Projektowanie efektywnych algorytmów

Zadanie PR1: Implementacja i analiza efektywności algorytmu podziału i ograniczeń oraz programowania dynamicznego .

Jakub Oszust
241392

8 grudnia 2019

Godzina zajęć: Środa, 19:05

Prowadzący: dr. inż. Dariusz Banasiak

Spis treści

1 Wstęp teoretyczny

1.1 Opis problemu

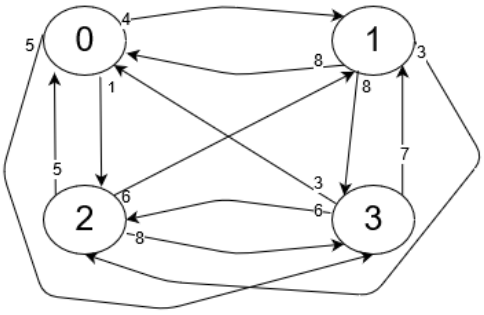
Problem komiwojażera jest jednym z najbardziej popularnych problemów optymalizacyjnych. Stawia on następujące pytanie "Mając listę miast oraz odległości pomiędzy nimi, jaka jest najkrótsza możliwa ścieżka, która odwiedza każde miasto i wraca do miasta, od którego zaczęła się ścieżka?". Problem ten należy do klasy NP- trudnej, co oznacza, że nie znaleziono rozwiązania problemu z wielomianową złożonością obliczeniową. W wersji asymetrycznej problemu dla dwóch dowolnych miast A i B, ścieżka od A do B może być inna niż ścieżka z B do A.

1.2 Opis algorytmu i oszacowanie złożoności obliczeniowej

W projekcie zaimplementowano trzy metody rozwiązania tego problemu- przegląd zupełny, algorytm podziału i ograniczeń oraz programowanie dynamiczne. Złożoność obliczeniowa przeglądu zupełnego wynosi O(n!), algorytm ten polega na sprawdzeniu wszystkich możliwych ścieżek a następnie wybraniu najlepszej. Algorytm podziału i ograniczeń to popularna metoda wykorzystywana w wielu problemach optymalizacyjnych. Polega ona na odrzucaniu ścieżek, których koszta są za duże, na podstawie obliczeń granic ścieżki. Złożoność obliczeniowa jest zależna od danych. W pesymistycznym wypadku, gdy nie odrzucimy żadnej ze ścieżek będzie ona wynosiła O(n!). W programowaniu dynamicznym koszta przejścia ścieżek obliczane są od korzenia lub od liści, drzewa stworzonego ze wszystkich możliwych ścieżek. Przy wchodzeniu na wyższe poziomy drzewa, obliczane są koszty ścieżek rozgałęzień i wybierana jest najmniejsza z nich. Złożoność obliczeniowa algorytmów dla problemu komiwojażera stworzonych tą metodą różni się w zależności od implementacji około $O(2^n * n^2)$.

2 Przykład praktyczny

W przykładach posłużymy grafem napisanym poniżej, na podstawie którego stworzono macierz sąsiedztwa.'X' symbolizuje nieskończoność. Wszystkie pętle w grafie, już w trakcie wczytywania macierzy do programu oznaczam jako nieskończoność.



Graf

$$\begin{bmatrix} 'X' & 4 & 1 & 5 \\ 8 & 'X' & 3 & 8 \\ 5 & 6 & 'X' & 8 \\ 3 & 7 & 6 & 'X' \end{bmatrix}$$

Macierz sąsiedztwa

2.1 Metoda podziału i ograniczeń

Algorytm rozpocznijmy od obliczenia ograniczenia dla korzenia drzewa - wierzchołek 0.
Rozpoczynamy od redukcji wierszy macierzy, przez wybranie najmniejszej wartości z wiersza i odjęcie jej od wszystkich komórek w wierszu.

$$\begin{bmatrix} 'X' & 4 & 1 & 5 \\ 8 & 'X' & 3 & 8 \\ 5 & 6 & 'X' & 8 \\ 3 & 7 & 6 & 'X' \end{bmatrix}$$

$$\begin{bmatrix} 'X' & 3 & 0 & 4 \\ 5 & 'X' & 0 & 5 \\ 0 & 1 & 'X' & 3 \\ 0 & 4 & 3 & 'X' \end{bmatrix}$$

