GIS Projekt

Problem komiwojażera w obecności ulic jednokierunkowych.

Tomasz Korzeniowski, 265753
Jacek Sochacki, 259741

11 stycznia 2018

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 w kolejnych wierzchołków odwiedzanych przez komiwojażera. Wiemy, że każde miasto występuje w tym wektorze tylko raz, a cykl zamykany jest przez skrajne wierzchołki wektora. Tak utworzony cykl zawsze będzie cyklem Hamiltona, jeśli operacjami wykonywanymi na jego elementach będzie zamiana ich kolejności.

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. Ze względu na trudności w implementacji nieskończoności ograniczymy się do dostatecznie dużej wartości takiej krawędzi.

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. Sąsiadem rozwiązania w nazywamy ciąg wierzchołków w', który różni się od oryginalnego na dokładnie dwóch pozycjach. Najlepszy ze wszystkich znalezionych sąsiadów staje się nowym rozwiązaniem lokalnym dla kolejnej iteracji. Najlepszy z sąsiadów charakteryzuje się mniejszą wartością funkcji celu (1) od obecnego rozwiązania.

Wybrany algorytm 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). Z każdą iteracją algorytmu czas, przez który wcześniej zazanczony ruch jest zakazany, zostaje zmniejszony o 1. Lista tabu jest listą tylko z nazwy. W rzeczywistości będziemy operowali na macierzy górnej trójkątnej $LT = [lt_{ij}]_{n\times n}$. Element lt_{ij} będzie zawierał informację o liczbie iteracji, przez które zamiana wierzchołków i oraz j jest zakazana. Gdy wartość $lt_{ij} = 0$, dany ruch przestanie być zakazany.

W implementacji, powyższa dekrementacja listy będzie mało wydajna. Z tego względu będziemy na liście tabu zapisywać aktualny numer iteracji algorytmu powiększony o stałą liczbę iteracji przez które dany ruch jest tabu. Do sprawdzenia czy dany ruch jest zakazany, będziemy porównywać wartość listy tabu tego ruchu z akrulnym licznikiem iteracji. Gdy wartość listy tabu będzie co najmniej taka jak aktualy licznik iteracji, ruch przestanie być zakazany. 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.

Na początku działania algorytmu potrzebne jest pewne wstępne rozwiązanie. W naszym przypadku będzie to różnowartościowy ciąg, zawierający wszystkie liczby naturalne ze zbioru $[1, n]$. 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 bardzo duży koszt.

2.3 2-opt move

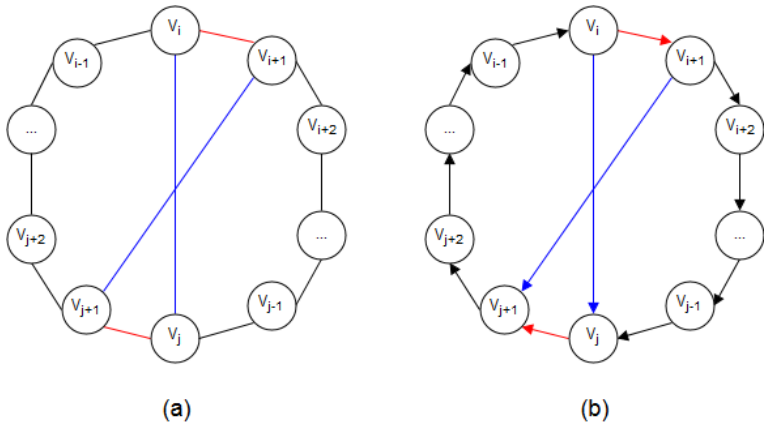
Poszukiwanie sąsiada w' obecnego rozwiązania wykorzystuje technikę „2-opt move", Metoda ta została przewidziana dla grafów nieskierowanych. Polega na zamianie dwóch wierzchołków w poszukiwanej drodze, które nie sąsiadują. Zmiana ta powoduje usunięcie dwóch krawędzi z oryginalnej drogi, zamianie wierzchołków i ponowne połączenie przestawionych wierzchołków w drogę. Dla przykładu rozważmy drogę w o następujących wierzchołkach v

