Programowanie Efektywnych Algorytmów

Prowadzący zajęcia: dr inż. Jarosław Mierzwa
Termin zajęć: Poniedziałek godz. 15:15

Autor sprawozdania:
- Kamil Wojcieszak 264487

Projekt 1:

Zaimplementowane algorytmy:

- Brute Force
- Branch & Bound
- Dynamic Programming

1. Wstęp teoretyczny

Problem komiwojażera to jedno z fundamentalnych zagadnień w dziedzinie teorii grafów i optymalizacji kombinatorycznej. Polega on na znalezieniu najkrótszej ścieżki lub cyklu przechodzącego przez wszystkie wierzchołki grafu, przy założeniu, że każdy wierzchołek jest odwiedzony dokładnie raz, a następnie powraca się do punktu początkowego. Problem komiwojażera jest NP-trudny, co oznacza, że nie istnieje znany skuteczny algorytm rozwiązujący ten problem w czasie wielomianowym dla dowolnych danych wejściowych.

Złożoność obliczeniowa algorytmów:

Przegląd zupełny (Brute Force):
Złożoność czasowa: $O(N!)$, gdzie N to liczba miast. Przegląd zupełny sprawdza wszystkie możliwe permutacje tras, co czyni go nieefektywnym dla większych instancji problemu.

Podział i Ograniczenia (B&B):
Złożoność czasowa zależy od jakości funkcji ograniczającej. W najgorszym przypadku może być podobna do przeglądu zupełnego, ale zazwyczaj jest znacznie bardziej efektywna.

Programowanie Dynamiczne (DP):
Złożoność czasowa: $O(N^2 * 2^N)$ gdzie N to liczba miast. Algorytm DP dla TSP używa techniki programowania dynamicznego do rozwiązania podproblemów i unika wielokrotnego rozwiązywania tych samych problemów.

2.1 Implementacja algorytmu Branch & Bound

Algorytm ten dokonuje przeglądu drzewa rozwiązań w głąb, obliczając dla następników bieżącego węzła ograniczenia dolne. Przegląd jest kontynuowany w węźle, dla którego dolne ograniczenie jest co do wartości najniższe z pozostałych a zarazem niższe od ograniczenia górnego. Wybrany węzeł po jego przejściu, nie jest rozważany podczas dalszego przeglądu. Eksploracja kolejnych węzłów odbywa się rekurencyjnie, modyfikując przy tym bieżącą scieżkę oraz ograniczenie górne. Do implementacji opisywanego algorytmu zostały wykorzystane struktury danych takie jak:

- Tablice – dwuwymiarowa, przechowująca reprezentację grafu oraz dwie jednowymiarowe, jedna przechowująca informację o odwiedzonych wierzchołkach oraz jedna służąca do obliczania dolnych ograniczeń.
- Stos – instancja zapamiętująca ścieżkę będącą najlepszym znalezionym rozwiązaniem oraz instancja przechowująca ścieżkę tymczasową dla danego etapu wykonywania algorytmu.
- Kolejka priorytetowa – zawierająca węzły których priorytet wyznacza obliczona dla każdego z nich wartość ograniczenia dolnego. Najwyższy priorytet ma wierzchołek o najniższej wartości dolnego ograniczenia.

