Primjena “Greedy” metoda na rješavanje problema trgovačkog putnika

Ognjen Bostjančić

Napredni algoritmi i strukture podataka
Teorijska kompjuterska nauka
Odsjek za matematiku
Prirodno-matematički fakultet
Sarajevo, Bosna i Hercegovina
ognjen.bostjancic@gmail.com

I. Uvod

Problem trgovačkog putnika se definiše na sljedeći način. Za dati skup gradova i cijena putavanja između svaka dva grada iz skupa, naći najjeftiniju turu koja obilazi svaki grad jednom i vraća se u početni grad. Tj. problem je ekvivalentan problemu pronalaženja Hamiltonovog ciklusa u potpunom neusmjerenom težinskom grafu. Pokazuje se da je vremenska kompleksnost iscrpne pretrage za ovaj problem $O(n!)$ dok je metodom dinamičkog programiranja moguće problem riješiti u vremenu $O(n \cdot 2^n)$.

Obzirom da ni metod iscrpne pretrage, a ni metod dinamičkog programiranja ne daju rezultate u realnom vremenu za dovoljno veliko n, problem trgovačkog putnika se najčešće pokušava riješiti nalaženjem dovoljno jeftine ture u realnom vremenu. Jedan od pristupa za nalaženje takve ture je korištenje pohlepnih (eng. Greedy) algoritama. Radi se o algoritmima koji rješenje nalaze kroz sekvencu izbora, i koji se u svakoj iteraciji odlučuju za izbor koji je u tom trenutku najbolji.

Primjeri takvih algoritama za rješavanje problema trgovačkog putnika bili bi:

- Metod najbližeg susjeda (u svakoj iteraciji uzima grad koji je najbliži zadnjem dodanom gradu i dodaje ga u turu)
- Metod najmanje grane (u svakoj iteraciji dodaje granu čiji se krajevi ne nalaze već u turi.)

Također, moguće je primjeniti i sljedeći “Greedy” algoritam na problem trgovačkog putnika:

Izabere se grad *0*, i napravi se tura na način *0, 1, 0, 2, 0, 3, 0, ...* (ovo nije validna tura u smislu TSP rješenja jer se ne posjećuje svaki grad jednom). Dakle, početna tura ide do svakog grada i odmah se vraća do početnog. Tada se pravi greedy izbor: izabere se par gradova (i, j), $i, j \neq 0$, za koji se dobija najveća ušteda ako se ide od grada i do grada j (umjesto da se vraća do početnog grada). Ovaj postupak se ponavlja dok se ne sastavi validna tura za TSP problem.

II Implementacija

Na početku, potrebno je od date ture dužine n, kreirati turu dužine $2 \cdot n - 2$ koja će na parnim pozicijama (p, počevši od nule) imati početni grad, a na svakoj neparnoj poziciji imati grad koji se u početnoj turi nalazio na poziciji $\frac{p}{2} + 1$. Za početni grad je uzet prvi grad u datoj turi. Nakon toga, potrebno je u *n - 1* koraka pronaći uređen par *(i, j)* za koji se ostvaruje najveća

ušteda (izbacivanjem početnog grada). Tada se tura mijenja *i* grad *j* zauzima poziciju iza grada *i* u turi, a početni grad koji je bio ispred *j*, se izbacuje iz ture. Medjutim, već u drugom koraku je moguć sljedeći slučaj. Neka je nakon prvog koraka algoritma tura u sljedećem obliku:

0, i, j, 0, k, 0, ...

Pretpostavimo da se najveća ušteda ostvaruje ako se ide od grada *k* do grada *i*, U ovom slučaju nije moguće jednostavno prebaciti grad i na poziciju iza grada *k*, jer je *i* već u "povezanom dijelu ture", odnosno komponenti sa gradom *j*. Obzirom da grana *(i, j)* mora biti dio krajnje ture, potrebno je čitavu komponentu prebaciti iza grada *k*, pa tura nakon drugog koraka treba da izgleda ovako:

0, k, i, j, 0, ...

Moguć je još i kompleksniji slučaj: Neka je nakon m-tog koraka tura u sljedećem obliku:

0, i, ..., j, 0, k, ..., l, ...

Neka se najveća ušteda ostvaruje ako se ide od grada *l* do grada *i*. Sada su ovo dvije povezane komponente, pa je potrebno čitavu komponentu koja počinje od grada *i* prebaciti iza grada *l*, da bi tura izgledala ovako:

0, k, ..., l, i, ..., j, 0, ...