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

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

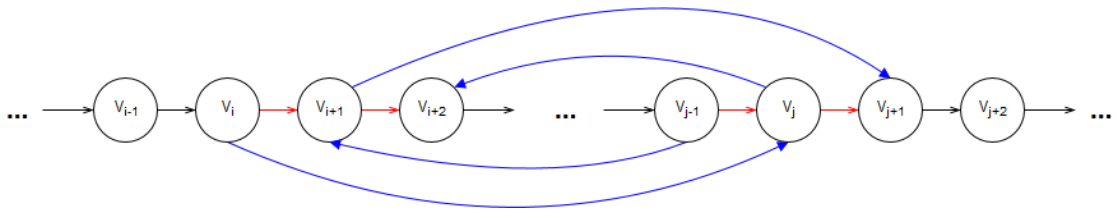
Zostały usunięte krawędzie między wierzchołkami (v_i, v_{i+1}) oraz (v_j, v_{j+1}), lecz powstały dwie nowe krawędzie (v_i, v_j) oraz (v_{i+1}, v_{j+1}). Dla reszty krawędzi pozostała droga przed zmianą, a dla niektórych z nich zmienił się kierunek ich przechodzenia w nowo wyznaczonej drodze. Zamianę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

Ze względu na skierowanie grafu naszego zadania, technika „2-opt move" zostanie zachowana, lecz zamiast dwóch krawędzi, zostaną zmienione cztery. Wynika to z wykresu 1b. Po dodaniu skierowania cyklu, widać, że nie da się dodać dwóch nowych krawędzi tak, aby pozostałe krawędzie można było przejść zgodnie z ich kierunkiem przed zmianą.



Wykres 1: Zamiana wierzchołków v_{i+1} oraz v_j zgodnie z 2-opt move dla grafu nieskierowanego (a) oraz skierowanego (b).

Przyjmując zamianę wierzchołków jak w przykładzie grafu nieskierowanego, 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 2.



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

Taki sposób zamiany niesąsiadujących wierzchołków umożliwia znalezienie wszystkich sąsiadów obecnego rozwiązania. Wystarczy, że sprawdzimy każdą parę wierzchołków, nawet tych sąsiadujących. Sąsiad o najmniejszej wartości funkcji celu stanie się wtedy bieżącym rozwiązaniem. Do obliczenia wartości funkcji celu wystarczy przejść nowo wyznaczonym cyklem, sumując wagi kolejnych krawędzi.

Ponadto, w celu skrócenia czasu wykonania algorytmu wprowadzimy licznik *count*. Będzie on odpowiedzialny za zliczanie iteracji, w których znaleziony sąsiad nie daje lepszego rozwiązania od aktualnie znanego.

2.4 Kryterium aspiracji

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.

2.5 Schemat algorytmu

Schemat działania przeszukiwania tabu przestawia algorytm 1. Licznik *tries* odpowiada za liczbę prób poszukiwania rozwiązania. W ramach każdej próby zostanie wykonanych *iter* poszukiwań sąsiada i aktualizacji listy tabu. Najlepsze rozwiązanie w danej próbie (lokalne) jest przechowywane w *bestSolverScore*. Jeśli to rozwiązanie jest najlepsze z dotychczas znalezionych dla wszystkich prób (globalne) zostanie zapamiętane w *bestSolutionScore*. 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).
w	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. Początkowo będzie dany graf pełny, dla którego znany jest koszt drogi optymalnej. Liczba wierzchołków grafu będzie rzędu kilkunastu. Program kilkukrotnie poszuka rozwiązania, a następnie najlepsze z nich zostanie zapamiętane i porównane z rozwiązaniem optymalnym.

