Kapacitativni problem usmjeravanja vozila iz višebrojnih skladišta

Krešimir Baksa, Mihael Šafarić i Matija Šantl

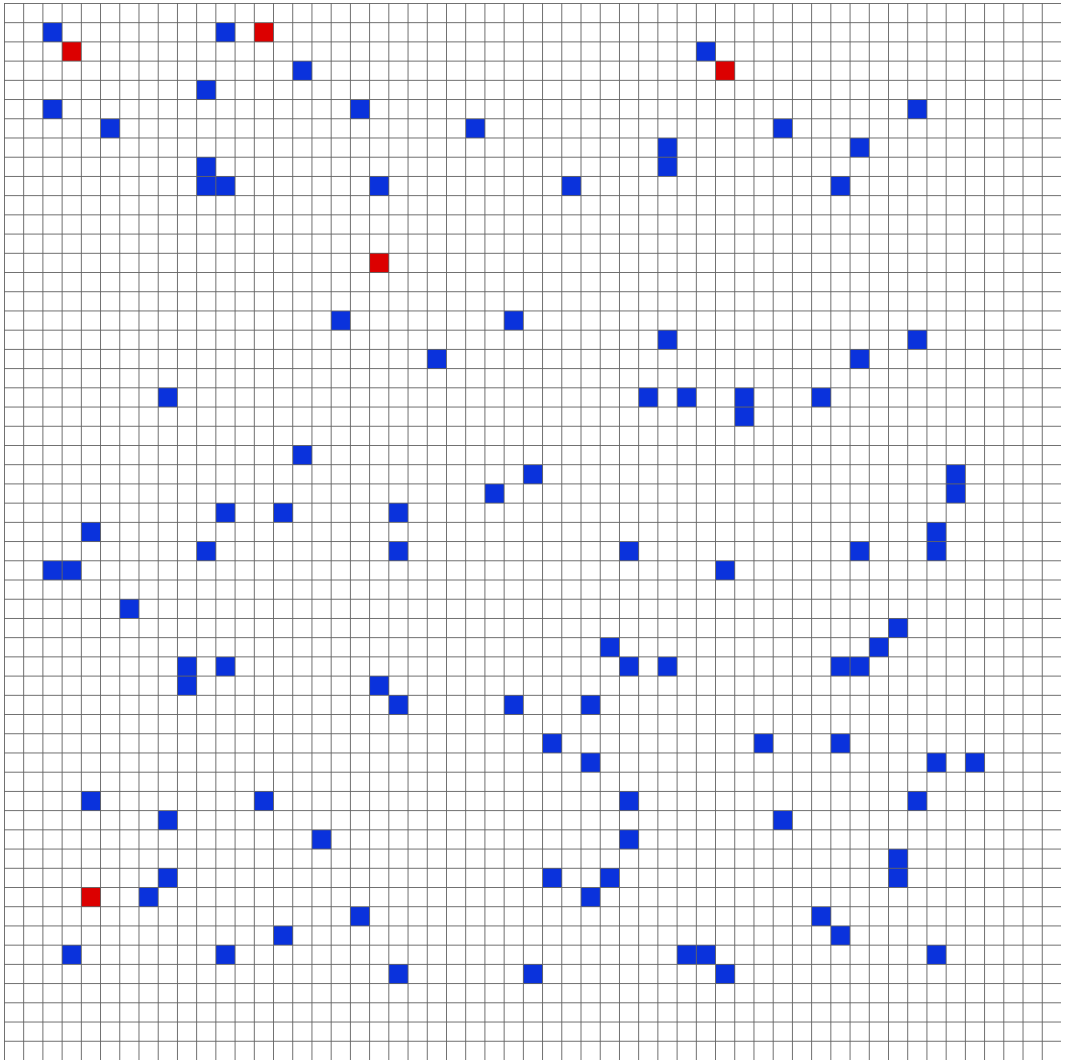
Heurističke metode optimizacije
Fakultet elektrotehnike i računarstva

Zagreb, siječanj 2015.