Jasno je da će u svakom od *n - 1* koraka (jer toliko se puta početni grad javlja u turi, ne računajući prvu poziciju), doći do premještanja elemenata unutar ture, te u većini slučajeva do brisanja početnog grada iz ture. Operacija premještanja elementa može se posmatrati kao kombinacija brisanja (sa stare pozicije) i ubacivanja (na novu poziciju), (iako se može i drugačije implementirati). Obje operacije su ograničene sa *O(n)* za strukturu podataka kakva je niz. Zbog toga, je mnogo bolje koristiti strukturu podataka baziranu na sekvencijalnim referencama, kakva je recimo jednostruko povezana lista. Tada je moguće spajanje dvije komponente izvršiti u *O(1)* vremenu prespajanjem pokazivača.

Ako se ponovno posmatra prethodni slučaj posmatra kada je tura u obliku:

0, i, ..., j, 0, k, ..., l, 0 ...

, *i* najveća ušteda se ostvaruje putem iz grada *l* u grad *i*. Uz pretpostavku da je tura sačuvana u jednostruko povezanoj listi, tada se spajanje komponenti vrši sa:

- Sljedeći od *j* je sljedeći od *l*
- Prethodni od *k* je prethodni od *i*,
- Sljedeći od *l* je *i*.

I dobija se tura oblika:

0, k, ..., l, i, ..., j, 0, ...

Nakon *n - 1* koraka algoritma dobija se validna tura dužine n.

III Rezultati

Rezultati ovog greedy algoritma, sa nasumično odabranom početnom turom, u poređenju sa ranije navedenim greedy algoritmima, prikazani su u sljedećoj tabeli:

Broj gradova	**Najbliži susjed**	**Najmanja grana**	**Greedy (max. ušteda)**
5	1161	1161	1161
10	9144	9179	8680
15	10800	11273	11907
20	12589	11365	13423
50	18328	17706	31110
100	24790	22139	55756

Tabela 1 - Poređenje algoritama nad različitim skupovima gradova

Iz tabele 1 je vidljivo da algoritam daje slične rezultate kao i ostali greedy algoritmi za relativno mali skup gradova. Za veće skupove algoritam daje rješenja koja su jako daleko od optimalnog (poređenja radi 3 Opt za **100** gradova daje turu dužine **20506**, dok je optimalna tura vjerovatno ispod **20000**).

Jasno je da izbor početne ture (a samim tim i početnog grada) utiču na rezultate algoritma. U sljedećoj tabeli prikazani su rezultati **10** nasumično izabranih početnih tura na skupu od **50** gradova (istih 50 kao u tabeli 1), u poređenju sa 3 Opt algoritmom (kao kontrolnim uzorkom) nad istim početnim turama.

Početni grad	**Greedy**	**3 Opt**
7	31740	15061
45	30151	15180
43	24340	15060
12	30872	15060
23	29624	15113
8	26104	15015
2	24756	15159
32	30972	15117
36	30327	15180
25	28719	15060

Tabela 2 - Poređenje greedy i 3 Opt algoritama nad istim nasumično izabranim početnim turama

Iz tabele 2 je vidljivo da je prosječna dužina ture koju daje greedy algoritam je **28761,4**, a 3 Opt **15100,5**. Drugim riječima, na skupu od 50 gradova 3 Opt daje **1,9** puta kraću turu od greedy algoritma. Najveća varijacija rješenja (razlika između najboljeg i najgoreg) za greedy algoritam je **7400**, a za 3 Opt samo **165.** Ovo govori da su rezultati greedy algoritma direktno zavisni od izbora početne ture.

Sljedeća tabela prikazuje isto poređenje kao i tabela 1, uz razliku da sada greedy algoritam počinje sa dobrom početnom turom (konkretno počinje od ture generisane od najmanjih grana)

Broj gradova	**Najbliži susjed**	**Najmanja grana**	**Greedy (random)**	**Greedy (pocetna)**
5	5973	5267	5509	4803
10	9144	9179	8680	9242
15	10800	11273	11907	12624
20	12589	11365	13423	14381
50	18328	17706	31110	29795
100	24790	22139	55756	51364

Tabela 3 - Poređenje algoritama nad različitim skupovima gradova

Iz tabele 3 je vidljivo da dobra početna tura ne garantuje bolji rezultat. Naprotiv, nasumično odabrana početna tura nerijetko daje bolji rezultat od početne ture generisane algoritmom koji u svakom koraku bira najmanju granu grafa.

IV Zaključak

Iz ranije navedenih rezultata, može se zaključiti da algoritam daje zadovoljavajuće rezultate samo na izuzetno malim skupovima, dok sa rastom broja gradova, odstupanje od optimalnog rješenja raste mnogo brže od ostalih greedy algoritama. Već za **100** gradova algoritam daje **2** puta dužu turu od ostalih greedy algoritama. Također, treba imati u vidu da je implementacija algoritama “najmanja grana” i “najbliži susjed” jednostavnija od implementacije posmatranog algoritma. Iz navedenog, može se zaključiti da posmatrani algoritam nema praktičnu primjenu.