Problem Komiwojażera

# Zadanie projektowe nr 1

## Programowanie Efektywnych algorytmów

Maciej Kucharczyk, 259211

1. **Wstęp**

Problem Komiwojażera jest zagadnieniem optymalizacyjnym, w którym należy znaleźć minimalny/maksymalny cykl Hamiltona w grafie ważonym.

Problem ten charakteryzuje się dużą ilością danych do analizy. Istnieje wiele metod rozwiązań tego problemu, lecz my skupimy się na dwóch: algorytm przeglądu zupełnego (Brute Force) oraz algorytm podziału i ograniczeń (Branch & Bound).

Oba algorytmy mają wykładniczą złożoność obliczeniową O(n!) i są algorytmami dokładnymi.

Algorytm przeglądu zupełnego sprawdza każdy możliwy wariant ułożenia wierzchołków w grafie, po czym wybiera ten najkorzystniejszy.

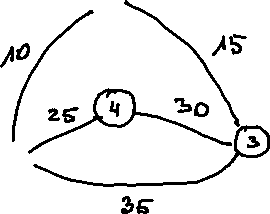
Algorytm podziału i ograniczeń wyszukuje optymalne rozwiązanie bez konieczności sprawdzania wszystkich rozwiązań (w najgorszym przypadku wykonuje przegląd zupełny). Jego działanie opiera się na tworzeniu drzewa, którego reprezentują potencjalne rozwiązania. W każdym etapie algorytm odrzuca gałęzie, które reprezentują rozwiązania najmniej korzystne, w związku z czym ogranicza się finalną liczbę wariantów do rozpatrzenia.



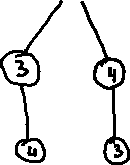
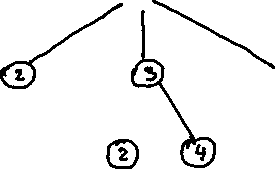
1. **Praktyczne objaśnienia algorytmów**

2.1 Przegląd zupełny

Przykładowy graf:



1. Wyznaczamy wszystkie możliwe ścieżki będące cyklami Hamiltona:



1. Korzystając z wag krawędzi w grafie obliczamy koszt każdej drogi. Zaczynamy od drogi A:

Koszt(A) = 95

Koszt(B) = 80

Koszt(C) = 95

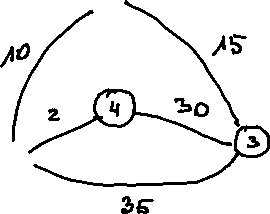
Koszt(D) = 80

Koszt(E) = 95

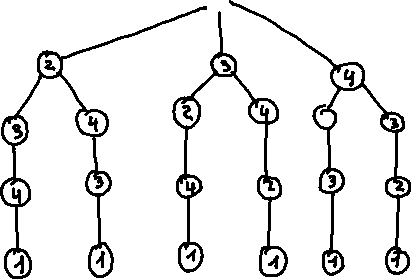
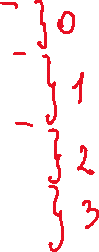
Koszt(F) = 95

1. Wyznaczamy najniższy koszt. U nas najniższy koszt ma droga B oraz D i wynosi on 80. Jednak naszym rozwiązaniem będzie droga B, ponieważ została wyliczona wcześniej niż droga D.
2. Rozwiązanie: droga B.
   1. Algorytm Podziału i Ograniczeń

Przykładowy graf:



Wyznaczenie wszystkich możliwych ścieżek będącymi cyklami Hamiltona (czerwoną przerywaną linią oddzielono poszczególne poziomy w drzewie):



1. Zgodnie z założeniami algorytmu, musimy nadać pewne ograniczenia podczas poszukiwać najlepszej trasy. Na początku należy wyliczyć minimalne ograniczenie kosztu drogi.

