početna valuacija, gde svaka promenljiva ima verovatnoću 0.5 da uzme vrednost True. Računa se vrednost funkcije cilja za početnu valuaciju i pamti kao lokalni i globalni maksimum. U svakoj iteraciji, generiše se nova valuacija tako da je razlika od prethodne promena samo jedne promenljive iz True u False ili obrnuto. Računa se nova vrednost funkcije cilja i, ako je bolja od lokalnog

maksimuma, vrednost funkcije i valuacija se pamte kao lokalni maksimum i valuacija koja ga dostiže. Ako odredimo da je valuacija u trenutnom lokalnom maksimumu, ispitujemo da li je i u trenutnom globalnom maksimumu. Ako jeste, pamtimo novu vrednost funkcije cilja kao globalni maksimum i pamtimo valuaciju koja je dostiže. Ako vrednost nije u lokalnom maksimumu, sa određenom verovatnoćom p, koja se smanjuje u svakoj iteraciji, dopuštamo da se nova vrednost tretira kao lokalni maksimum i da se iz nje nastavi, da bismo izašli iz lokalnog maksimuma koji potencijalno nije globalni. Ako odlučimo da to ne dopustimo, vraćamo skup promenljivih U u prethodno stanje i nastavljamo sledeću iteraciju. Posle određenog broja iteracija, vraća se do tada najbolja pronađena valuacija i vrednost funkcije cilja za tu valuaciju.

Algoritam simuliranog kaljenja je metaheuristika, optimizacioni algoritam koji se koristi za veliki broj problema gde nije uvek moguće odrediti tačno rešenje deterministički. Neke informacije o algoritmu se mogu naći na https://www.baeldung.com/cs/simulated-annealing

Eksperimentalni rezultati

Rezultati izvršavanja algoritma:

|U|, |C|	rezultat	200 iteracija	500 iteracija	1000 iteracija	2000 iteracija	4000 iteracija	6000 iteracija	8000 iteracija	brute force
10, 80	test primer 1	79/80	79/80	79/80	79/80	79/80	79/80	79/80	79/80
12, 90	test primer 2	87/90 - 90/90	90/90	90/90	90/90	90/90	90/90	90/90	90/90
8, 50	test primer 3	48/50	48/50	48/50	48/50	48/50	48/50	48/50	48/50
13, 2400	test primer 4	2232/2400 - 2242/2400	2239/2400 - 2242/2400	2239/2400 - 2242/2400	2242/2400	2242/2400	2242/2400	2242/2400	2242/2400
40, 1600	test primer 5	1579/1600 - 1586/1600	1585/1600 - 1589/1600	1586/1600 - 1590/1600	1588/1600 - 1590/1600	1588/1600 - 1590/1600	1589/1600 - 1590/1600	1588/1600 - 1590/1600	? (2^40 mogucih valuacija)
30, 2000	test primer 6	1961/2000 - 1969/2000	1967/2000 - 1971/2000	1968/2000 - 1971/2000	1970/2000 - 1971/2000	1971/2000	1971/2000	1971/2000	? (2^30 mogucih valuacija)
20, 4000	test primer 7	3848/4000 - 3854/4000	3852/4000 - 3854/4000	3852/4000 - 3854/4000	3854/4000	3854/4000	3854/4000	3854/4000	3854/4000
10, 1000	test primer 8	925/1000 - 929/1000	926/1000 - 929/1000	929/1000	929/1000	929/1000	929/1000	929/1000	929/1000

Vreme izvršavanja algoritma:

