Hard capacitated k-facility location problem

Seminarski rad u okviru kursa
Matematičko programiranje i optimizacija
Matematički fakultet

Student:Miloš Manić 1087/2014
Problem br:14
Metode: Genetski algoritmi, iterativna lokalna pretraga i njihova hibridizacija

9. april 2015.

Sažetak

Sadržaj

1 Problem

Za Capacitated k-facility location problem(CKFL) dat je skup klijenata D i skup potencijalnih postrojenja(lokacija na kojima se može izgraditi postrojenje F[1].

a) Svako postrojenje $i \in F$ ima kapacitet s_i

b) Izgradnja postrojenja $i \in F$ košta f_i

c) Svaki klijent $j \in D$ ima potražnju d_i

d) Slanje x_{ij} jedinica robe od postrojenja i do klijenta j košta $c_{ij}x_{ij}$, gde je c_{ij} jedinicna cena proporcionalna rastojanju između i i j

e) Na svakoj potencijalnoj lokaciji $i \in F$ može se izgraditi najviše jedno postrojenje

f) Bez gubitka opštosti može se smatrati da su cene izgradnje f_i, kapaciteti s_i, i potražnje d_j celi brojevi

Cilj je opslužiti sve klijente koristeći najviše k postrojenja sa što manjim(minimalnim) troškovima izgradnje postrojenja i dopremanja robe.

1.1 Matematička formulacija problema

CKFL se može formulisat kao sledeći Mixed Integer Problem(MIP) gde promenljiva x_{ij} označava količinu potražnje klijenta j koja je opslužena postrojenjem i, a y_i označava da li je postrojenje i otvorenoi[1]:

$$min \sum_{i \in F} \sum_{j \in D} c_{ij}x_{ij} + \sum_{j \in F} f_i y_i \tag{1}$$

$$subject\ to: \quad \sum_{i \in F} x_{ij} = d_j, \forall j \in D, \tag{2}$$

$$\sum_{j \in D} x_{ij} \leq s_i y_i, \forall i \in F, \tag{3}$$

$$\sum_{i \in F} y_i \leq k, \tag{4}$$

$$x_{ij} \geq 0, \forall i \in F, \forall j \in D, \tag{5}$$

$$y \in \{0, 1\}, \forall i \in F \tag{6}$$

1.2 Primena

Kao što ime kaže ovakvi problemi se javljaju prilikom planiranja postavljanja postrojenja i prostornog planiranja.

Neke od primera primena obuhvataju[8]:

- Optimizacija rasporeda zgrada u fabričkom postrojenju
- Nalaženje rasporeda skladišta nekog prodajnog lanca
- Planiranje lokacija bitnih gradskih ustanova(bolnice, vatrogasne stanice ...)
- Planiranje lokacija baznih stanica za bežične mreže
- Raspored elemenata na čipu radi optimalne jačine signala prilikom VLSI dizajna

1.3 Postojeći načini rešavanja

Pored korišćenja egzaktnih metoda kao što su metode linearnog programiranja, problem je rešavan raznim heuristikama. Neke od najkorišćenijih pristupa su[7]:

- Lokalna pretraga i varijacije
- Gramzive heuristike
- Tabu pretraga
- Genetski algoritmi

2 Heuristike

U radu će se primeniti jedna P-heuristika, genetski algoritam, jedna S-heuristika, iterativna lokalna pretraga, i jednu hibridna, genetski algoritam poboljšan iterativnom lokalnom pretragom na generacijskom nivou.

2.1 Genetski algoritmi

Genetski algoritam je metaheuristika inspirisana procesom prirodne selekcije koji pripada klasi evolucionih algoritama. Bazira se na operatorima koji su inspirisani prirodom kao što su mutacija, ukrštanje i selekcija[5].

Genetski algoritam radi nad populacijom kandidata rešenja(jedinke, rešenja, fenotipi) koja imaju određene karakteristike koje se predstavljaju nekim kodiranjem(uobičajeno nizom nula i jedinica) koje se naziva genotip.[9] Prilagođenost(*eng. fitness*).

Opšti koraci u genetskom algoritmu su sledeći:

1. Inicijalizacija:
   Generisanje početnih rešenja(jedinki) koje će se naknadno poboljšavati kroz algoritam,
2. Mutacija:
   Menjanje jednog ili više nasumičnih vrednosti genotipa neke jedinke. Primeri:
   - bit string mutacija: bitovi genotipa se nasumično menjaju sa određenom verovatnoćom na nasumičnim mestima
   - uniformna mutacija: menja se vrednost izabrane jedinke za novu nasumično generisanu jedinku
   - Swap mutacija: razmenjuju se vrednosti 2 proizvoljne lokacije
   - Scramble: nekoliko spojenih pozicija se permutuje
3. Selekcija:
   Odabir određenih jedinki koje će učestvovati u ukrštanju. Prilagođenost, kao i u prirodi ima veliki uticaj na selekciju. Primeri:
   - Rulet selekcija: Svaka jedinka dobije verovatnoću odabira proporcionalnu njenoj prilagođenosti
   - Rangovska selekcija: Svaka jedinka dobije verovatnoću odabira proporcionalnu njenom rangu prilagođenosti
   - Stohastičko univerzalno sampliranje: Svaka jedinka je jednako verovatno odabrana
   - Turnirska selekcija: Za svaku odabranu jedinku se održava turnir u kome pobeđuje najprilagođenija jedinka