Do wyliczenia ograniczenia skorzystamy ze wzoru:

gdzie:

- *L* to dolne ograniczenie kosztu drogi

- *W1i , W2i* to dwie minimalne krawędzie wychodzące z wierzchołka i

- wierzchołek *i* należy do zbioru wierzchołków w grafie

Ponieważ koszt jakiejkolwiek ścieżki, jest równy połowie sumy sum dwóch dowolnych krawędzi wychodzących z każdego wierzchołka, to koszt dowolnej podróży będzie większy od połowy sumy sum dwóch minimalnych krawędzi wychodzących z każdego wierzchołka.

Dzięki temu wiemy, że najlepsze rozwiązanie jest większe lub równe wyliczonemu właśnie dolnemu ograniczeniu. Zatem jeśli podczas badania danej ścieżki okaże się, że jej koszt jest mniejszy niż L, to odrzucamy je.

U nas ograniczenie wynosi:

L = 0.5\*[ (10 + 15) + (10 + 25) + (30 + 15) + (20 + 25) ] = 75

1. Następnie wybieramy jedną ścieżkę (u nas będzie to ścieżka A) i wyliczamy dolne ograniczenie dla następnego wierzchołka z tej ścieżki z poziomu pierwszego (u nas będzie to wierzchołek 1 na poziomie pierwszym). Użyjemy do tego wzoru:

gdzie:

- Lp  to poprzednio wyliczone dolne ograniczenie

- min(1) to koszt minimalnej krawędzi wychodzącej z wierzchołka 1

- min(2) to koszt minimalnej krawędzi wychodzącej z wierzchołka 2

Zatem na tym etapie ograniczenie wynosi:

L(1) = 75 – 0.5\*(10+10) + 10 = 65

1. Teraz porównujemy otrzymany wynik z uzyskanym optymalnym dotąd rozwiązaniem. Jeżeli wynik będzie mniejszy, to rozpatrujemy kolejne wierzchołki na ścieżce. W przeciwnym wypadku odrzucamy tą ścieżkę i przechodzimy do kolejnej. Na razie nie wyliczyliśmy optymalnego rozwiązania, więc przyjmujemy, że ma ono wartość nieskończenie dużą (zatem rozpatrujemy ścieżkę dalej). Kolejną czynnością będzie znów wyliczenie dolnego ograniczenia dla kolejnego wierzchołka na rozpatrywanej ścieżce (u nas wierzchołek 2). Jednak nie możemy użyć wzoru z poprzedniego etapu, ponieważ znowu dodalibyśmy do naszego ograniczenia min(2). Zatem w tym etapie (i w każdym następnym) posłużymy się następującym wzorem:

gdzie:

- sec\_min(2) to druga z kolei minimalna krawędź wychodząca z wierzchołka 2

Na tym etapie ograniczenie wynosi:

L(2) = 90

Czynność z podpunktu c) kontynuujemy dla całej ścieżki i dochodzimy do momentu, w którym ścieżka A ma rozwiązanie optymalne, wynoszące 95.

Dla kolejnych gałęzi wykonujemy czynności z podpunktów a), b) oraz c). Obliczamy ograniczenie zgodnie ze wzorami dla każdego poziomu w ścieżce, a następnie porównujemy je z dotychczas znalezionym najlepszym rozwiązaniem. Jeżeli podczas rozpatrywania ścieżki na danym poziomie okaże się, że dolne ograniczenie jest większe niż najlepsze dotychczas rozwiązanie, do kończymy rozpatrywanie tej ścieżki i przechodzimy do kolejnej. W przeciwnym wypadku rozpatrujemy ścieżkę dalej. W ten sposób pomijamy ścieżki, które już na etapie rozpatrywania mają większy koszt niż dotychczas wyliczony najniższy koszt i dochodzimy do rozwiązania optymalnego. U nas tym rozwiązaniem jest ścieżka B, a jej koszt wynosi 80.