Drugim krokiem będzie zakazanie ruchu w określonych połączeniach, które nie należały do najlepszego rozwiązania. Koszt niedozwolonych połączeń będzie zwiększony o umowną wartość nieskończoności - 1e38. Tak dobrana wartość może powodować utratę precyzji, lecz nie jest to problem, gdyż interesuje nas liczba ulic jednokierunkowych w znalezionym rozwiązaniu a nie

Algorytm 1 Przeszukiwanie tabu

```
Require: C
  tries ← 0
  while tries ≠ MAX-TRIES do
      wygeneruj drogę początkową w
      count ← 0
      iter ← 0
      while iter ≠ MAX-ITER do
          while istnieje nieodwiedzony sąsiad w do
              if w′ ma mniejszą wartość funkcji celu od innych sąsiadów nieobjętych tabu LUB
  w′ spełnia kryterium aspiracji then
                  wybierz najlepszego sąsiada: w ← w′
              end if
          end while
          uaktualnij listę tabu: lt_ij ← iter + 3n
          if w′ jest lepszy od w dla danego tries then
              bestSolverScore ← w
              count ← 0
              if w jest najlepszy dla wszystkich tries then
                  bestSolutionScore ← w
              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
```

dokładna wartość funkcji celu rozwiązania. Testy te zostaną poprowadzone dla kilku określonych procentów połączeń jednokierunkowych.

Następnie zostaną wygenerowanie losowe macierze kosztów C wraz z zadanym procentem ulic jednokierunkowych. Wagi krawędzi zostaną dostosowane tak, by zapewnić istnienie cyklu Hamiltona w grafie. Pozwoli to zbadać zadania dużej skali (ok. 1000 wierzchołków).

Wszystkie te testy zostaną poprowadzone dla różnej liczby miast. Wyniki zostaną przedstawione w postaci tabeli. Dla określonej liczby miast i procentu niedozwolonych połączeń zostanie przedstawiony procent prawidłowych rozwiązań, a dla grafów pełnych podany zostanie również stosunek rozwiązania optymalnego do najlepszego odnalezionego przez algorytm.

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.

4 Wyniki

Dla każdego z przyjętych procentów ulic jednokierunkowych oraz liczby miast wykonano 10 testów. Testy dla mniejszych grafów (10-500 wierzchołków) wykonywały 5 prób poszukiwania rozwiązania ($tries = 5$). Testy na dużych grafach umożliwiały jedynie jedną próbę. Maksymalna liczba iteracji algorytmu, którą algorytm mógł wykonać to MAX-ITER = 10000. Jeżeli przez MAX-COUNT = 200 iteracji wynik nie uległ poprawie, obliczenia zostawały przerywane i rozpoczynała się kolejna próba testu.

Za poprawne wykonanie algorytmu uznawane jest znalezienie cyklu Hamiltona, w którym nie występuje żadna ulica jednokierunkowa. Procentowy udział sukcesów w wykonanych testach przedstawia tabela 1. Wartości „–" oznaczają, że nie da się uzyskać danych spełniających warunki na istnienie cyklu oraz zadanego procenta ulic jednokierunkowych.

Można zaobserwować, że jeśli ulice jednokierunkowe stanowią co najwyżej 30% wszystkich dostępnych ulic, algorytm przeszukiwania tabu zawsze znajduje poprawne rozwiązanie. Ponadto, wraz ze wzrostem liczby wierzchołków granica procentowego udziału ulic jednokierunkowych wzrasta. Dopiero po przekroczeniu 90% algorytm nie znajduje poprawnego rozwiązania.

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	100	100	100	30	20	–	–
25	100	100	100	90	0	0	–
50	100	100	100	100	20	0	0
100	100	100	100	100	70	0	0
250	100	100	100	100	90	0	0
500	100	100	100	100			
1000	100	100	100	100	100	80	0

Tabela 1: Procent sukcesów.

