SPRAWOZDANIE

Projektowanie efektywnych algorytmów

Zadanie projektowe nr 1

Implementacja i analiza efektywności algorytmu zachłannego, podziału i ograniczeń oraz programowania dynamicznego dla problemu komiwojażera (TSP)

Autor:
Patryk Dulęba 259213
Czwartek 11:15

Prowadzący:
Mgr inż. Antoni Sterna

Spis treści

1 Opis rozpatrywanego problemu

Celem tego zadania projektowego było zaimplementowanie oraz późniejsza analiza efektywności algorytmów znajdujących rozwiązania dla problemu komiwojażera. Problem komiwojażera polega na poszukiwaniu w grafie takiego cyklu, który zawiera wszystkie wierzchołki (każdy tylko raz), kończy się i zaczyna w tym samym punkcie i ma jak najmniejszy koszt całej drogi. Inaczej mówiąc problem komiwojażera polega na znalezieniu cyklu Hammiltona o najmniejszej wadze.

Aktualnie nie jest znany efektywny, czyli działający w czasie co najwyżej wielomianowym algorytm dający gwarancję znalezienia optymalnego rozwiązania problemu komiwojażera. Z tego powodu problem ten jest zaliczany do klasy problemów NP-trudnych. Dzieje się tak z powodu ogromnej liczby kombinacji do sprawdzenia przy grafach o większej liczbie wierzchołków. W grafie pełnym mającym n wierzchołków liczba możliwych cykli Hammiltona wynosi aż $\frac{(n-1!)}{2}$.

2 Rozpatrywane algorytmy rozwiązujące problem TSP

Poniżej zostały omówione wszystkie algorytmy rozwiązujące problem komiwojażera zaimplementowane w programie w ramach tego zadania projektowego.

2.1 Przegląd zupełny (Brute Force)

Najprostszą w implementacji metodą rozwiązującą rozważany problem jest metoda przeglądu zupełnego, która polega na znalezieniu wszystkich możliwych rozwiązań problemu, oraz wybraniu tego najlepszego w tym przypadku o najniższym koszcie. Algorytm jest bardzo prosty, natomiast sprawdzenie przez niego wszystkich możliwych kombinacji jest bardzo czasochłonne, a dla większych instancji niemalże niewykonalne. Złożoność czasowa tej metody to O(n!). Główną zaletą tej metody jest pewność, że znajdziemy optymalne rozwiązanie, o ile takie istnieje.

2.1.1 Implementacja algorytmu przeglądu zupełnego dla problemu TSP

Implementacja tego algorytmu jest trywialna i polega na przeglądaniu w dwóch pętlach wszystkich możliwych kombinacji ścieżek dostępnych w zadanym grafie. Struktury danych, jakie zostały użyte to:

- **Tablice** - jednowymiarowa oraz dwuwymiarowa, przechowująca koszty danych ścieżek oraz aktualną ścieżkę.
- **Wektor** - przechowujący wszystkie wierzchołki.

2.2 Podział i ograniczenia (Branch & Bound)

Metoda podziału i ograniczeń działa w nieco inny sposób niż przegląd zupełny. W tym przypadku nie będziemy musieli przejrzeć całego drzewa, aby dostać optymalne rozwiązanie. Algorytm zaczyna w korzeniu drzewa i przechodząc do któregoś liścia, konstruuje rozwiązanie. W celu zawężenia obszaru przeszukiwań metoda podziału i ograniczeń w każdym węźle oblicza granicę, która pozwala określić go jako obiecujący bądź nie. W dalszej fazie algorytm przegląda tylko potomków węzłów obiecujących. Pozwala to, razem z dobraniem odpowiedniej strategii odwiedzania wierzchołków (branch) oraz liczenia granicy (bound), zmniejszyć ilość odwiedzonych wierzchołków i szybciej znaleźć optymalne rozwiązanie problemu. Podobnie jak w przypadku przeglądu zupełnego, złożoność obliczeniową tej metody ogranicza funkcja wykładnicza n!.