1. **Opis implementacji algorytmów**

Algorytmy jako dane wejściowe przyjmowały macierz sąsiedztwa. Do jej przechowywania wykorzystano strukturę vector z biblioteki STL. Zarówno algorytm przeglądu zupełnego jak i algorytm podziału i ograniczeń były zdefiniowane w osobnych klasach. Dodatkowo w przypadku algorytmu podziału i ograniczeń zdefiniowano dwie dodatkowe struktury: listę przechowującą ścieżkę z optymalnym rozwiązaniem, oraz boolowksą listę odwiedzin, która przechowywała informację, czy dany wierzchołek został już rozpatrzony podczas analizy danej ścieżki. Do przechowania obu struktur ponownie wykorzystano strukturę vector z biblioteki STL.

W przypadku algorytmu Brute Force istotny był sposób generowania kombinacji miast. Wykorzystano tutaj metodę next\_permutation(), która przyjmuje dwa argumenty: początek oraz koniec wektora, który to przechowywał listę miast. Funkcja ta generuje za każdym razem inną kombinację, dzięki czemu każda ścieżka mogła być rozpatrzona tylko raz.

W przypadku algorytmu Branch & Bound ważnym aspektem było zdefiniowanie sposobu obliczania ograniczenia. W tym celu zaimplementowano funkcję CheckLevel(), która była rekurencyjnie wywoływana na każdym poziomie rozpatrywanej ścieżki. Dla każdego poziomu wyliczała ona dolne ograniczenie zgodnie ze wzorami przedstawionymi z punkcie 2. Funkcja jako jeden z argumentów przyjmowała poziom w drzewie, dzięki czemu używała do tego odpowiedniego wzoru. Następnie porównywała obliczoną wartość z najlepszym dotychczas rozwiązaniem, i jeśli ograniczenie było niższe, to funkcja była rekurencyjnie wywoływana dla następnego poziomu, przekazując jako argumenty wartość poziomu, aktualne ograniczenie, listę odwiedzonych wierzchołków oraz listę informującą, które wierzchołki nie zostały jeszcze odwiedzone. W przeciwnym wypadku kończyła rozpatrywanie ścieżki i przechodziła do kolejnej.

1. **Plan eksperymentu**

Do wykonania eksperymentu należało wygenerować macierze sąsiedztwa o różnych rozmiarach i o różnym rozkładzie danych. Nasze eksperymenty były wykonane na macierzach kwadratowych o 7 rozmiarach, zaczynając od rozmiaru 6x6, i potem rozmiar był zwiększany o 1, aż do rozmiaru 12x12. Dla każdego rozmiaru wygenerowano sto macierzy o różnym rozkładzie danych, z założeniem, że waga krawędzi była losowana z przedziału 0 – 99 oraz, że dla tego samego indeksu wiersza i kolumny wartość komórki wynosiła 0. Dla danego rozmiaru i rozkładu danych wykonano pomiary czasu, a wyniki dla każdego rozmiaru uśredniono. Do wykonania pomiaru czasu użyto biblioteki chrono, a jednostką pomiaru była milisekunda.

1. **Wyniki eksperymentów**

Przedstawienie wyników pomiarów w formie tabeli:



Przedstawienie wyników w formie graficznej:

1. **Wnioski i podsumowanie**

Z uzyskanych wyników można wywnioskować, iż złożoności obliczeniowe obu algorytmów są zgodne z wartościami teoretycznymi. Dodatkowo algorytm przeglądu zupełnego jest dużo wolniejszy od algorytmu podziału i ograniczeń. Jednakże dla ilości miast mniejszej niż 11 przegląd zupełny szybciej zwracał optymalne rozwiązanie. Mogło to być spowodowane tym, że w algorytmie Branch & Bound wyliczanie ograniczenia dla każdego wierzchołka na danej ścieżce nie było wystarczająco optymalne, a ilość ścieżek była na tyle mała, że rozpatrzenie każdej z osobna mogło dawać szybsze rezultaty.

Jedną z trudności podczas wykonywania eksperymentu było dobranie odpowiednich rozmiarów macierzy do przeprowadzania testów. Przy liczbie miast równej 17 czasy wykonywania obu algorytmów były bardzo duże, więc metodą prób i błędów trzeba było dobrać odpowiedni górny limit.