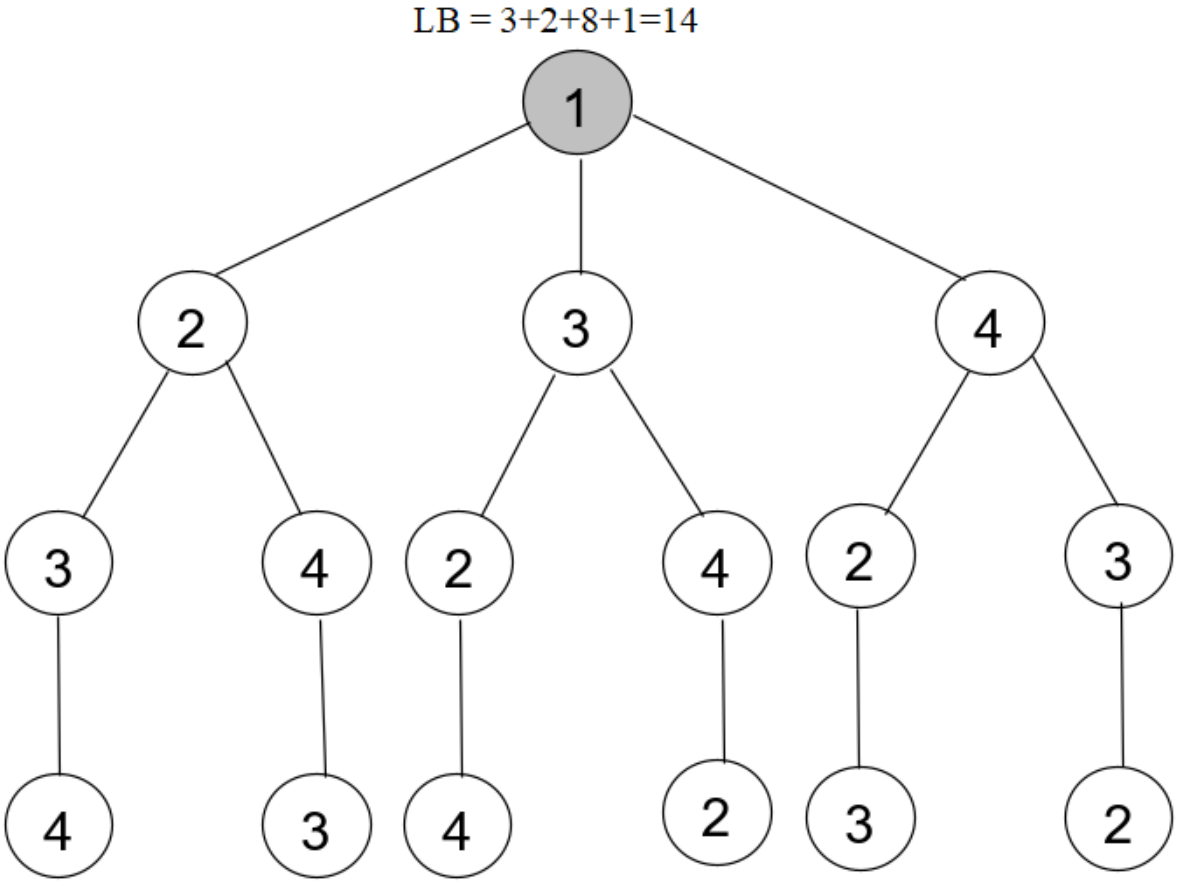
Obliczanie ograniczenia dolnego odbywa się poprzez wybór z wierszy macierzy krawędzi o jak najniższej wadze. Krawędzie te są zapamiętywane w tablicy której indeks odpowiada wierzchołkowi z którego wychodzi dana krawędź. Suma wag tych krawędzi wyznacza dolne ograniczenie dla wierzchołka początkowego. Ograniczenie to jest aktualizowane i przypisywane do każdego z