GIS Projekt

Problem komiwojażera w obecności ulic jednokierunkowych.

Tomasz Korzeniowski, 265753
Jacek Sochacki, 259741

15 grudnia 2017

1 Zadanie

Należy zaimplementować (nietrywialny) algorytm rozwiązujący problem komiwojażera, a następnie zbadać, czy można modelować ulice jednokierunkowe w ten sposób, że połączeniu w kierunku A→B (A, B - wybrane miasta) nadaje się wagę równą odległości miedzy miastami, a połączeniu B→A nadaje się wagę o bardzo dużej wartości. Głównym celem zadania jest identyfikacja warunków, w których algorytm wskazuje drogę „pod prąd" pomimo istnienia drogi „legalnej".

2 Opis zadania

2.1 Problem komiwojażera

Problem komiwojażera można przedstawić następująco. Rozważmy graf $G = (V, E)$, gdzie V to zbiór n miast, które mają zostać odwiedzone przez komiwojażera, natomiast E to zbiór możliwych połączeń między tymi miastami. W ogólności rozważamy graf pełny. Potrzebujemy wyznaczyć macierz kosztów przejść między miastami, czyli wagi krawędzi c_{ij} między miastem i oraz j. Do zaznaczenia, którymi krawędziami przejedzie komiwojażer wykorzystamy zmienne binarne x_{ij}

$$x_{ij} = \begin{cases} 1, & \text{gdy komiwojażer przejeżdża z miasta i do j} \\ 0, & \text{w przeciwnym przypaku} \end{cases}$$

Wiadomo, że każde miasto musi być odwiedzone dokładnie raz, z wyjątkiem miasta startowego, w którym komiwojażer zakończy swoją podróż. Sprowadza się to do znalezienia drogi w grafie G, której pierwszy i ostatni wierzchołek są tożsame, a koszt przejazdu wyznaczoną drogą jest jak najmniejszy.

Powyższe zadanie można opisać następującym modelem

$$\min \sum_{i,j \in V} c_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in V} x_{ij} = 1, \quad i \in V \backslash \{j\} \tag{2}$$

$$\sum_{i \in V} x_{ij} = 1, \quad j \in V \backslash \{i\} \tag{3}$$

$$x_{ij} \in \{0, 1\} \tag{4}$$

$$x \text{ jest drogą w grafie G} \tag{5}$$

Zauważmy, że w macierzy $X = [x_{ij}]_{n\times n}$ w każdej kolumnie i każdym wierszu występuje dokładnie jedna jedynka. Zatem można reprezentować znaleziony cykl jako wektor kolejnych wierzchołków odwiedzanych przez komiwojażera. Wiemy, że każde miasto występuje w tym wektorze tylko raz, a cykl zamykany jest przez skrajne wierzchołki wektora.

Tak przedstawione zadanie rozwiązuje problem komiwojażera dla grafu pełnego. Zadanie jednak nie jest jedynie problemem komiwojażera. Należy rozważyć połączenia jednokierunkowe między dowolnymi miastami A i B, zatem musimy rozważać graf skierowany. Struktura problemu i przyjęty model nie musi być zmieniany. Gwarantuje nam to odpowiednia konstrukcja macierzy kosztów $C = [c_{ij}]_{n\times n}$. W przeciwieństwie do grafu nieskierowanego nie będzie ona symetryczna. Zatem otrzymujemy graf skierowany gdzie ulice jednokierunkowe będą reprezentowane przez krawędzie o nieskończonym koszcie.

W ramach zadania sprawdzimy czy powyższe modelowanie ulic jednokierunkowych jest wystarczające do znalezienia dopuszczalnego rozwiązania problemu komiwojażera. Spróbujemy wskazać warunki w jakich algorytm znajduje błędne rozwiązania (ulice „pod prąd").

2.2 Przeszukiwanie tabu

Wybranym sposobem rozwiązania problemu będzie algorytm przeszukiwania tabu. Jest to metaheurystyka stosowana do rozwiązywania problemów optymalizacyjnych. Polega na wykonaniu ciągu kolejnych ruchów w celu przeszukiwania przestrzeni rozwiązań dopuszczalnych zadania.

Przeszukiwanie tabu rozpoczyna się wybraniem rozwiązania początkowego. W każdej iteracji znajdowane jest rozwiązanie lokalnie najlepsze, a także poszukiwani są sąsiedzi (w zbiorze rozwiązań dopuszczalnych) takiego rozwiązania. Najlepszy ze znalezionych sąsiadów staje się nowym rozwiązaniem lokalnym dla kolejnej iteracji. Najlepszy z sąsiadów charakteryzuje się mniejszą wartością funkcji celu (1) od obecnego rozwiązania.