2.2.1 Implementacja algorytmu podziału i ograniczania dla problemu TSP

Działanie algorytmu rozpoczyna się od utworzenia węzła korzenia, który posiada macierz wag, wektor z obecną ścieżką oraz informację o obecnym poziomie drzewa. W kolejnym kroku macierz przechowywana w węźle zostaje zredukowana poprzez odjęcie najmniejszego elementu z wiersza i kolumny. Odjęte wartości dodawane są do kosztu węzła, następnie taki węzeł dodawany jest do kolejki priorytetowej, która przechowuje dane w taki sposób, aby węzeł o najniższym koszcie znajdował się na szczycie. Następnie wchodzimy do

głównej pętli przeszukującej, gdzie wyliczamy koszta dla dzieci węzła znajdującego się na szczycie kolejki. Dla każdego wychodzącego węzła na podstawie macierzy rodzica, ustawiane są nieskończoności w miejscach wiersza rodzica i kolumny dziecka oraz zostaje dodany koszt dostępu z wierzchołka rodzica do wierzchołka dziecka. Następnie obliczamy koszt dla wychodzącego węzła na podstawie zredukowanej macierzy, który jest dodawany do wcześniej uzyskanego kosztu dostępu. Każdego potomka dodajemy do kolejki, następnie uruchamiana jest kolejna iteracja pętli dla węzła ze szczytu i tak do momentu opróżnienia kolejki, bądź dojścia do węzła znajdującego się na poziomie o jeden mniejszym od ilości wierzchołków w grafie. Do implementacji opisywanego algorytmu zostały wykorzystane struktury danych takie jak:

- **Tablice** - jednowymiarowe oraz dwuwymiarowe, przechowujące koszty danych ścieżek, kolumn, wierszy.
- **Wektor** - przechowujący aktualną ścieżkę.
- **Kolejka priorytetowa** - zawierająca węzły, których priorytet wyznacza obliczona dla każdego z nich wartość ograniczenia dolnego. Najwyższy priorytet ma wierzchołek o najniższej wartości dolnego ograniczenia.

2.3 Programowanie dynamiczne (Dynamic Programming)

Ostatnią wykorzystaną w tym projekcie metodą rozwiązywania problemu komiwojażera jest programowanie dynamiczne. Polega ona na dzieleniu problemu na mniejsze podproblemy oraz rozwiązywania ich począwszy od tych najprostszych i wykorzystywaniu ich wyników do rozwiązywania kolejnych, bardziej złożonych. Wyniki poszczególnych podproblemów zapisywane są w tablicy stanów. Złożoność obliczeniowa takiego algorytmu wynosi $O(n^2 2^n)$.

2.3.1 Implementacja algorytmu programowania dynamicznego dla problemu TSP

Działanie całego algorytmu opiera się w głównej mierze na zapełnieniu tablicy stanów. Każdy nieuporządkowany podzbiór reprezentowany jest przez maskę bitową, są to słowa bitowe, w których waga bitu odpowiada numerom poszczególnych wierzchołków. Zasada działania jest bardzo prosta, jeśli na danej pozycji słowa bitowego jest 1, to wierzchołek odpowiadający tej pozycji znajduje się w tym podzbiorze i analogicznie, jeśli jest 0, to znaczy, że danego wierzchołka nie ma w tym podzbiorze. Tablica stanów jest tablicą dwuwymiarową, gdzie liczba wierszy jest równa 2^n co odpowiada liczbie wszystkich możliwych kombinacji, czyli podzbiorów, natomiast liczba kolumn to n, czyli liczba wszystkich wierzchołków w badanym grafie. Do implementacji algorytmu zostały wykorzystane takie struktury danych jak:

- **Tablice** - dwuwymiarowe, przechowujące koszty danych ścieżek oraz rozwiązania podproblemów.
- **Wektor** - przechowujący aktualną ścieżkę.