|U|, |C|	vreme izvrsavanja	200 iteracija	500 iteracija	1000 iteracija	2000 iteracija	4000 iteracija	6000 iteracija	8000 iteracija	brute force
10, 80	test primer 1	0.0s	0.0s	0.0s	0.0s	0.1s	0.1s	0.2s	0.0s
12, 90	test primer 2	0.0s	0.0s	0.0s	0.1s	0.1s	0.2s	0.3s	0.1s
8, 50	test primer 3	0.0s	0.0s	0.0s	0.0s	0.0s	0.1s	0.1s	0.0s
13, 2400	test primer 4	0.2s - 0.3s	0.5s - 0.7s	1.1s - 1.5s	2.1s - 2.8s	4.3s - 5.5s	6.3s - 9.0s	8.5s - 10.6s	9.4s
40, 1600	test primer 5	0.3s - 0.5s	0.9s - 1.2s	1.8s - 2.4s	3.6s - 4.8s	7.3s - 9.1s	11.2s - 15.0s	15.9s - 18.3s	predugo
30, 2000	test primer 6	0.3s - 0.4s	0.9s - 1.3s	1.8s - 2.3s	3.7s - 4.5s	7.4s - 9.1s	12.1s - 13.4s	14.9s - 17.6s	predugo
20, 4000	test primer 7	0.5s - 0.6s	1.3s - 1.6s	2.5s - 3.3s	5.2s - 6.6s	10.7s - 12.1s	16.7s - 19.5s	23.0s - 26.6s	52m 8.7s
10, 1000	test primer 8	0.0s - 0.1s	0.2s	0.3s - 0.5s	0.7s - 1.1s	1.4s - 1.8s	2.1s - 2.9s	2.9s - 3.9s	0.3s

Tabele sa slika prikazuju rezultate izračunavanja algoritma pri različitim brojevima iteracija, kao i vreme izvršavanja. Isti podaci postoje za brute force algoritam koji je redom prolazio kroz sve valuacije i računao funkciju cilja za svaku. Algoritam se u relativno kratkom vremenu izvršava u svim ispitanim primerima i pri dovoljnom broju iteracija se postiže velika tačnost. U slučaju

sa 20 promenljivih i 4000 klauza, brute force algoritmu je bilo potrebno preko 52 minuta da izračuna maksimalnu zadovoljivost, dok je algoritam simuliranog kaljenja već sa 2000 iteracija konzistentno nalazio tačno rešenje, za šta je bilo potrebno 5.2 do 6.6 sekunde. Algoritam simuliranog kaljenja već sa 2000 iteracija skoro uvek daje optimalno rešenje ili rešenje blizu optimalnog, a za izvršavanje algoritma u ispitanim primerima čak i za 8000 iteracija, vreme izvršavanja je ostajalo ispod 30 sekundi. Brute force algoritam i drugi deterministički rešavači ne bi za neke od ovih primera mogli da nadju rešenje u razumnom vremenu, jer ima previše mogućih valuacija. Algoritam simuliranog kaljenja se izvršava mnogo brže jer se samo i puta, gde je i broj iteracija, računa funkcija cilja, čija vremenska složenost zavisi samo od broja klauza (broj elemenata u listi C, |C|) i maksimalnog broja literala po klauzi, što je broj promenljivih (broj elemenata u listi U, |U|), dok se u brute force algoritmu funkcija cilja računa 2^|U| puta.

Eksperiment je vršen u Visual Studio Code okruženju, koristeći Jupyter Notebook ekstenzijum na operativnom sistemu Windows 10. Procesor računara je Intel i7-7700k, računar raspolaže sa 16GB RAM memorije.

Za generisanje test primera korišćen je generator DIMACS fajlova https://github.com/CompUtahs/Rand_DIMACS_gen/blob/master/r_dimacs_gen.c sa izmenom srandom() i random() funkcija u srand() i rand()

Zaključak

Simulirano kaljenje je jedan od mnogih optimizacionih algoritama i metoda aproksimacije koji se mogu primeniti na ovom problemu. Rezultati ukazuju na to da je algoritam dobar način za rešavanje ovog problema, ali postoji prostor za unapređivanje. Možda će algoritam češće nalaziti bolja rešenja uz bolji izbor početne valuacije. Algoritam bi svakako bio vremenski efikasniji ako se ne bi pamtile i kopirale valuacije koje su trenutno u lokalnom ili globalnom maksimumu, već kada bi se samo pratio broj zadovoljenih klauza i čuvao najveći.

Literatura

https://www.csc.kth.se/~viggo/wwwcompendium/node225.html

https://math.mit.edu/~goemans/18434S06/max-sat-phil.pdf

https://www.baeldung.com/cs/simulated-annealing

https://arxiv.org/pdf/2111.01551.pdf