Analogicznie redukujemy kolumny:

$$\begin{bmatrix} 'X' & 3 & 0 & 4 \\ 5 & 'X' & 0 & 5 \\ 0 & 1 & 'X' & 3 \\ 0 & 4 & 3 & 'X' \end{bmatrix}$$

$$\begin{bmatrix} 'X' & 2 & 0 & 1 \\ 5 & 'X' & 0 & 2 \\ 0 & 0 & 'X' & 0 \\ 0 & 3 & 3 & 'X' \end{bmatrix}$$

Obliczamy koszt ograniczenia poprzez zsumowanie wartości, o których redukowano wiersze oraz kolumny, czyli: Koszt = [1 3 5 3] + [0 1 0 3] = 12 + 4 = 16
Następnie będziemy sprawdzać które rozwinięcie drzewa będzie miało najniższy koszt, możliwe rozwinięcia: 0->1, 0->2, 0->3.
Rozważmy krawędz 0->1:
Przed obliczeniem kosztu wierzchołka 1 dla krawędzi 0->1, musimy ustawić wszystkie ścieżki wychodzące z wierzchołka 0 i wszystkie ścieżki wchodzące do wierzchołka 1 na nieskończoność, a także ścieżkę powrotną do rozważanej ścieżki, czyli 1->0.

$$\begin{bmatrix} 'X' & 'X' & 'X' & 'X' \\ 'X' & 'X' & 0 & 2 \\ 0 & 'X' & 'X' & 0 \\ 0 & 'X' & 3 & 'X' \end{bmatrix}$$

Tak przygotowana macierz, jest gotowa do obliczenia ograniczenia.
Redukcja wierszy:

$$\begin{bmatrix} 'X' & 'X' & 'X' & 'X' \\ 'X' & 'X' & 0 & 2 \\ 0 & 'X' & 'X' & 0 \\ 0 & 'X' & 3 & 'X' \end{bmatrix}$$

Redukcja kolumn:

$$\begin{bmatrix} 'X' & 'X' & 'X' & 'X' \\ 'X' & 'X' & 0 & 2 \\ 0 & 'X' & 'X' & 0 \\ 0 & 'X' & 3 & 'X' \end{bmatrix}$$

Koszt ograniczenia: 0
Koszt wierzchołka 1 : Koszt poprzednich wierzchołkow + koszt ścieżki 0->1 + Koszt ograniczenia = 16 + 2 + 0 = 18
W ten sam sposób postępujemy dla krawędzi 0->2 i 0->3, następnie sprawdzamy, która krawędź ma najmniejszy koszt i powtarzamy procedurę, aż do momentu gdy będziemy mieli pewność która ścieżka ma najniższy koszt.
Należy szczególnie pamiętać o tym, żeby wykorzystywać odpowiednie dane np. gdy obliczamy koszt ścieżki 0->4->1->2, należy jako koszt poprzednich wierzchołków użyć kosztu ścieżki 0->4->1, i aby obliczyć koszt ograniczenia będziemy redukować wiersze i kolumny z macierzy 0->4->1.

3 Opis implementacji algorytmu

Macierz w przypadku wszystkich algorytmów została zaimplementowana w dwuwymiarowej liście.

3.1 Metoda podziału i ograniczeń

Do zaimplementowania metody podziału i ograniczeń stworzyłem trzy klasy:

- LiveNode- klasa przechowująca informację o wierzchołku, posiada zmienne cost, matrix oraz route, zawierające informacje o koszcie wierzchołka, macierzy, z której został wyliczony jego koszt oraz jego ścieżki.
- NodeHelper- klasa zawierająca metody statyczne pomagajace w działaniach nad kolekcją obiektów LiveNode
- NewBranchAndBounder - klasa posiadająca metody potrzebne do obliczenia trasy metodą podziału i ograniczeń