następników bieżącego węzła. Odbywa się to według reguły przedstawionej na następującym przykładzie.

2.2. Przykład

Dany jest pełny graf ważony zadany macierzą sąsiedztwa oraz tablica pomocnicza zawierająca najniższe wagi krawędzi wychodzących z wierzchołków:

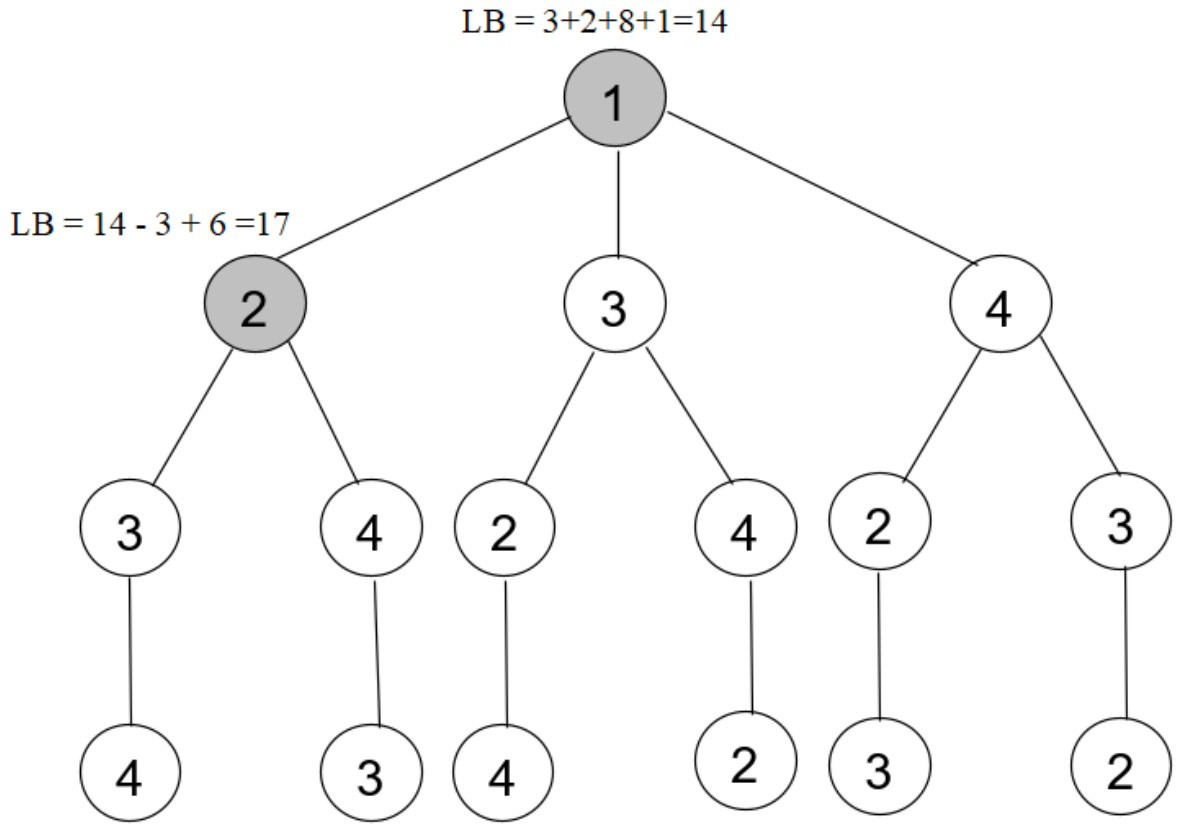
	1	**2**	**3**	**4**	Zawartość tablicy pomocniczej
1	-1	6	3	9	3
2	2	-1	5	4	2
3	10	8	-1	9	8
4	5	1	8	-1	1

