IBPSO za problem maksimalnog pakovanja skupova

Projekat u okviru kursa
Računarska inteligencija
Matematički fakultet

Mladen Dobrašinović

dobrasinovic.mladen@gmail.com

11. januar 2021.

Sažetak

Karakteristika NP-teških problema je da u suštini, nema efikasnih algoritama za njihovo rešavanje u vreme pisanja ovog rada. Ipak, oni zanimaju veliki broj ljudi i traženje algoritma za rešavanje i njihovu aproksimaciju može da bude korisno. Maksimalno pakovanje skupova je jedan od takvih problema, a PSO je jedan od pristupa koji nas može zanimati u primeni na ovom problemu, kao način aproksimacije rešenja istog. Ovaj rad predstavlja upotrebu IBPSO algoritma za aproksimaciju problema MSP.

Sadržaj

1 Uvod

Problem maksimalnog pakovanja skupova je nalaženje skupa disjunktnih skupova $C' \subseteq C$, gde je C skup konačnih skupova, takvog da se maksimizira kardinalnost pakovanja, tj. $|C'|$. [1] Za bliske probleme, poznato je da je pakovanje skupova NP-kompletno [2], a da je maksimalno k-pakovanje skupova, gde je kardinalnost skupova u C ograničeno sa $k \geq 3$ APX-kompletno [1].

Prethodno su za rešavanje srodnih problema primenjene optimizacija kolonijom mrava [2], GRASP tehnika [3]. Dostupne su gotove instance ovog problema koje se mogu korisititi za optimizaciju i testiranje tehnika, kao i metode za generisanje novih instanci, posebno se mogu naći instance konstruisane tako da imaju teško nalazive maksimume za ovaj problem [4][5].

Postoje rezultati za granice efikasnog aproksimiranja problema maksimalnog k-pakovanja skupova [6].

2 Implementacija algoritma

Algoritam koji je implementiran kao rešenje problema, i predstavlja predmet ispitivanja ovog rada je zasnovan na osnovnom PSO, s tim što ne predstavlja klasični BPSO, već njegovu modifikaciju. Klasični BPSO može biti neintuitivan i meni nije delovao kao najbolje rešenje za ovaj problem, uz moj pristup. On ne odgovara potpuno originalnom PSO, pošto, između ostalog, trenutna pozicija čestice ne utiče na sledeću. Iz ovih razloga sam koristio IBPSO. IBPSO je preuzet iz literature, pri tome što je prilagođen za konkretni problem, i predstavlja zanimljivu, i vernu, viziju binarne implementacije PSO, koja takođe nema element inercije pri promeni pozicije. Mogući nedostatak je manja stohastičnost pri biranju novih pozicija, sa kojom dolazi do potpune konvergencije rešenja. [7] [8]

U mojoj implementaciji, ima nekih značajnih karakteristika, koje se i razlikuju od rada koji je inspirisao osnovni algoritam. Kodiranje potencijalnog rešenja, tj. čestice, je binarni niz koji govori da li skup pripada rešenju ili ne. Pri inicijalizaciji rešenja, manja je verovatnoća pojavljivanja 1 nego 0, jer dobra rešenja za ovaj problem u mom iskustvu izgledaju češće ovako. Balansirani pristup, vodio je ka preranoj konvergenciji u preliminarnom testiranju, u kome je oblikovana konačna implementacija.

Funkcija pogodnosti je prirodno, veličina pakovanja skupa koje rešenje predstavlja. Neispravna rešenja, tj. ona koja bi uključila skupove koji imaju preklapanja su kažnjavna sumom prekomernih pojavljivanja elemenata u potencijalnom rešenju, i ne ocenjuje im se broj skupova koji su uključili. Ovo se pokazalo kao dobro rešenje, jer u početnom skupu obično nema ispravnog rešenja, a ovo vodi konvergenciji ka ispravnom i dobrom rešenju u velikom broju slučajeva.

Korišćena je Von Nojmanova topologija povezanosti čestica, pri čemu su u razvoju probane i druge (*gbest* i *lbest*) [8], ali se nisu pokazale kao značajno bolje ili posebno zanimljive. Broj čestica se u roju bira dinamički u skladu sa brojem potencijalnih skupova u instanci, što se takođe pokazalo kao važno za dobru konvergenciju u većini instanci, i bira se prvi kvadratni broj koji je veći od broja skupova. Ovo omogućava da se čestice organizuju u kvadratnu topologiju.

Kriterijum zaustavljanja je prelazak maksimalnog broja iteracija (700), ili potpuna konvergencija svih čestica ka istom rešenju. Pri testiranju se

pokazalo da i posle velikog broja iteracija (više stotina) može doći do poboljšanja.

Mehanizam promene pozicije čestice je slučajno pomeranje pojedinačnih binarnih vrednosti u rešenju ka svom najboljem, ili najboljem u komšiluku čestica. Formula je relativno složena, može se videti u implementaciji 1 ili radu [7].

```
# Racunamo brzinu po slozenoj formuli.
omega1 = self.rand_bool()
omega2 = self.rand_bool()

social_diff = [t[0] != t[1] for t in zip(social_pos, self.pos)]
social = [t[0] and t[1] for t in zip(omega1, social_diff)]

cog_diff = [t[0] != t[1] for t in zip(cog_pos, self.pos)]
cog = [t[0] and t[1] for t in zip(omega2, cog_diff)]

# Brzina predstavlja bitove koje treba promeniti.
 self.velocity = [t[0] or t[1] for t in zip(cog, social)]
```