3 Plan eksperymentu

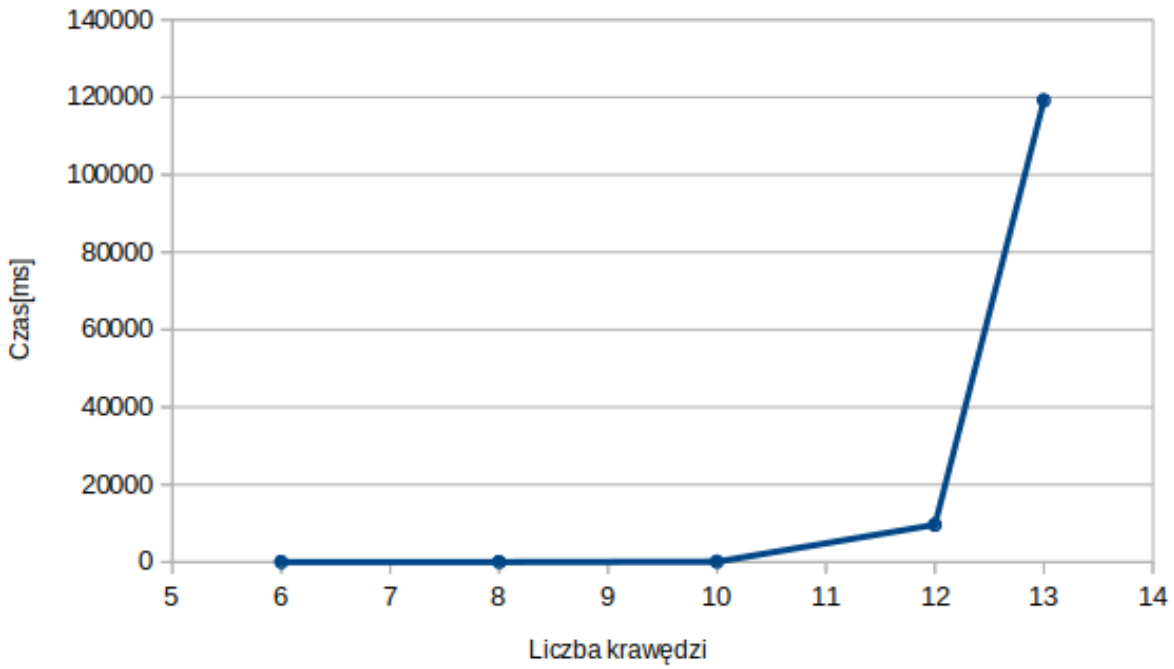
W celu porównania efektywności wszystkich omówionych algorytmów został zmierzony czas rozwiązywania problemu komiwojażera dla 7 różnych pod względem wielkości instancji tego problemu. Dane testowe generowane były automatycznie, tworząc macierz sąsiedztwa z losowymi wartościami mieszczącymi się w przedziale 1-100, co zagwarantowało wygenerowanie asymetrycznych instancji problemu. Wartości po przekątnej, czyli droga z wierzchołka do niego samego ustawiona została na 0. Struktury danych przechowujące dane wejściowe były alokowane dynamicznie. Badane zostały przypadki, w których liczba wierzchołków wynosiła od 6 do 15. Pomiar czasu został wykonany za pomocą QueryPerformanceCounter. Dla każdej instancji problemu zostało wykonane 100 pomiarów, a następnie z uzyskanych wyników wyliczona została średnia arytmetyczna.

4 Wyniki eksperymentu

Liczba wierzchołków	Brute Force	Branch & Bound	Dynamic Programming
6	0,0713	0,5432	0,0331
8	2,8089	2,7027	0,1955
10	85,8436	28,3425	0,6297
12	9612,3112	372,8021	2,8728
13	119167,0781	1154,9223	5,3672
14	-	8617,6456	12,0161
15	-	17157,3877	26,5655

