Multiple Traveling Salesmen Problem

Marcin Juraszek
Jakub Noga

15 grudnia 2014

1 Wstęp

1.1 Opis problemu

Problem wielu komiwojażerów (MTSP) jest odmianą szeroko znanego problemu komiwojażera (TSP). Problem charakteryzuje się większą złożonością (większa liczba rozwiązań dopuszczalnych), natomiast nie powoduje to zwiększenia trudności w jego matematycznym modelowaniu.

Istnienie problemu wielu komiwojażerów możemy zaobserwować na przykład w logistyce, kiedy to dysponując ograniczoną flotą pojazdów i kierowców (w odróżnieniu do TSP, gdzie dysponujemy dokładnie jednym) chcemy dostać się do wielu miejscowości ponosząc możliwie najniższe, zdefiniowane przez nas koszty.

1.2 Podejście do problemu

Zdecydowaliśmy, że przystąpimy do rozważań nad problemem sprowadzając go do klasycznego problemu jednego komiwojażera, stosując jedynie drobną modyfikację: miasto - *baza* może zostać osiągnięte wielokrotnie. Spowoduje to powstanie opartych na nim podcykli, które, równoważnie, można traktować jako zakończenie trasy jednego, a rozpoczęcie drugiego komiwojażera.

2 Model matematyczny

2.1 Struktura danych

Do przedstawienia danych problemu w języku posłużyliśmy się macierzami oraz, w niektórych przypadkach, wektorami.

Podstawowe dane na temat konkretnej instancji problemu tj. koszty przejść pomiędzy kolejnymi miastami przedstawiliśmy w postaci macierzy kosztów $n \times n$:

$$C = \begin{bmatrix} 0 & c_{01} & c_{02} & \dots & c_{0n} \\ c_{10} & 0 & c_{12} & \dots & c_{1n} \\ \vdots & & \ddots & & \vdots \\ c_{n0} & & \dots & & 0 \end{bmatrix}$$

Gdzie c_{ij} to koszt przejścia pomiędzy miastem i a j.

Do przechowania rozwiązań skorzystaliśmy z dwóch równoważnych postaci zapisu sekwencji przejść z miasta do miasta: macierzy przejść X oraz wektora r. Gdzie:

$$X = [X_{ij}] \in \{0, 1\}$$

oraz

$$r = [0, r_0, \dots, r_m, 0] : r_k \in \{0, \dots, n\}$$

Na przykład macierz:

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Odpowiada wektorowi:

$$r = [0, 1, 2, 3, 0, 4, 0]$$

2.2 Model

Problem wielu komiwojażerów można przedstawić jako następujący problem programowania całkowito-liczbowego:

X - macierz przejść
C - macierz kosztów
n - rozmiar problemu (ilość miast)
m - ilość komiwojażerów
c_m - koszt wysłania jednego komiwojażera

$$F(X) = \sum_{i=0}^{n} \sum_{j=0}^{n} (C_{ij} X_{ij} + m c_m) \rightarrow min \tag{2.2.1}$$

$$0 \leq \sum_{j=1}^{n} X_{0j} = \sum_{i=1}^{n} X_{i0} \leq m \tag{2.2.2}$$

$$\sum_{i=1}^{n} X_{ij} = 1 : j = 1, \ldots, n \tag{2.2.3}$$

$$\sum_{j=1}^{n} X_{ij} = 1 : i = 1, \ldots, n \tag{2.2.4}$$

3 Adaptacja algorytmu pszczelego

3.1 Przebieg pracy algorytmu

3.1.1 Scouting

Scouting - wysłanie pszczół zwiadowców, polega na wygenerowaniu zbioru losowych rozwiązań dopuszczalnych. Klasa *ScoutBee* posiada metodę:

```
public int[][] call() throws Exception {
  int [][]transitionMatrix = new int[dimensions][dimensions];
  while(!allTownsVisited()){
    int nextTown = -1; int currentTown = stepOutOfDepot();
    transitionMatrix[0][currentTown] = 1;
    visitedTowns.put(currentTown, true);
    while(!allTownsVisited()){
      nextTown = takeStep();
      if(nextTown == 0){
        break;
      } else {
        transitionMatrix[currentTown][nextTown] = 1;
        visitedTowns.put(nextTown, true);
      }
      currentTown = nextTown;
    }
    transitionMatrix[currentTown][0] = 1;
  }
  return transitionMatrix;
}
```

Metoda ta buduję losową macierz przejść w następujący sposób:

(1) Jeżeli nie odwiedzono jeszcze wszystkich miast, wylosuj, do którego wyjść z bazy.

(2) Dopóki nie odwiedzono wszystkich miast losuj kolejne miasto do momentu powrotu do bazy.

(3) Wróć do pkt. 1.

Każda pszczoła-zwiadowca zwraca jedną losową macierz przejść. Ilość pszczół-zwiadowców ustalana jest przez użytkownika.

3.1.2 Wybór sąsiedztw elitarnych

Za sąsiedztwa elitarne uważamy zbiór rozwiązań (liczność - l tego zbioru jest określana przez użytkownika), które charakteryzują się najwyższą wartością funkcji celu. Przypisanie odbywa się poprzez sortowanie malejące listy sąsiedztw zwyczajnych (losowo wybranych poprzez Scouting), a następnie wybór pierwszych l obiektów z listy.

3.1.3 Eksploracja sąsiedztw elitarnych

Na tym etapie algorytm korzysta z postaci równoważnej do macierzy przejść - X, czyli wektora przejść - r.

Eksploracja sąsiedztw elitarnych polega na modyfikacji rozwiązań na jeden z dwóch sposobów:

(a) Permutowanie wektora przejść - losowa zmiana kolejności na n kolejnych pozycjach wektora, liczba n zmienia się proporcjonalnie do liczby pozostałych iteracji w zakresie podanym przez użytkownika.

(b) Losową zmianę pozycji m elementów wektora przejść poprzez kolejno usunięcie i losowe umieszczenie każdego z nich.

Każde z sąsiedztw elitarnych eksplorowane jest przez w pszczół-robotnic - liczba w ustalana jest przez użytkownika.

3.1.4 Iteracje

Ilość iteracji ustalana jest przez użytkownika. Podczas każdej itaracji powstaje $l \cdot w$ sąsiedztw dlatego wysoce niewydajnym, pod względem pamięci operacyjnej, jest przechowywanie wszystkich wyników. Z tego powodu, co ustaloną liczbę iteracji i następuje ponowny wybór sąsiedztw elitarnych spośród sąsiedztw powstałych z eksploracji. Po tej operacji algorytm kontynuuje standardową eksplorację.

3.1.5 Zbiór parametrów algorytmu

- s - liczba pszczół-zwiadowców
- l - liczba sąsiedztw elitarnych
- w - liczba pszczół-robotnic przypadająca na jedno sąsiedztwo elitarne
- n - ilość elemetów wektora przejść, które będą permutowane (wielkość sąsiedztwa)
- m - ilość elemetów wektora przejść, którym pozycja zostanie zmieniona (wielkość sąsiedztwa)
- i - liczba iteracji, po których sąsiedztwa elitarne zostaną przypisane na now
- t - ilość iteracji algorytmu