Univerzitet u Beogradu
Matematički fakultet

Maximum Betweenness

Autor: Vukašin Marković

Profesor: dr Vladimir Filipović

Asistenti: Stefan Kapunac i Ivan Pop Jovanov

Datum: april 2024

SADRŽAJ

1. Uvod

U ovom radu obrađuje se dobro poznati problem "MAXIMUM BETWEENNESS", koji spada u klasu NP-teških problema, odnosno problema za koje ne postoji algoritam koji bi ih rešio u polinomijalnom vremenu. Stoga će se koristiti sirove tehnike optimizacije za rešavanje ovog problema.

Nakon uvoda, biće opisan sam problem "MAXIMUM BETWEENNESS", a zatim implementacija algoritama i razvoj optimizacionih metoda za njegovo rešavanje. Za rešavanje ovog problema koristiće se različiti pristupi poput grube sile i njene BNB optimizacije, pohlepnih algoritama kroz različite vrste lokalne pretrage, razne varijacije simuliranog kaljenja, kao i genetski algoritam.

Na kraju, biće analizirani rezultati dobijeni primenom navedenih metoda i izneće se zaključci na osnovu njih.

2. MAXIMUM BETWEENNESS

Opis problema:

Postoji skup elemenata A i kolekcija uređenih trojki (a, b, c), pri čemu su a, b i c različiti elementi iz skupa A. Cilj je pronaći jednoznačnu funkciju f koja preslikava elemente iz skupa A u opseg od 1 do veličine skupa A, tako da se maksimalizuje broj trojki u kojima se poštuje određeni redosled, tj. ili $f(a)<f(b)<f(c)$, ili $f(c)<f(b)<f(a)$.

Prilikom razvoja rešenja za ovaj NP-težak problem, ključno je osigurati da optimizacione metode mogu rukovati velikim skupovima podataka i pružiti pouzdane rezultate. Validacija tačnosti rezultata optimizacionih metoda izvršiće se upoređivanjem sa algoritmima grube sile (Brute-force) na manjim instancama problema.

3. Implementacija algoritma

3.1 Gruba sila

Brute-force algoritam obezbeđuje uporednu analizu optimizacionih metoda. Efikasan je za obradu manjih instanci problema i daje optimalna rešenja, što omogućava proveru tačnosti optimizacionih pristupa. Međutim, postaje nepraktičan za veće instance našeg problema, kako se dimenzije skupa A povećavaju.

U ovom radu su implementirani:

- Brute-force algoritam sa obezbeđenim deterministčkim ponašanjem.
- Unapređenje Brute-force algoritma primenom ideje iz Branch and Bound (BNB) tehnike. Ova tehnika selektivno istražuje grane pretrage koje imaju potencijal da pruže bolje rezultate od trenutno najboljeg rešenja, dok odbacuje ostale grane. To značajno smanjuje prostor pretrage i omogućava brže pronalaženje optimalnog rešenja

3.2 Pohlepni algoritam

Kada je primarni cilj optimizacija vremenske složenosti rešavanja problema, koriste se pohlepni algoritmi. Oni omogućavaju brzo izvršavanje, ali ne garantuju uvek najoptimalnije rešenje kao što to čine algoritmi grube sile. U ovom istraživanju su implementirani algoritmi lokalne pretrage, koji generišu okolinu trenutnog rešenja i iz nje biraju sledeće.

Implementirana su dva pristupa za odabir sledećeg rešenja iz okoline:

* ***local_search_first_improvement:*** Odabir sledećeg rešenja iz okoline vrši se tako što se uzima prvo rešenje koje je bolje od trenutnog.

* ***local_search_best_improvement:*** Algoritam prolazi kroz sva rešenja iz okoline i bira ono koje je najbolje za sledeće.

U oba slučaja pretraga se zaustavlja ako se algoritam zaglavi u lokalnom maksimumu, odnosno ako trenutno rešenje ne bude moglo da se unapredi u neko drugo iz okoline.
Takođe, za oba algoritma, definisano je gornje ograničenje - maksimalan broj iteracija za koje će biti moguće unapređivanje trenutnog rešenja nekim iz okoline.

3.3 Simulirano kaljenje

Glavni nedostatak pohlepnih algoritama je usmeravanje pretrage samo ka trenutno najboljem sledećem rešenju, bez razmatranja manje povoljnih ili lošijih opcija, iako ga one kasnije mogu dovesti do znatno boljeg krajnjeg resenja. Tada je bolje koristiti algoritam inspirisan procesom fizičkog kaljenja metala, poznat kao algoritam simuliranog kaljenja.