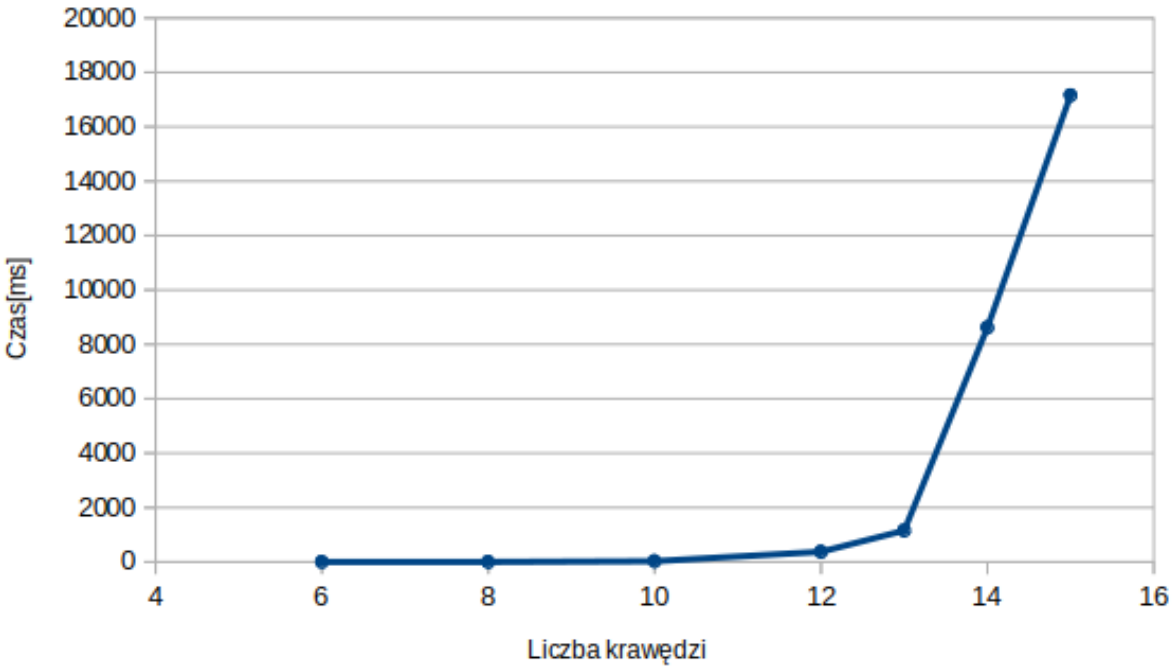
Tabela 1: Wyniki pomiarów czasu wykonania zaimplementowanych algorytmów.

4.1 Przegląd zupełny (Brute Force)



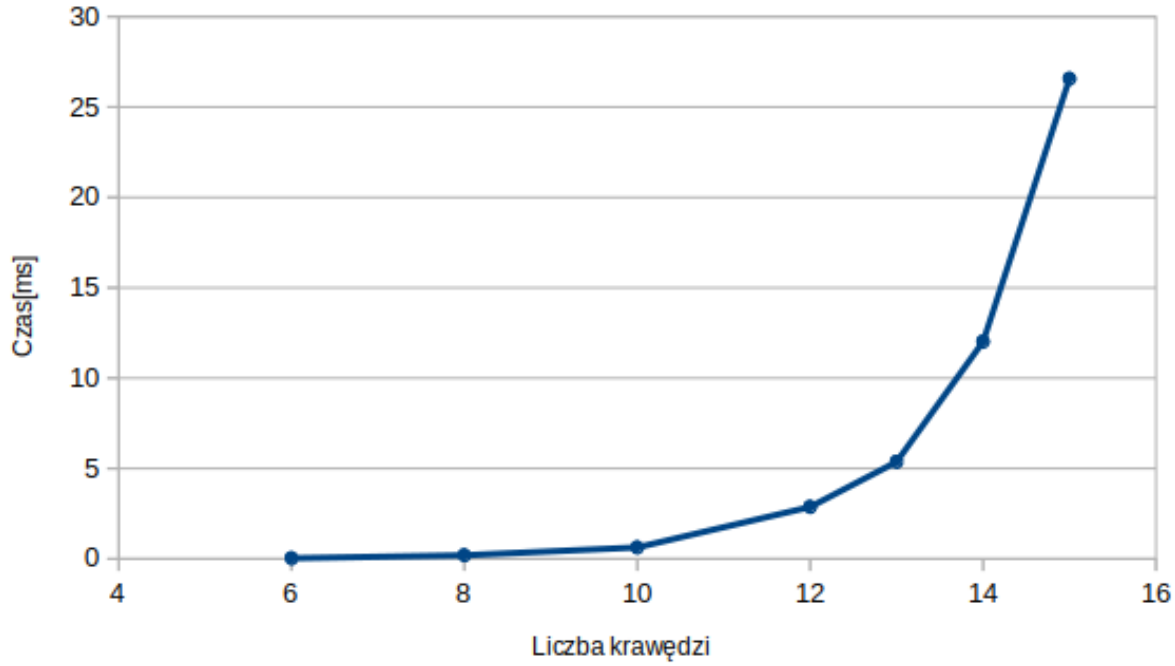
Rysunek 1: Zależność czasu wykonania od rozmiaru problemu dla przeglądu zupełnego