Definicija problema

Kapacitativni problem usmjeravanja vozila iz višebrojnih skladišta sastoji se od odabira skladišta i pronalaska Hamiltonovih ciklusa minimalne težine u težinskom grafu, poštujući kapacitativna ograničenja ciklusa na način da su svi čvorovi posluženi. Cilj je pronaći skup skladišta koje je potrebno otvoriti i ruta koje je potrebno obići kako bi se minimizirao ukupni trošak usmjeravanja paketa do korisnika (trošak otvaranja skladišta i ruta te trošak svih odabranih ruta).

Zadana instanca problema[1]



[1] crveno su označena skladišta, a plavo korisnici

Pristup rješavajnu

Ovom smo problemu odlučili pristupiti na dva načina:

- Genetskim algoritmom
- Pohlepnim algoritmom

Jedinka

Jedinka je predstavljena kao niz vektora, gdje vektor s indeksom *i* odgovoara skladištu s indeksom *i*, dok su značajke vektora *i* indeksi korisnika za kojeg je to skladište zaduženo.

Uz niz vektora, *Jedinka* također pamti svoj rezultat (*fitness*) zbog njegove složenosti izračuna.

Jedinka

```
1  class Jedinka {
2          private:
3              std::vector<User> skladiste[NUM_WAREHOUSE];
4              int fitness;
5
6          public:
7              int getFitness(void) const;
8              std::vector<User> getWarehouseUsers(const
                   int &warehouse) const;
9
10             void setFitness(const int &_fitness);
11             void setWarehouseUser(const int &warehouse,
                    const User &user);
12
13             void updateFitness(const std::vector<
                   Warehouse> &w, int capacity, int cost);
14 };
```

Početna populacija

Početna populacija je pseudo-slučajna ali se pazi da bude ispravna. Prilikom slučajnog dodijeljivanja korisnika skladištu, pazi se da se ne prekorači kapacitet skladišta. Dodatno, nasumičan je odabir umanjen uvođenjem vjerojatnosti izbora skladišta koja su bliža korisniku.

Parametar veličine početne populacije zadaje se prilikom pokretanja programa, a podrazumijevani iznos je 100.

Selekcija

Korištena je troturniska selekcija s elitizmom. Sudionici troturniske selekcije se odabiru slučajno.

Parametar eliminacije se zadaje prilikom pokretanja programa, a podrazumijevani iznos je 25.

Križanje

Za jedinke A i B, slučajno odabiremo dva indeksa X i Y te gradimo novu jedinku na način da od prve jedinke, A, uzimamo korisnike koji pripadaju skladištima iz intervala $[0, X]$ i $[Y, BROJ\_SKLADISTA]$, dok od druge jedinke, B, uzimamo korisnike koji pripadaju skladištima iz intervala $[X, Y]$.

Ako tako nastala jedinka narušava kapacitet nekog skladišta ponavljamo postupak.

Mutacija

Za slučajno odabrana skladišta X i Y, slučajnim odabirom uzmi korisnika Z iz skladišta X te ga probaj staviti u skladište Y ako time ne narušavamo kapacitet skladišta Y.

Parametar mutacije se zadaje prilikom pokretanja programa, a podrazumijevani iznos je 25%.

Fitness

Dobrotu jedinke računamo na sljedeći način:

- za svako skladište, koristeći pohlepni algoritam, dodijelimo vozila za neki skup korisnika sve dok ne zadovoljimo sve korisnike (*dist_fit* funkcija)
- za svaku rutu vozila, grubom silom izračunamo najkraći Hamiltonov ciklus (*bitonic_tour_brute*) te njegovu cijenu (*bitonic_tour_cost* funkcija)
- tako dobivene troškove pozbrajamo i spremamo u jedinku

Fitness

```
1  int res = 0;
2  for (int i = 0; i < NUM_WAREHOUSE; ++i) {
3    if (skladiste[i].size() > 0) {
4      vector<vector<User> > tours = dist_fit(skladiste[i
           ], capacity);
5      for (int j = 0; j < (int)tours.size(); ++j) {
6        vector<User> tour = bitonic_tour_brute(tours[j],
             w[i]);
7        res += cost;
8        res += bitonic_tour_cost(tour);
9       }
10
11       res += w[i].getCost();
12     }
13 }
```

Pohlepni algoritam

Pohlepni algoritam nakon učitavanja podataka razvrstava korisnike po skladištima na način da u trenutnom koraku minimizira trošak trenutnog korisnika do njemu najbližeg iz skladište u koje ga pokušava staviti uz dodatak troška povratka od tog korisnika do skladišta. Ako skladište u tom trenutku nema ni jednog korisnika, cijeni se dodaje trošak otvaranja skladišta.