Ključna ideja algoritma je simulacija procesa hlađenja metala.
Na početku procesa, metal je najusijaniji i najpodložniji velikim promenama svog oblika.
- *to je momenat kada algoritam uzima u obzir i lošija rešenja i prihvata ih sa većom verovatnoćom.*

Vremenom kako temperatura metala opada, sve je teže izvršiti veliku promenu njegovog oblika i tada se trudimo da što više poboljšamo već ustanovljeni oblik.
- *to je momenat kada naš algoritam počinje da teži pohlepnom algoritmu, odnosno smanjuje verovatnoću prihvatanja lošijih rešenja i za sledeća rešenja uzima samo ona koja su bolja od trenutnog.*

Ova tehnika ne zahteva iscrpnu pretragu celokupnog prostora pretrage, već iterativno unapređuje trenutno rešenje i korisna je kod velikih i složenih prostora pretrage.

Pri implementaciji algoritma simuliranog kaljenja korišćen je **Metropolisov kriterijum**.
Kada je temperatura visoka, Metropolisov kriterijum je veći, što znači da je veća verovatnoća prihvatanja lošijeg rešenja. Kako temperatura tokom vremena opada, verovatnoća prihvatanja lošijeg rešenja takođe opada, što omogućava algoritmu da se postepeno fokusira samo na poboljšavanje kvaliteta rešenja.

e_next = *rešenje koje razmatramo za sledece*
e_curr = *rešenja koje imamo za trenutno*
ΔE = ***e_next - e_curr***
T = *temperatura*

Algoritam je podeljen u dva slučaja u zavisnosti od vrednosti ***ΔE***:

1) Ako je razlika ***ΔE*** pozitivna:
e_next je bolje od ***e_curr***, sto znači da imamo unapređenje i naš ***e_curr*** postaje ***e_next***

2) Ako je razlika ***ΔE*** negativna:
U ovom slučaju, rešenje ***e_next*** ne unapređuje trenutnu pretragu. Ipak, postoji verovatnoća da će biti prihvaćeno (da će ***e_curr*** postati ***e_next***). Ova verovatnoća je određena Metropolisovim kriterijumom.

Verovatnoća definisana Metropolisovim kriterijumom je izrazena preko $e^{\frac{\Delta E}{T}}$ funkcije

U nastavku je definisan ***"primer vrednosti funkcije Metropolisovog kriterijuma"*** za negativnu vrednost ***ΔE*** *i različite vrednosti* ***T*** :

a. Za negativnu vrednost ***ΔE*** i visoku vrednost ***T***:
- ***ΔE*** = -10 (na primer, ***e_next*** lošije za 10 jedinica u odnosu na ***e_curr***)
- **T** = 100 (visoka temperatura)
- e^(-10 / 100) = e^(-0.1) ≈ 0.9048

Zaključak: Verovatnoća prihvatanja ***e_next*** je oko 0.9048,
što znači da je vrlo verovatno da će ***e_next*** biti prihvaćen.

b. Za negativnu vrednost ***ΔE*** i nisku vrednost ***T***:
- ***ΔE*** = -10 (***e_next*** lošije za 10 jedinica u odnosu na ***e_curr***) kao u primeru pod 'a'
- ***T*** = 10 (niska temperatura)
- e^(-10 / 10) = e^(-1) ≈ 0.3679

Zaključak: Verovatnoća prihvatanja ***e_next*** je oko 0.3679,
što znači da je manja verovatnoća da će ***e_next*** biti prihvaćen.

zaključak iz našeg primera:
U oba slučaja, kada je ***ΔE*** negativan, novi kandidat ***e_next*** će biti prihvaćen sa većom verovatnoćom pri višoj temperaturi, dok će se verovatnoća prihvatanja smanjivati kako temperatura opada. Ovo omogućava algoritmu da istraži različite delove prostora rešenja na početku, a zatim se fokusira na poboljšavanje kvaliteta rešenja kako pretraga napreduje.

Illustration 1: primer vrednosti funkcije Metropolisovog kriterijuma

U ovom istraživanju, razvijena su ćetiri različita tipa algoritma simuliranog kaljenja, koji su nazvani:

1) **simulated_annealing** - Ovaj tip koristi standardnu linearnu strategiju hlađenja temperature. Nakon svake iteracije, trenutna temperatura se množi sa unapred definisanim parametrom između 0 i 1.