4.2 Podział i ograniczenia (Branch & Bound)



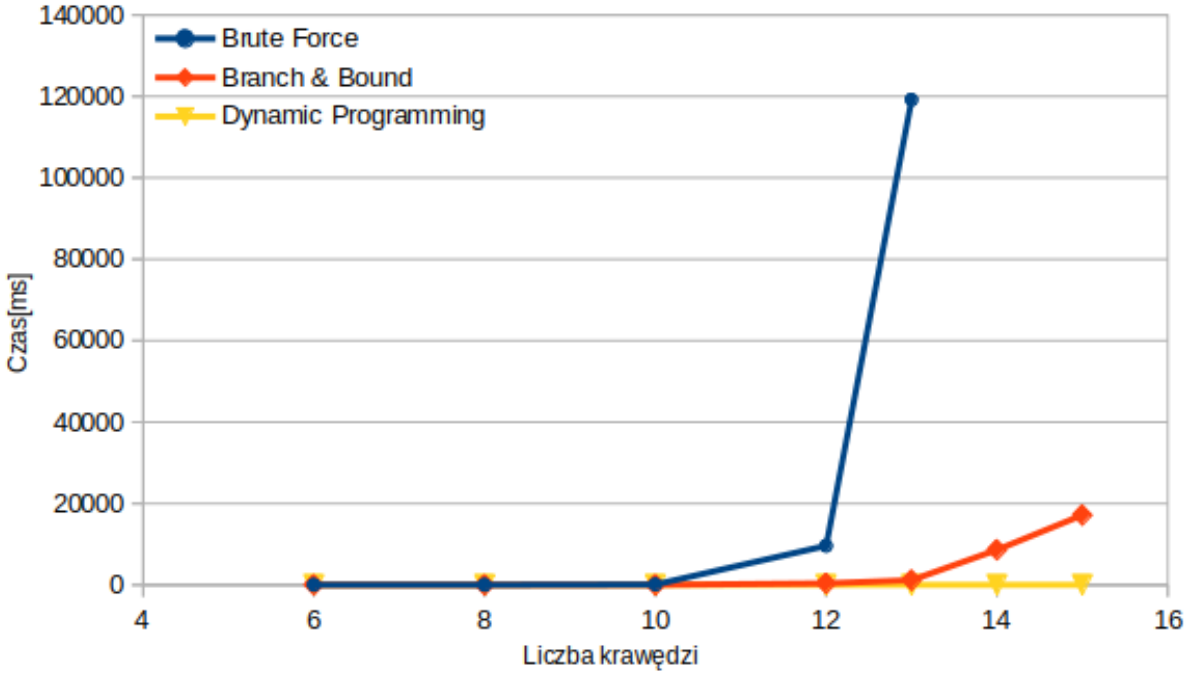
Rysunek 2: Zależność czasu wykonania od rozmiaru problemu dla metody podziału i ograniczeń

4.3 Programowanie dynamiczne (Dynamic Programming)



Rysunek 3: Zależność czasu wykonania od rozmiaru problemu dla programowania dynamicznego

4.4 Porównanie efektywności wszystkich algorytmów



Rysunek 4: Ogólne zestawienie efektywności

5 Wnioski

Na podstawie otrzymanych wyników oraz wygenerowanych z nich wykresów możemy stwierdzić, iż złożoność algorytmu przeglądu zupełnego jest zgodna ze złożonością teoretyczną wynoszącą O(n!). Podczas wykonywania eksperymentu dla tej metody byłem zmuszony przerwać działanie algorytmu dla instancji problemu, których wielkość przekraczała 13 wierzchołków. Powodem takiego działania był zbyt długi czas potrzeby do rozwiązania problemu przez metodę naiwną.

Następnym zaimplementowanym algorytmem był algorytm B&B. Z otrzymanego wykresu złożoności wynika, że jego złożoność zgadza się ze złożonością wzorcową, którą podobnie jak w przypadku przeglądu zupełnego ogranicza funkcja wykładnicza n!.

Ostatnią użytą w eksperymencie metodą było programowanie dynamiczne. Tak jak w poprzednich przypadkach, również tutaj została zauważona zgodność z teoretyczną klasą złożoności, która wynosi $O(n^2 2^n)$

Podsumowując, wszystkie zaimplementowane metody rozwiązywania problemu TSP mają złożoność zgodną z tą wzorcową. Możemy zauważyć, iż najwydajniejszy okazał się algorytm wykorzystujący programowanie dynamiczne, uzyskane pomiary czasów są całe rzędy wielkości niższe od tych uzyskanych metodą B&B czy też przeglądu zupełnego, który z kolei wypadł w tym zestawieniu najgorzej.