Funkcja wyliczająca najlepszą ścieżkę nazywa się get_best_route.
Na początku w funkcji modyfikowana jest macierz, która została podana w konstruktorze obiektu, ze względu na redukcję kolumn i wierszy dla korzenia. Tworzone są też zmienne node- wierzchołek, który jest rozpatrywany, live_nodes- lista zawierająca informacje o wszystkich rozpatrzonych wierzchołkach oraz zmienna pomocnicza last_best_node.
Reszta algorytmu zawarta jest w podwójnej pętli- w pętli zewnętrznej zawarty jest warunek kończący działanie funkcji oraz wybranie następnego wierzchołka do rozwinięcia, za pomocą sortowania kolekcji live_nodes po zawartych tam informacjach o kosztach. Jest tam również funkcja get_routes_to_check, która wylicza jakie ścieżki należy rozpatrzyć dla wierzchołka (np. w macierzy 4x4 dla korzenia 0 ścieżkami do rozpatrzenia będą 0-1, 0-2 oraz 0-3). W pętli wewnętrznej obliczany i zapisywany do live_nodes jest nowy element live_node. Jego koszt jest obliczany, dzięki funkcjom: prep_matrix, która zamyka odpowiednie ścieżki i przygotowuje nową macierz do obliczenia dolnego ograniczenia, oraz funkcja get_lower_bound, która zawiera funkcje ograniczające wiersze i kolumny, zwraca ona wartość ograniczenia.
Funkcja kończy się, kiedy po posortowaniu live_nodes po kosztach, w dwóch kolejnych iteracjach zostanie zwrócona ta sama wartość.
W algorytmie nie zostały wykorzystane żadne wspomagające biblioteki, ze struktur danych najczęściej wykorzystywane były listy.

4 Plan eksperymentu

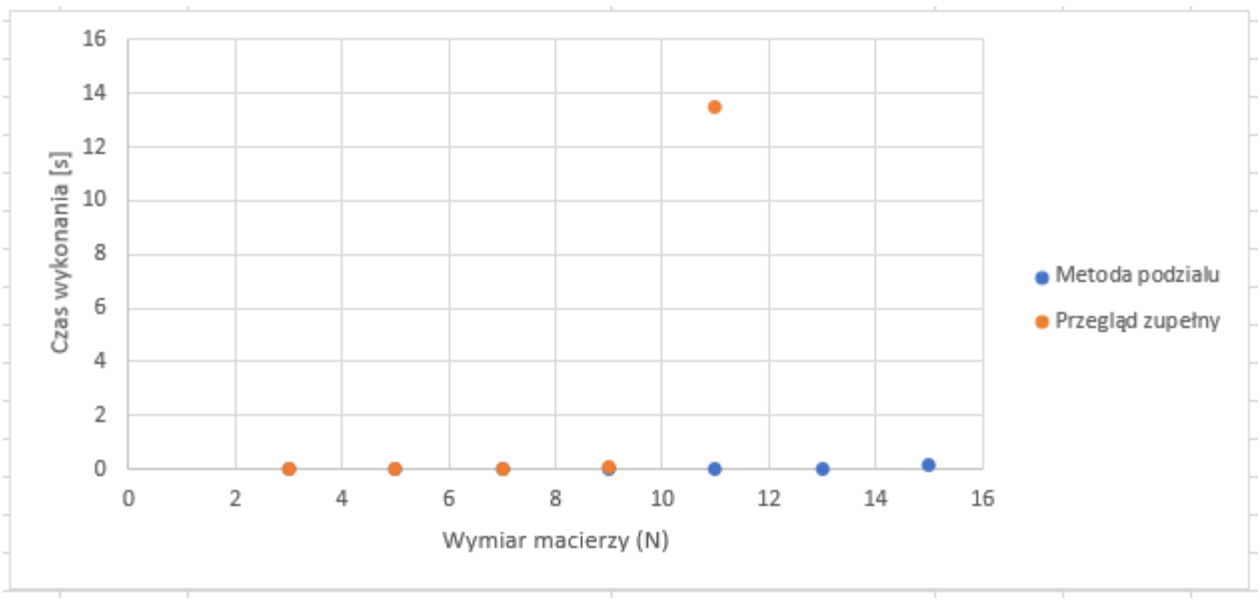
Eksperyment wygenerowano dla 100 różnych instancji problemu N, gdzie N przyjmuje wartość: 3,5,7,9,11,13,15, oraz dla 10 różnych instacji problemu dla kolejnych N aż do 23, tylko dla metody podziału i ograniczeń.
Dane zostały wygenerowane w sposób losowy za pomocą biblioteki random, gdzie waga ścieżek przybrała wartość od 0 do 50, pomiaru czasu dokonano za pomocą biblioteki timeit.