Na początku działania algorytmu potrzebne jest pewne wstępne rozwiązanie. W naszym przypadku wystarczy przyjąć dowolny ciąg wierzchołków (miast). Rozważamy grafy pełne, więc wybór ulicy jednokierunkowej nie jest przeszkodą. Krawędzie które „nie istnieją" (nie ma połączenia między danymi miastami) lub są to ulice jednokierunkowe mają po prostu koszt nieskończony. Ze względu na trudności w implementacji nieskończoności ograniczymy się do dostatecznie dużej wartości takiej krawędzi.

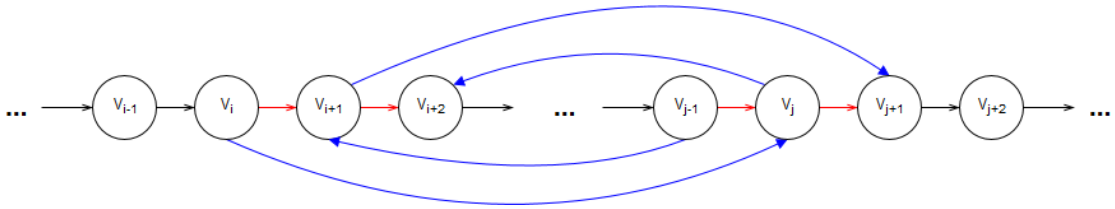
Sąsiadem rozwiązania v nazywamy ciąg wierzchołków v', który różni się od oryginalnego na dokładnie dwóch pozycjach. Poszukiwanie sąsiada obecnego rozwiązania wykorzystuje technikę „2-opt move". Polega ona na zamianie dwóch wierzchołków w poszukiwanej drodze, które nie sąsiadują. Zmiana ta powoduje usunięcie dwóch krawędzi z oryginalnej drogi, zamianie wierzchołków i ponowne połączenie przestawionych wierzchołków w drogę. Dla przykładu rozważmy drogę o następujących wierzchołkach v

Przed zmianą:	$\cdots$	v_{i-1}	v_i	$\mathbf{v_{i+1}}$	v_{i+2}	$\cdots$	v_{j-1}	$\mathbf{v_j}$	v_{j+1}	v_{i+2}	$\cdots$
Po zmianie:	$\cdots$	v_{i-1}	v_i	$\mathbf{v_j}$	v_{i+2}	$\cdots$	v_{j-1}	$\mathbf{v_{i+1}}$	v_{j+1}	v_{i+2}	$\cdots$

Zostały usunięte krawędzie między wierzchołkami (v_i, v_{i+1}) oraz (v_j, v_{j+1}), lecz powstały dwie nowe krawędzie (v_i, v_j) oraz (v_{i+1}, v_{j+1}). Dla reszty krawędzi pozostała droga przed zmianą, a dla niektórych z nich zmienił się kierunek ich przechodzenia w nowo wyznaczonej drodze.

Ze względu na skierowanie grafu naszego zadania, technika „2-opt move" zostanie zachowana, lecz zamiast dwóch krawędzi, zostaną zmienione cztery. Przyjmując zamianę wierzchołków jak w powyższym przykładzie, zostaną usunięte krawędzie (v_i, v_{i+1}), (v_j, v_{j+1}) jak poprzednio oraz dodatkowo (v_{i+1}, v_{i+2}) i (v_{j-1}, v_j). Zamiast nich powstaną krawędzie (v_i, v_j), (v_{i+1}, v_{j+1}) jak wcześniej oraz nowe (v_j, v_{j+2}) oraz (v_{j-1}, v_{i+1}). Graficznie zamianę tę przedstawia wykres 1.

Taki sposób zamiany niesąsiadujących wierzchołków umożliwia znalezienie wszystkich sąsiadów obecnego rozwiązania. Wystarczy, że sprawdzimy każdą parę wierzchołków, nawet tych sąsiadujących (graf skierowany). Sąsiad o najmniejszej wartości funkcji celu stanie się



Wykres 1: Zamiana wierzchołków v_{i+1} oraz v_j zgodnie w 2-opt move dla grafu skierowanego.

wtedy bieżącym rozwiązaniem. Do obliczenia wartości funkcji celu wystarczy przejść nowo wyznaczonym cyklem, sumując wagi kolejnych krawędzi.

Przeszukiwanie tabu bierze swoją nazwę z ruchów tabu. Jest to sposób zaznaczania, które wierzchołki zostały zamienione. Na listę tabu wpisujemy liczbę iteracji algorytmu, przez którą ponowna zamiana tychże wierzchołków staje się zakazana (tabu). Trudno określić jak długo elementy listy tabu powinny być zakazane. Autorzy [1] sugerują by czas pobytu na liście tabu był równy $3n$, gdzie n oznacza liczbę miast problemu.