W tabeli 2 zostały przedstawione średnie liczby iteracji algorytmu. Dla bardzo małych grafów (10-25 wierzchołków) liczba iteracji jest niemal stała. Dla większych grafów wraz ze wzrostem liczby ulic zakazanych, liczba iteracji maleje. Uzasadnieniem takiej obserwacji jest fakt, że każda zamiana wierzchołków w cyklu może zmniejszyć jego koszt przez usuwanie krawędzi, lecz jednocześnie dodawane krawędzie drastycznie go podnoszą. W takim przypadku, rozważany sąsiad nie zostanie zapamiętany. Jeśli sytuacja powtórzy się wielokrotnie, doprowadzi to do zwiększenia licznika *count* i wcześniejszego zakończenia poszukiwania. Powyższą sytuację najlepiej widać w przypadku grafów o 500 i 1000 wierzchołków.

Ponadto czas zakazu lt_{ij} długości $3n$ jest słuszny dla małych grafów. W przypadku dużych grafów wydaje się zbyt długi. Dla grafu o 1000 wierzchołków ruch tabu będzie zakazany przez 3000 iteracji. Szukany cykl liczy 1000 krawędzi, czyli po 1000 iteracji wiele z możliwych ruchów jest zakazanych przez około 2000 kolejnych iteracji. Tak długi czas oczekiwania na nowego sąsiada może zostać złamany jedynie przez kryterium aspiracji, które to zachodzi dość rzadko. Algorytm byłby mało wydajny, gdyby pozwolić mu tak długo czekać na kolejne rozwiązanie, więc licznik braku poprawy rozwiązania *count* przerywa poszukiwania.

Do wykonania testów wykorzystano komputer z systemem operacyjnym Windows 7, procesorze Intel Core i5-3210M 2.5 GHz oraz pamięci operacyjnej 8 GB.

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	210	208	207	205	205	–	–
25	236	234	229	223	214	212	–
50	303	296	280	255	233	233	230
100	436	432	203	352	283	273	268
250	954	927	804	693	511	386	394
500							
1000	4424	4183	3577	2883	2095	1441	1040

Tabela 2: Średnia liczba iteracji.

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	0,005	0,004	0,003	0,002	0,002	–	–
25	0,046	0,05	0,044	0,082	0,04	0,04	–
50	0,5	0,48	0,44	0,38	0,35	0,37	0,37
100	7,87	8,54	7,06	5,94	4,67	4,2	4,26
250	38	35,9	31,8	28,4	21	15,2	14,2
500							
1000	8,8h	8h	6,1h	5,5h	4,3h	2,7h	2h

Tabela 3: Średni czas wykonania algorytmu.

Literatura

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

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	0,003	0	0	0	0,001	–	–
25	0,006	0,004	0,007	0,12	0,006	0,006	–
50	0,035	0,066	0,053	0,032	0,036	0,076	0,086
100							
250							
500							
1000	1,5h	1,3h	1,3h	0,9h	0,6h	0,4h	4min

Tabela 4: Odchylenie standardowe czasu wykonania algorytmu.

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	115,7	159	162,2	81	10	–	–
25	256	290	396,9	538,73	3e+38	7e+38	–
50	422,3	461,6	629,3	1057,4	1883	7e+38	1,3e+39
100	751,6	797,25	1052	1644,3	1,4e+37	5e+38	1,9e+39
250	1486,8	1587	2103	3118	5664,6	3,6e+38	2,21e+39
500							
1000	3959,3	4221,2	5760,1	8291,9	14231	24660	2,21e+39

Tabela 5: Średnia wartość funkcji celu.

	Procent ulic jednokierunkowych						
Liczba miast	0	10	30	50	70	80	90
10	42,31	64,46	84,03	84,75	0	–	–
25	44,61	58,9	81,1	142,8	1,3e+38	1,3e+38	–
50	52,07	68,2	55,2	149,7	157,07	3,2e+38	1,2e+38
100	75,27	60,02	81,9	166,5	3,8e+37	2,5e+38	2,3e+38
250	85	99,3	146,1	145,6	277,5	1,6e+38	5,6e+38
500							
1000	60,22	88,2	182,7	219,3	443,7	727,6	3e+38

Tabela 6: Odchylenie standardowe wartości funkcji celu.