4. Ukrštanje
   Izabrane jedinke se ukrštaju tako što se kombinuju njihovi genotipi za dobijanje novih jedinki. Primeri:
   - one-point ukrštanje: genomi dve jedinke se seku u jednoj tački i razmenjuju se sadržaji genome- levi deo prve jedinke sa levim druge i desni prve sa desnim druge
   - n-point ukrštanje: genotipi se seku u n tačaka i odgovarajući delovi se uzimaju za rezultujuću jedinku naizmenično
   - uniformno ukrštanje: za svaku poziciju u reprezentaciji se nasumično bira da li će da bude uzeta od jednog ili drugog roditelja
5. Prekid:
   Proces se prekida kada je neki kriterijum prekida ispunjen. Neki od čestih kriterijuma:
   - Fiksni broj generacija je dostignut,
   - Vrednost prilagođenosti je dostigla plato, i naredne iteracije ne daju bolje rezultate,
   - Neki drugi kriterijum je dostignut(dovoljno dobro rešenje je nađeno),
   - Resursi su istrošeni(memorija, vreme izvršavanja ...),
   - Ručni prekid

   Obično se koristi neka kombinacija navedenih kriterijuma i drugih.

Koraci 2-4 se ponavljaju sve dok se ne ispuni neki kriterijum prekida.

2.1.1 Prilagođeni genetski algoritam za rešavanje CFLP

2.2 Iterativna lokalna pretraga

Iterativna lokalna pretraga(*eng. iterated local search*) je heuristika koja generiše niz rešenja generisanih unutrašnjom heuristikom(neka varijanta lokalne pretrage), čime se dobijaju kvalitetnija rešenja od prostog ponavljanja te heuristike[4]. Proces se sastoji iz dva globalna koraka:

1. Perturbacija rešenja dobijenog lokalnom pretragom
2. Lokalna pretraga počev od modifikovanog rešenja

Ovi koraci se ponavljaju sve dok se ne dostigne neki kriterijum zaustavljanja(fiksni broj iteracija, dostignut plato minimalne vrednosti ...)

Na kraju svake iteracije je izabrana jedna instanca između modifikovane i instance dobijene u prethodnoj iteraciji. Na osnovu kriterijuma prihvatanja instanci se formira putanja (*eng. walk*) do izabranog najboljeg rešenja, neki od čestih izbora su:

- Better: Izabrati uvek bolje rešenje
- Random: Izabrati uvek novo rešenje
- Restart:Generisati novo rešenje ako se rešenje nije poboljšalo u prethodnih n iteracija
- Nasumično izabrati jedno od ponuđenih rešenja iz prethodnih n iteracija.

Prva dva primera kriterijuma prihvatanja su određena samo trenutnim izborom i na njih ne utiču prethodne iteracije. Putanje koje se prave takvim kriterijumima se zovu Markovljeve putanje (*eng. Markovian walks,*

Markovian chains). Pošto poslednja dva primera uključuju u odluku prethodnih n iteracija, oni ne prave Markovljeve putanje.

Snaga i oblik perturbacije su takođe bitni i od njih zavisi uspešnost iterativne lokalne pretrage. Iterativna lokalna pretraga izbegava zadržavanje na istom lokalnom optimumu tako što primenjuje perturbacije. Snaga perturbacije treba biti dovoljna da ne možemo da se lokalnom pretragom vratimo na prethodno rešenje, dakle potrebno je izvršiti modifikaciju koja je većeg reda od lokalne pretrage. S druge strane, ako snaga perturbacije bude prevelika, iterativna lokalna pretraga se ponaša kao multistart pretraga(*eng. Multistart local search*), pa se bolja rešenja nalaze sa manjom verovatnoćom.

Neki od primera za perturbacije nad binarnom reprezentacijom rešenja:

- n-flip:Komplementiranje n nasumičnih bitova u reprezentaciji rešenja(U zavisnosti od lokalne pretrage, na primer ako je prostor lokalne pretrage n-flip, tada perturbacija mora da bude bar n+1-flip)
- n-switch: zamena vrednosti dva nasumična bita međusobno n puta
- permutacija n nasumičnih bitova
- ...

Takodje, perturbacije mogu biti adaptivne tako što se snaga perturbacije određuje na osnovu prethodnih rešenja, recimo, ako se rešenje nije poboljšalo nekoliko iteracija, povećava se snaga iteracije(recimo sa 2-flip na 3-flip).