2) **simulated_annealing_cooling_strategy** - Ovaj tip omogućava korisniku da algoritmu prosledi funkciju hlađenja temperature. Implementirane funkcije hlađenja u ovoj verziji uključuju geometrijsku, korensku i logaritamsku strategiju.

3) **simulated_annealing_adaptive** - Ovaj tip dinamički određuje hlađenje temperature. Algoritam prati brojače prihvaćenih i ispitanih rešenja, koristeći ih da bi svaka iteracija petlje definisala trenutnu stopu prihvatanja novih rešenja. Funkcija "adaptive_cooling" prilagođava temperaturu u zavisnosti od stope prihvatanja novih rešenja, smanjujući je ako je stopa prihvatanja veća od ciljane, inače je blago povećava.

Svi ovi tipovi dele zajedničku karakteristiku - funkciju za generisanje suseda, koja vrši izmenu trenutnog rešenja i vraća novo rešenje. Za razliku od lokalne pretrage, koja generiše celu okolinu suseda za trenutno rešenje, ovi tipovi algoritma simuliranog kaljenja generišu samo jednog suseda, koji može biti bolje ili lošije rešenje u odnosu na trenutno.

4) **simulated_annealing_with_intensive_search**.

"Intenzivna pretraga" u kontekstu naziva algoritma označava da algoritam koristi agresivniji pristup pri generisanju suseda i odabiru najboljeg među njima. Umesto samo nasumičnog generisanja suseda, kao što je to slučaj u prethodno implementiranim verzijama algoritma, ova verzija koristi posebnu heuristiku ili metodologiju koja se fokusira na pronalaženje što optimalnijeg nasumično generisanog suseda. Dakle, "intenzivna pretraga" ukazuje na to da se algoritam trudi da pažljivije istraži prostor rešenja kako bi pronašao što bolje rešenje.

Ovaj pristup se razlikuje od prethodnih verzija algoritma jer algoritam pažljivije istražuje prostor rešenja, ciljajući na pronalaženje najboljeg rešenja. Tokom analize prethodnih implementacija algoritama i njihovih rezultata, linearna i geometrijska strategija hlađenja su pokazale se kao najefikasnije. Stoga je u ovom algoritmu primenjena linearna strategija hlađenja zbog njenih dokazanih rezultata.

Funkcija koja omogućava ovu pažljiviju pretragu i selekciju suseda je "generate_neighbor_with_heuristic". Ova funkcija generiše n suseda, od kojih se zatim bira najbolji. Tako odabrani sused postaje kandidat za sledeće rešenje koje algoritam razmatra.

Prednosti simuliranog kaljenja:

- Efikasno pretraživanje prostora: Simulirano kaljenje uspešno može istražiti velike prostore pretrage i pronaći globalne optimume, što ga čini korisnim za različite optimizacione probleme.
- Prilagodljivost temperature: Temperatura u simuliranom kaljenju može se prilagoditi tokom izvršavanja algoritma, što omogućava različite strategije pretrage i prilagođavanje različitim vrstama problema.
- Verovatnosno prihvatanje rešenja: Metropolisov kriterijum omogućava algoritmu da prihvati lošija rešenja sa određenom verovatnoćom, čime se sprečava zaglavljivanje u lokalnim optimumima.
- Jednostavna implementacija: Simulirano kaljenje je relativno jednostavno za implementaciju i ne zahteva složene strukture podataka ili procese.
- Robusnost: Algoritam se može prilagoditi različitim problemima i nije previše osetljiv na početne uslove kao neki drugi optimizacioni algoritmi.

Mane simuliranog kaljenja:

- Podešavanje parametara: Algoritam ima nekoliko parametara koji se moraju odabrati, što može biti izazovno jer loše odabrane vrednosti mogu uticati na efikasnost algoritma.

- Konvergencija: Iako simulirano kaljenje ima mehanizam za izbegavanje lokalnih optimuma, nije garantovano da će uvek pronaći globalni optimum, posebno za probleme sa visokim dimenzijama ili nelinearnim funkcijama cilja.

- Efikasnost: Za neke probleme, simulirano kaljenje može biti manje efikasno u poređenju s drugim optimizacionim algoritmima, posebno kada su poznate specifične karakteristike problema koje se mogu iskoristiti.

- Zavisnost od temperature: Efikasnost simuliranog kaljenja zavisi od odgovarajućeg odabira rasporeda temperature. Loše podešavanje temperature može rezultirati lošim performansama algoritma.