Kod 1: Računanje nove pozicije čestice

3 Rezultati ispitivanja

Da bi se testirao IBPSO, pošto u literaturi nema jednostavnih metoda koje smo mogli da primenimo, ili rezultata koji se mogu direktno uporediti, razvijena je metoda Monte Karlo za ovaj problem. Monte Karlo algoritam radi ekvivaletno odabiranju nasumične permutacije skupova i uzimanja najvećeg pakovanja koje možemo dobiti ako krenemo od početka i dodajemo skupove dok ne dođe do preklapanja. Takođe je razvijen *brute-force* algoritam da bi se testirala optimalnost rešenja. Za određene instance je dostupno optimalno rešenje na njihovom izvoru [5]. IBPSO se generalno pokazao kao dobra alternativa ovom algoritmu, za koji je teško dobiti bolje rezultate povećanjem broja iteracija, jer je potrebno veliko uvećanje da bi se došlo do boljih rezultata. Primer upoređenja je na slici 1, koja je zasnovana na instancama koje su teške za rešavanje [5].

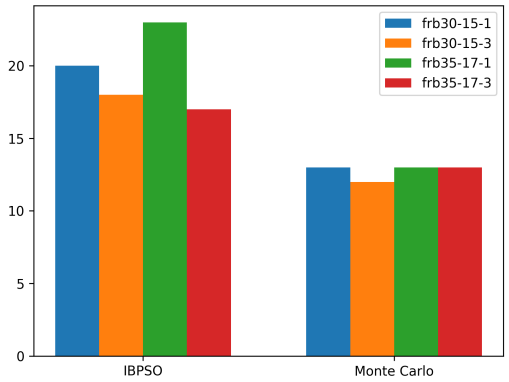
3.1 Puni eksperimentalni rezultati

Potpuni rezultati testiranja su dostupni na GitHub repozitorijumu projekta, a ovde će biti predstavljeni reprezentativni primeri. Za male instance je korišćena gruba sila, za koju se pokazalo da treba previše vremena za instance preko 50 primeraka na mom računaru. Najveće instance [4], nisu rešavane, jer se pokazalo da previše vremena treba za to, i rešenje nije realno upotrebljivo. Sažeti rezultati su prikazani na tabeli 1.

Korišćen je *python3.6*, testiranje je vršeno na računaru sa procesorom *Intel i5-3300*, i rađeno je paralelno.

4 Zaključak

IBPSO algoritam je upotrebljiv za aproksimaciju problema maksimalnog pakovanja skupova. Konvergencija se tipično odvija tako što se počne od rešenja se negativnom ocenom (koje je neprihvatljivo), čija se nedopustivost smanjuje dok ne dođe do upotrebljivog. Nakon toga algoritam unapređuje prihvatljivo rešenje do konačne konvergencije. Ako bismo želeli da unapredimo algoritam, koraci u testiranju bi mogli biti dodavanje



Slika 1: Rezultati na „teškim" instancama. 30 i 35 predstavljaju optimalno rešenje.

Tabela 1: Rezultati testiranja sve tri metode

Naziv	IBPSO	Monte-Karlo	Gruba sila
self30rnd1	9	8	9
self30rnd2	6	8	9
self50rnd1	15	12	15
self50rnd2	15	13	16
pb_100rnd3	34	18	
pb_100rnd4	33	17	
pb_200rnd1	55	24	
pb_200rnd2	47	24	
pb_500rnd1	83	21	
pb_500rnd2	89	25	
pb_500rnd3	103	33	
pb_500rnd4	106	34	

različitih težina verovatnoći promene ka ličnom najboljem rešenju, odnosno najboljem u susedstvu. Moguće je eksperimentisanje i sa ponovnim pokretanjem, koje daje bolja rešenja pri kratkoj analizi koja je izvršena, novim topologijama i većim skupom čestica, i podešavanjem parametara pretrage. Prelaženje na efikasniji programski jezik bi omogućilo paralelizam u rešavanju problema, i ubrzalo samu pretragu, pa bi ovaj pristup mogao dati upotrebljivije rešenje u nekim slučajevima.

Problem maksimalnog pakovanja skupova je prilično složen, i povezan je za NP-teškim problemima, što vodi do teškog nalaženja algoritma koji bi mogao da zadovolji proizvoljne kriterijume koji mogu nastati u potrebi rešavanja problema svedenog na isti. Zanimljiv je pristup u [2], dok se rešenje predstavljeno u ovom radu pokazalo kao ograničeno vremenom i sposobnošću konvergencije na težim zadacima.

Literatura

[1] Viggo Kann. Maximum set packing, 2000. at: https://www.csc.kth.se/~viggo/wwwcompendium/node144.html.

[2] Xavier Gandibleux et al. An Ant Colony Optimisation Algorithm for the Set Packing Problem. 2004.

[3] Xavier Delorme et al. GRASP for set packing problems. 2010.

[4] Xavier Delorme. Benchmarks for the Set Packing Problem, 2020. at: https://www.emse.fr/~delorme/SetPacking.html.

[5] Ke Xu. Benchmarks with Hidden Optimum Solutions for Set Covering, Set Packing and Winner Determination, 2006. at: http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/set-benchmarks.htm.

[6] Elad Hazan et al. On the complexity of approximating k-set packing. 2006.

[7] Xiaohui Yuan et al. An improved binary particle swarm optimization for unit commitment problem. 2009.

[8] Andries Engelbrecht. *Computational Intelligence - An Introduction, 2nd edition.* 2007.