Dakle, procedura iterativne lokalne pretrage na visokom nivou može se opisati na sledeći način:

```
1  procedure IteratedLocalSearch:
2          s_0=GenerateInitialSolution;
3          s*=LocalSearch(s_0);
4          repeat:
5                  s'=Perturbation(s*,history);
6                  s*'=LocalSearch(s');
7                  s*=AcceptanceCriterion(s*,s*',history);
8          until(termination condition met)
9  end
```

Kriterijumi zaustavljanja mogu da budu bazirani na broju iteracija, vremenu ili nekom drugom resursu, platou minimalne vrednosti, itd.

2.2.1 Prilagođena iterativna lokalna pretraga za rešavanje CFLP

2.3 Hibridizacija

Genetski algoritmi i iterativna lokalna pretraga se mogu kombinovati na nekoliko načina. Jedan od njih je korišćenje genetskog algoritma za dobijanje populacije rešenja pa korišćenje iterativne lokalne pretrage za poboljšavanje tih rešenja. Drugi način je u poboljšavati deo svake generacije iterativnom lokalnom pretragom i ubacivati poboljšane jedinke u generaciju, takva varijacija genetskog algoritma se zove memetski algoritam(*eng. Memetic algorithm*)[6]. U ovom radu su prikazani rezultati obe metode. U prvoj metodi se primenjivala iterativna lokalna pretraga na sve jedinke rezultujuće populacije genetskog algoritma. U memetskom algoritmu je zbog performansi izvršavana iterativna lokalna pretraga samo na prvih 10 jedinki, uz pamćenje vrednosti za jedinke koje se održe kroz generacije.

3 Eksperimentalni rezultati

Opisani metodi su implementirani u jeziku c# i .NET okruženju. Za rad je korišćen Visual Studio Ultimate 2013. Za dobijanje egzaktnog rešenja korišćen je *IBM ILOG CPLEX Teaching Edition 12.1*, čije je vreme izvršavanja po instanci ograničeno na 4 sata. Za testiranje je korišćen računar sa *Intel i7* procesorom na $3.1GHz$ i $8GB\,RAM$ memorije. Aplikacija je izvršavana na jednom jezgru(bez paralelizacije) zato što strukture korišćene pri implementaciji nisu bile bezbedne za paralelni rad(*thread-safe*).

Sve metode su sa svim instancama testirane po 20 puta sa različitim *random seed* vrednostima. *Random seed* vrednosti su dobijene pomoću drugog nasumičnog niza po sledećoj proceduri:

```
1  procedure testing(c):
2          Random a=new Random(c);
3          for 1 to 20 do:
4                  Random b=new Random(seed=a.next());
5                  test(random=b);
6          end;
7  end.
```

3.1 Instance

Korišćene su već postojeće instance za CFL(*capacitated facility location*) problem, uz dodat parametar k. Za skup malih instanci i za skup instanci najvećih dimezija je korišćen skup instanci koji je generisan od strane J.E. Beasley-a[2], dok je za skup srednjih instanci korišćen skup generisan od strane K. Holmberg-a, M. Ronnqvist-a i D. Yuan-a [3].

Da bi se izabralo odgovarajuće k korišćen je algoritam za izdvajanje optimalnih rešenja za različite vrednosti k čiji je pseudo kod prikazan:

```
1  for each instance
2          bestSolution=executeProblem(k=m);
3          for j=1 to m:
4                  sol=executeProblem(k=j);
5                  if(unfeasable)
6                          continue;
7                  if(sol is the optimal solution):
8                          if(sol=bestSolution):
9                                  break;
10                         log(sol);
11         end;
12 end.
```

Dalje se na osnovu upisanih rezultata(log) bira instanca sa odgovarajućom vrednošću parametra k. Pošto ima mnogo više instanci nego što je potrebno, instance sa sličnim osobinama i rezultatima se odbacuju, a ako neka instanca ispoljava žanimljive"osobine, uzima se više puta sa odgovarajućim vrednostima parametra k.

Primer zapisa jedne instance koja je korišćena za proveru ispravnosti programa je prikazan. Instanca je ručno smišljena tako da funkcija cilja bude različita za različite vrednosti parametra k.

```
n=2 m=3
k=2

d_j
2 3
```

f_i
1 2 3

s_i
1 5 7

c_i_j
1 2
3 2
3 2

3.2 Rezultati

3.3 Analiza rezultata

4 Zaključak

Literatura

[1] Karen Aardal, Pieter L van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, 242(2):358–368, 2015.

[2] John E Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65(3):383–399, 1993.

[3] Kaj Holmberg, Mikael Rönnqvist, and Di Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999.

[4] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[5] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[6] Nicholas J Radcliffe and Patrick D Surry. Formal memetic algorithms. In *AISB Workshop on Evolutionary Computing*, pages 1–16. Springer, 1994.

[7] Francisco José Ferreira Silva and DS De la Figuera. A capacitated facility location problem with constrained backlogging probabilities. *International journal of production research*, 45(21):5117–5134, 2007.

[8] Jens Vygen. *Approximation algorithms facility location problems*. Forschungsinstitut für Diskrete Mathematik, Rheinische Friedrich-Wilhelms-Universität, 2005.

[9] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.