Ipak, u ovom radu, za MAXIMUM BETWEENNESS problem neke verzije simuliranog kaljenja su se pokazale kao prilično efikasne, i po pitanju vremenske složenosti, i po pitanju optimalnih rezultata.

3.4 Genetski algoritam

Još jedna tehnika koja je korišćena prilikom rešavanja MAXIMUM BETWEENNESS je tehnika genetskih algoritama. Genetski algoritmi su metaheurističke tehnike inspirisane procesom evolucije u prirodi. Ovi algoritmi koriste princip selekcije, ukrštanja i mutacije kako bi pretražili prostor rešenja i pronašli optimalna ili suboptimalna rešenja za različite probleme. Često se koriste u optimizacionim problemima, problemima kombinatorne optimizacije i problemima pretrage prostora stanja.

Implementacija genetskog algoritma uključuje stvaranje početne populacije, evaluaciju rešenja, selekciju roditelja, ukrštanje između roditelja i primenu mutacije. Kroz iteracije generacija, algoritam evoluira populaciju ka željenom rešenju.

Genetski algoritmi su odabrani za ovaj problem zbog njihove sposobnosti rada sa velikim prostorima pretrage i sposobnosti pronalaženja globalnih optimuma. Takođe, oni su pogodni za probleme koji nemaju jasno definisanu strukturu ili za probleme u kojima je teško pronaći optimalno rešenje klasičnim metodama. Kroz iterativni proces selekcije, ukrštanja i mutacije, genetski algoritam teži ka pronalaženju rešenja koja zadovoljavaju kriterijume optimalnosti za naš problem.

4.0 Testiranje algoritama i njihovi rezultati

Pre nego što je pristupljeno testiranju prethodno opisanih algoritama nad nekim podacima, oni su prvenstveno generisani i serijalizovani u datoteku data_sets.pkl.
Iz pomenute datoteke podaci su kasnije deserijalizovani i korišćeni prilikom testiranja.
Podatke smo podelili u tri skupa

data_sets_BF
data_sets
data_sets_extra

legenda:
BF = brute_force
BF_BNB = brute_force_BNB
LSFI = local_search_first_improvement
LSBI = local_search_best_improvement

|A|	|C|	BF	time [sec]	BF_BNB	time [sec]	LSFI	time [sec]	LSBI	time [sec]
5	4	3	0	3	0.001	3	0	2	0
7	14	10	0.038	10	0.033	10	0.001	10	0.001
10	20	14	18.811	14	15.096	11	0.003	10	0.003
25	50	-	too long	-	too long	38	0.068	36	0.106
50	100	-	too long	-	too long	81	1.766	74	2.092
100	200	-	too long	-	too long	155	35.654	159	54.253
125	250	-	too long	-	too long	191	88.907	198	173.201
150	300	-	too long	-	too long	242	299.812	233	396.027
175	350	-	too long	-	too long	283	610.709	277	810.542

SA = simulated_annealing
SACS = simulated_annealing_cooling_strategy

|A|	|C|	SA	time [sec]	SACS geometry	time [sec]	SACS sqrt	time [sec]	SACS log	time [sec]
5	4	3	0.029	3	0.025	3	0.058	3	0.053
7	14	10	0.033	10	0.032	10	0.052	10	0.059
10	20	14	0.027	14	0.028	13	0.07	12	0.069
25	50	40	0.065	38	0.07	32	0.216	28	0.217
50	100	80	0.205	79	0.202	52	0.597	52	0.594
100	200	152	0.669	148	0.65	99	2.004	90	2.181
125	250	171	1.021	178	0.995	118	3.05	110	3.184
150	300	205	1.393	222	1.463	140	4.255	124	4.925
175	350	254	1.844	247	2.122	158	5.944	147	5.696

|A|	|C|	SA adaptive	time [sec]	SA with intensive search	time [sec]	Genetic algorithm	time [sec]
5	4	2	0.053	3	0.197	3	0.541
7	14	6	0.061	10	0.206	10	0.896
10	20	5	0.069	14	0.272	14	1.258
25	50	18	0.209	45	0.86	29	3.752
50	100	26	0.615	90	2.465	55	11.259
100	200	84	2.186	174	8.98	86	39.942
125	250	79	3.079	207	12.627	104	60.23
150	300	106	4.427	254	19.042	130	83.224
175	350	118	5.693	297	25.537	144	111.108

5.0 Zaključak

Dakle da ponovimo,
Podaci nad kojima su testirani pomenuti algoritmi su podeljeni u 3 skupa

data_sets_BF	- maksimalne kardinalnosti 10 za skup |A|, 20 za skup |C|
data_sets	- maksimalne kardinalnosti 100 za skup |A|, 200 za skup |C|
data_sets_extra	- maksimalne kardinalnosti 150 za skup |A|, 300 za skup |C|

Brute force algoritam i njegova **BNB** optimizacija su pokazali tačne rezultate na manjim skupovima podataka (data_sets_BF), ali nisu bili efikasni za veće instance zbog loše vremenske složenosti. Ipak, ovi algoritmi su korisni kao referenca za proveru tačnosti drugih pristupa.