5 Wyniki eksperymentów

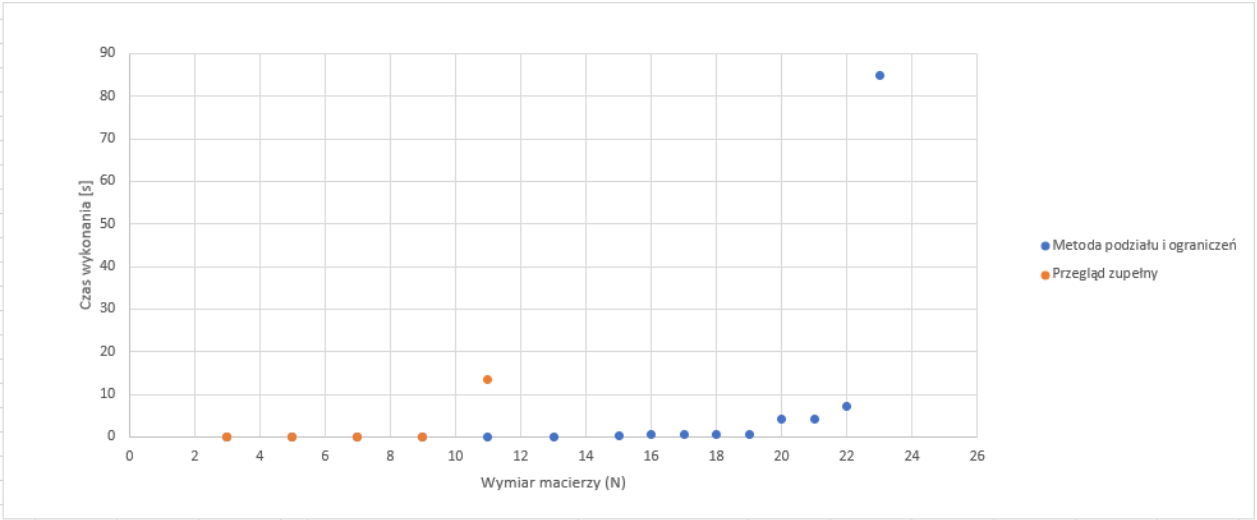
Algorytm programowania dynamicznego zwrócił czasy zbliżone do przęgladu zupełnego i większe, co świadczy o błędach w implementacji, dlatego nie będe zawierał dotyczących go danych w wynikach.

N	Metoda podziału i ograniczeń (s)	Przegląd zupełny (s)
3	5,91E-05	6,11E-06
5	0,00027297	4,26E-05
7	0,00125247	0,00157562
9	0,00292314	0,11047008
11	0,02820721	13,4981871
13	0,045251333	?
15	0,15623086	?
16	0,54139	?
17	0,592212	?
18	0,712573	?
19	0,71257	?
20	4,25203505	?
21	4,31132513	?
22	7,26012926	?
23	84,98842848	?

Rysunek 1: Wyniki eksperymentu, N oznacza wymiar macierzy



Rysunek 2: Wynik eksperymentu na wykresie, gdzie maksymalne N=11



Rysunek 3: Wyniki ze wszystkimi N z tabeli

6 Wnioski

- Algorytm programowania dynamicznego nie został zaimplementowany prawidłowo, ze względu na brak struktury danych zapamiętującej wyniki, co jest główną różnicą między algorytmem dynamicznym a przeglądem zupełnym. Dodatkowo moja prosta implementacja pozwala na łatwe sumowanie i wyświetlanie kosztów, ale ciężko pozyskać z niej trasę w postaci kolejnych wierzchołków, co mi się nie udało.
- Dla małych wartości N czasy metody podziału i ograniczeń i przeglądu zupełnego są podobne, wynika to z tego, że w przeglądzie metody podziału i ograniczeń przeprowadzane są bardziej skomplikowane działania, przez co czas jest dłuższy.
- Czas generowania N=13 i N=15 dla przeglądu zupełnego był na tyle długi, że nie mogłem go wygenerować.

- Czas obliczania ścieżek w metodzie podziału i ograniczeń w dużej mierze zależy od danych, w najkorzystniejszym przypadku trzeba obliczyć koszt n ścieżek w pesymistycznym n! ścieżek, co oznacza że czas wykonania dla pesymistycznego przypadku jest (n-1)! razy większy.

7 Literatura

[1] https://en.wikipedia.org/wiki/Travelling_salesman_problem

[2] http://cpp0x.pl/kursy/Teoria-w-Informatyce/Zlozonosc-obliczeniowa/Klasy-problemow-NP/430

[3] https://www.techiedelight.com/travelling-salesman-problem-using-branch-and-bound/

[4] http://lcm.csa.iisc.ernet.in/dsa/node187.html

[5] https://imada.sdu.dk/ jbj/DM85/bab2.pdf

[6] https://www2.seas.gwu.edu/ bell/csci212/Branch_and_Bound.pdf

[7] https://www.youtube.com/watch?v=JE0JE8ce1V0