Może się zdarzyć, że jeden z rozważanych sąsiadów objętych tabu daje świetny wynik, lepszy od jakiegokolwiek rozważanego wcześniej rozwiązania. W standardowym podejściu, algorytm przeszukiwania, wybrałby najlepsze z rozwiązań, w których nie ma ruchów zakazanych. Takie podejście powodowałoby, że algorytm jest mało elastyczny. Jeśli sytuacja jest nietypowa (znaleziono najlepsze ze wszystkich dotąd znanych rozwiązań) możemy zapomnieć o zakazie i wybierzemy nietypowe rozwiązanie.

Schemat działania przeszukiwania tabu przestawia algorytm 1.

Algorytm 1 Przeszukiwanie tabu

```
Require: C
  tries ← 0
  while tries ≠ MAX-TRIES do
      wygeneruj drogę początkową v
      count ← 0
      iter ← 0
      while iter ≠ MAX-ITER do
          znajdź sąsiadów v i wybierz najlepszego z nich
          zamień odpowiednie wierzchołki
          uaktualnij listę tabu
          if nowa droga jest lepsza przy danej wartości tries then
              uaktualnij informację o lokalnej najlepszej drodze
              count ← 0
              if bieżąca droga jest najlepsza dla wszystkich tries then
                  uaktualnij informację o globalnej najlepszej drodze
              end if
          else
              ++count
          end if
          if count = MAX-COUNT then
              zbyt długo nie znaleziono lepszego rozwiązania - przerwij tę próbę
          end if
          ++iter
      end while
      ++tries
  end while
```

Licznik *tries* odpowiada za liczbę prób poszukiwania rozwiązania. W ramach każdej próby zostanie wykonanych *iter* poszukiwań sąsiada i aktualizacji listy tabu. Jeśli jednak przez *count* iteracji algorytmu rozwiązanie nie poprawi się względem dotychczas najlepszego znalezionego, nie ma sensu dalej poszukiwać. Może to wynikać z osiągnięcia minimum lokalnego w przestrzeni rozwiązań dopuszczalnych.

3 Implementacja algorytmu

3.1 Struktury danych

Do wyznaczenia rozwiązania problemu komiwojażera będziemy potrzebowali

$C = [c_{ij}]_{n \times n}$	Macierz kosztów przejść między n miastami. Zakładamy, że graf jest pełny, a ulice kierunkowe przyjmują wartość wystarczająco dużą, żeby uznać ją za nieskończoność (np. 1e38). Reprezentacją takiej macierzy będzie tablica dwuwymiarowa (tablica tablic) typu zmiennoprzecinkowego podwójnej precyzji (double).
v	Wektor reprezentujący drogę komiwojażera. Zawiera indeksy kolejnych odwiedzonych miast, zatem będzie typu całkowitoliczbowego.
$LT = [lt_{ij}]_{n \times n}$	Lista tabu - macierz zakazanych przejść. Element lt_{ij} przechowuje informację o liczbie iteracji przez które zamiana miast i oraz j jest zakazana. Typ całkowitoliczbowy.

Ponadto będą potrzebne iteratory pętli czy zmienna przechowująca wartość funkcji celu.

3.2 Projekty testów

W ramach testów należy sprawdzić poprawność działania algorytmu. W tym celu zostaną wykorzystane dane (macierz kosztów przejść C) dla których znana jest wartość najtańszej drogi komiwojażera. Liczność miast w tych danych będzie rzędu kilkunastu.

Po potwierdzeniu poprawności wyznaczanego rozwiązania dane te zostaną uzupełnione o ulice jednokierunkowe tak, by wartości najtańszych dróg pozostały niezmienne. Algorytm powinien wyznaczać takie same rozwiązania jak w poprzednich testach.

Następnym krokiem będzie wygenerowanie macierzy kosztów C wraz z losowymi ulicami jednokierunkowymi. Wagi krawędzi zostaną dostosowane tak, by zapewnić istnienie cyklu w grafie. Liczba ulic jednokierunkowych będzie się zmieniać tak samo jak liczba miast dla których będą generowane dane. Pozwoli to zbadać zadania dużej skali (ok. 1000 wierzchołków). Wprowadzana liczba ulic jednokierunkowych będzie pewnym przyjętym procentem wszystkich ulic istniejących w grafie.

Liczba testów każdego rodzaju będzie rzędu kilkunastu dla każdego z przyjętych zestawów danych jak i liczby ulic jednokierunkowych.

3.3 Założenia programu

Implementacja przeszukiwania tabu zostanie wykonana w języku C++. Program będzie wyświetlał na standardowym wyjściu znaleziony cykl komiwojażera oraz jego koszt. Dane o problemie do rozwiązania będą pobierane z pliku tekstowego. W pierwszej linii tego pliku zapisana będzie liczba miast, a w kolejnych koszty przejścia między odpowiednimi nimi, oddzielone spacjami.

Literatura

[1] Z. Michalewicz, D. B. Fogel *Jak to rozwiązać czyli nowoczesna heurystyka*. WNT, Warszawa 2006