Pohlepni algoritmi (LSFI, LSBI) nisu se pokazali dovoljno dobro na manjim skupovima podataka (data_sets_BF), ali su se pokazali uspešnima na većim instancama (data_sets i data_sets_extra). Međutim, vreme izvršavanja je postajalo problematično kako su se dimenzije podataka povećavale, nagoveštavajući potrebu za efikasnijim pristupima. Na vreme izvršavanja ovog algoritma najvise je uticala neefikasna implementacija funkcije za biranje suseda (gde bi se za svako trenutno rešenje generisala čitava okolina suseda), stoga je u narednim algoritmima stavljen akcenat na unapređenje te funkcije.

Testirani su **algoritmi simuliranog kaljenja** koji pružaju brže vreme izvršavanja i mogućnost izlaska iz lokalnih optimuma. Među njima, najgore se pokazao algoritam sa adaptivnim hladjenjem (**SA adaptive**), i on nije dao optimalne rezultate ni na malim, ni na velikim instancama. Stoga, dinamičko hlađenje nije uzeto u obzir pri razvoju konačnog najboljeg rešenja.
Nakon toga, algoritmi sa logaritamskom (**SA log**) i korenskom (**SA sqrt**) strategijom hlađenja takođe nisu dali zadovoljavajuće rezultate, ali sa druge strane, simulirano kaljenje sa linearnom strategijom hlađenja (**SA**) i geometrijskom strategijom hlađenja (**SA geometry**) pokazali su najbolje rezultate za relativno kratko vreme izvršavanja. Obzirom na brzo vreme izvršavanja ova dva algoritma, otvoren je prostor za poboljšanje tačnosti ta dva algoritma na račun dužeg vremena izvršavanja.
Zbog prethodno pomenutog otvorenog prostora, za finalni algoritam simuliranog kaljenja, odabrana je linearna strategija hladjenja uz dodatak heuristike za navođenje pretrage. Ova heuristika produžava vreme izvršavanja algoritma ali i obezbeđuje bolje rezultate. Taj algoritam je **simulirano kaljenje sa intenzivnom pretragom**, i pokazao se kao najbolji za rešavanje NP teškog problema MAXIMUM BETWEENNESS. Dao je najbolje rezultate, pokazavši se odlično i na malim i na velikim instancama našeg problema. Parametri su mu podešeni tako da njegovi rezultati imaju efikasno vreme izvršavanja i da budu što optimalniji. Svako dalje poboljšavanje parametara (odnosno povećavanje parametara koji su srazmerni sa vremenom izvršavanja) ne bi dalo bolje rezultate, a povećalo bi vreme izvršavanja algoritma.

Genetski algoritam je dao zadovoljavajuće rezultate na malim instancama problema (data_sets_BF), ali je pokazao nedovoljnu efikasnost za veće instance, s obzirom na vreme izvršavanja. Povećanje parametara kao što su veličina populacije i broj generacija obezbeđuje optimalnije rezultate, ali bi dovodi do produženja vremena izvršavanja algoritma, tako da je algoritam neekonomičan.

U konačnom zaključku, simulirano kaljenje sa intenzivnom pretragom se izdvojilo kao najbolji algoritam za rešavanje problema MAXIMUM BETWEENNESS, pružajući optimalne rezultate kako na manjim tako i na većim instancama problema. Parametri su podešeni tako da pruže efikasno vreme izvršavanja i što optimalnije rezultate. Svako dalje povećanje parametara ne bi doprinelo poboljšanju rezultata, već bi samo produžilo vreme izvršavanja algoritma.

6.0 Literatura

- https://github.com/MATF-RI/Materijali-sa-vezbi
- Computational Intelligence - An Introduction, Andries Engelbrecht, John Willey & Sons, 2007.
- https://www.machinelearningplus.com/machine-learning/simulated-annealing-algorithm-explained-from-scratch-python/