To radi tako dugo dok ne razvrsta sve korisnike.

Početno rješenje

```
1  dok ima nereazvrstanih korisnika X:
2    za svako skladiste Y:
3      za svakog korisnika Z u skladistu Y:
4
5        cijena = udaljenost(X, Z) + udaljenost(X, Y)
6
7        ako je skladiste Y prazno:
8          cijena += cijena otvaranja skladista Y
9
10       ako je  cijena najmanja do sad:
11         W = Y
12
13       stavi korisnika X u skladiste W
```

Mutacija

Nakon dobivanja početnog rješenja, napravi se *POPULACIJA* broj kopija tog rješenja, te se nad svakim od njih radi mutacija koja je opisana kod genetskog algoritma.

Ukoliko je mutirano rješenje bolje od trenutnog, ono se zamjenjuje u populaciji.

Taj se postupak ponavlja *ITERACIJA* broj puta.

Mutacija

```
1  ponovi ITERACIJA broj puta:
2    za svako rjesenje R iz populacije:
3      slucajno odaberi skladiste X
4      slucajno odaberi skladiste Y
5      slucajno odaberi korisnika A iz skladista X
6
7      ako prebacivanje A iz X u Y narusava ogranicenja:
8        ponovi odabir skladista i korisinka
9      inace:
10       prebaci korisnika A iz X u Y
11       spremi izmjenjeno rjesenje
```

Lokalna pretraga

Nakon što smo dobili *POPULACIJA* broj mutiranih rješenja, nad svakim od njih se pokreće postupak lokalne pretrage.

Postupak lokalne pretrage, odabire par korisnika A i B, te zamjenjuje njihova skladišta. Ako je rješenje ne narušava kapacitet ni jednog skladišta, spremamo ga u red te nastavljamo lokalnu pretragu sa sljedećim rješenjem iz reda.

Postupak radimo tako dugo dok ima rješenja u redu ili napravimo *limit* broj iteracija.

Lokalna pretraga

```
1  dodaj rjesenje X u red Q
2
3  ponovi LIMIT broj puta ili dok ne konvergira:
4    rjesenje R = prvo rjesenje iz reda Q
5    za svaki par korisnika (A, B):
6      ako zamjenom skladista korisnika A i B u R ne
           narusavao ogranicenja:
7        R' = zamjeni skladista korisnika A i B
8
9        ako je R' bolje od X:
10         X = R'
11
12       ako je trenutno R' bolje R:
13         dodaj R' u Q
```

Fitness

Dobrotu rješenja računamo na sljedeći način:

- za svako skladište, koristeći pohlepni algoritam, dodijelimo vozila za neki skup korisnika sve dok ne zadovoljimo sve korisnike (*closest_fit* funkcija)
- za svaku rutu vozila, grubom silom izračunamo najkraći Hamiltonov ciklus (*tour*) te njegovu cijenu (*tour_cost* funkcija)
- za svaku rutu vozila dodajemo cijenu novog vozila

Fitness

```
1  fitness = 0
2  za svako skladiste X:
3    ako ima korisnika u X:
4      fitness += cijena otvaranja skladista X
5
6      R[] = closest_fit(X)
7      za svaku rutu R‘ iz R:
8        C = tour(R‘)
9        cijena += tour_cost(C)
10       cijena += cijena automobila * broj ruta
```

Usporedba rješenja genetskog i pohlepnog algoritma

Najmanji trošak dobiven genetskim algoritmom iznosi: 321140

Najmanji trošak dobiven pohlepnim algoritmom iznosi: 315983

Zaključak

Iako je genetski algoritam pristup koji bi mogao rješavati druge (i veće) instance zadanog problema, pohlepni se algoritam pokazao bolji na zadanoj instanci problema.

Uvođenje dijelova genetskog algoritma u pohlepni algoritam (mutacija) dobiveno rješenje se dodatno poboljšalo.