Dolne ograniczenie dla wierzchołka początkowego (kiedy w trakcie przeglądu nie została przebyta jeszcze żadna krawędź) stanowi suma elementów tablicy pomocniczej.



Kolejnym etapem jest obliczenie dolnych ograniczeń dla następników. W tym celu od przekazanej do następnika wartości dolnego ograniczenia poprzednika należy odjąć wartość wagi odpowiedniej krawędzi z tablicy pomocniczej (w tym przypadku o indeksie pierwszym) oraz dodać wartość wagi krawędzi przebytej do następnika. Stąd dla np. wierzchołka drugiego można zapisać:

$$LB_{12} = 14 - min[1] + matrix[1][2]$$

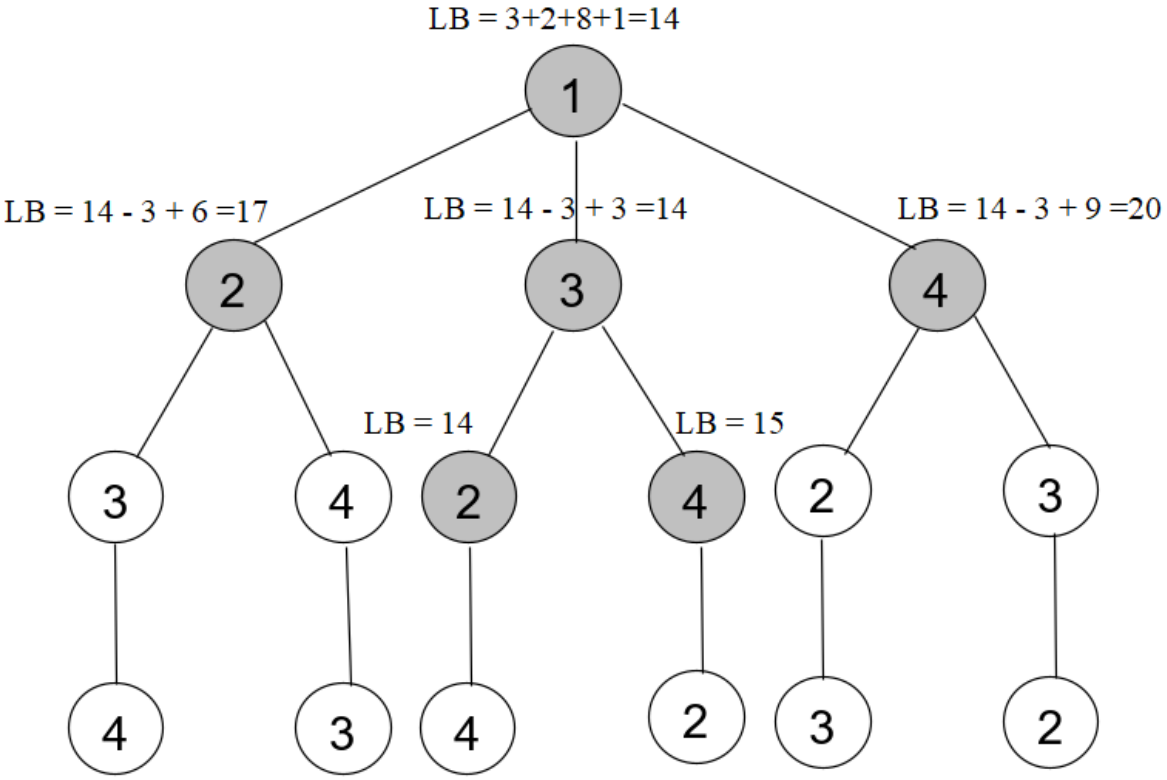


Zgodnie z tą regułą należy postąpić dla pozostałych następników, zatem dla węzła 3 oraz 4 będzie to kolejno:

$$LB_{13} = 14 - min[1] + matrix[1][3]$$
$$LB_{14} = 14 - min[1] + matrix[1][4]$$

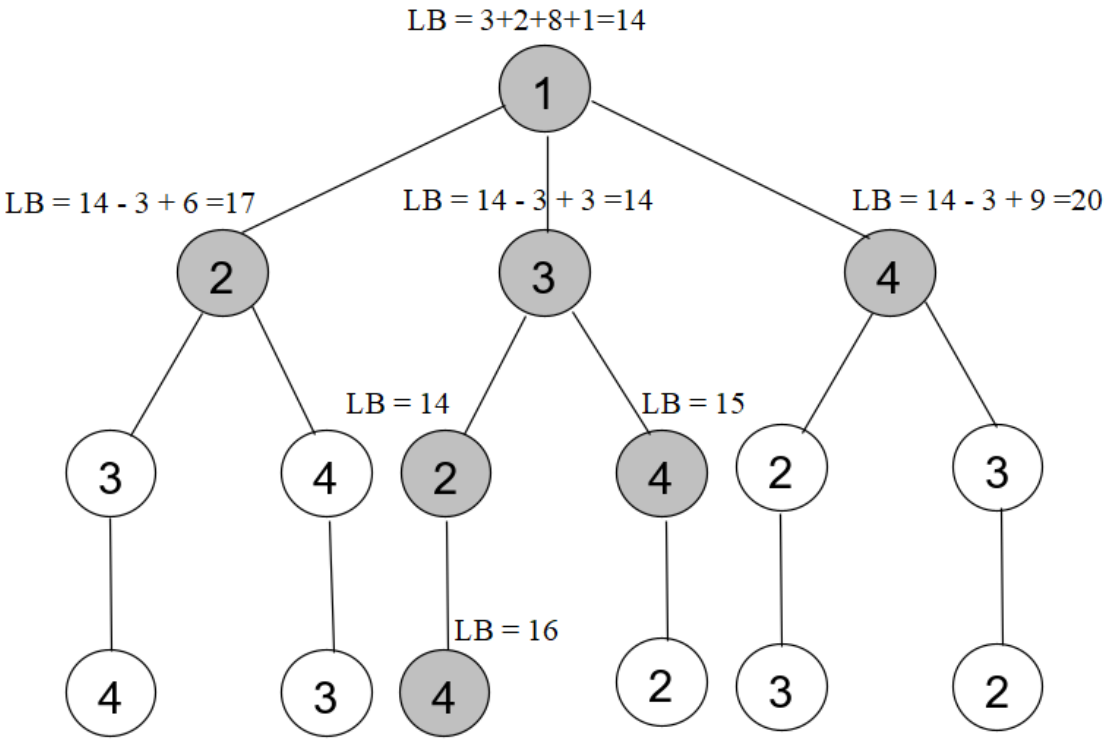
Następnym krokiem jest wybór węzła o najniższej wartości ograniczenia i jednocześnie mniejszej od górnego ograniczenia, które jest określone jako nieskończoność do momentu odnalezienia pierwszego rozwiązania. Należy zatem wybrać węzeł trzeci i dokonać analogicznej do węzła poprzedniego eksploracji gałęzi którą reprezentuje. Stąd:

$$LB_{32} = 14 - min[3] + matrix[3][2]$$
$$LB_{34} = 14 - min[3] + matrix[3][4]$$



Do dalszej eksploracji zostaje wybrany węzeł drugi. Policzone zostaje ograniczenie dolne dla jego następnika:

$LB_{24} = 14 - min[2] + marix[2][4]$



Węzeł czwarty jest liściem zatem osiągnięty został przypadek podstawowy. Należy sprawdzić zatem, czy suma wag utworzonej drogi po zamknięciu jej w wierzchołku startowym będzie mniejsza od górnego ograniczenia:

$$LB_{41} = 16 - min[4] + matrix[4][1]$$

Jako że jest to pierwsze znalezione rozwiązanie, spełniony jest warunek LB<UB. Można zatem zaktualizować wartość górnego ograniczenia przypisując do niego wartość bieżącego, dolnego ograniczenia:

$$UB = 20$$

Po ustanowieniu górnej granicy następuje przchodzenie do niższych poziomów drzewa w poszukiwaniu węzłów o spełniających warunek LB<UB i eksplorowanie ich w analogiczny sposób aż do momentu w którym żaden z węzłów nie będzie spełniał tego warunku. W takim wypadku nastąpi koniec algorytmu.

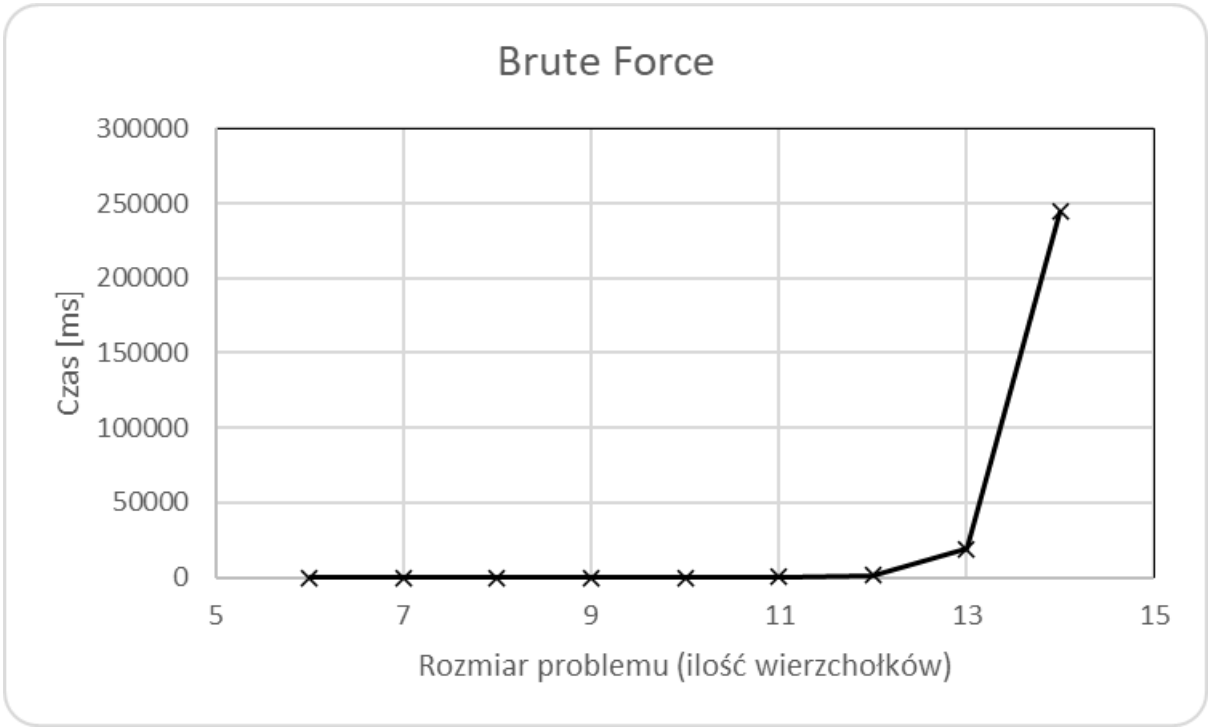
3. Plan eksperymentu.

W celu porównania efektywności rozpatrywanych metod, dla dziesięciu różnych pod względem wielkości instancji problemu komiwojażera, zostały przeprowadzone pomiary czasu wykonania każdego z algorytmów. Grafy wejściowe były generowane jako macierze zawierające losowe liczby z przedziału od 1 do 100, co zagwarantowało wygenerowanie asymetrycznych instancji problemu. Dane znajdujące się na ich przekątnej zostały potraktowane jako nieskończoności. Rozmiary użytych struktur są równe rozmiarom macierzy sąsiedztwa reprezentujących grafy zawierające od 8 do 18 wierzchołków. Pomiary czasów zostały przeprowadzone przy pomocy zegara o wysokiej rozdzielczości znajdującego się w bibliotece <chrono>. W następnym punkcie zestawione zostaną wyniki przeprowadzonego eksperymentu zarówno w postaci tabelarycznej jak i wykresów.

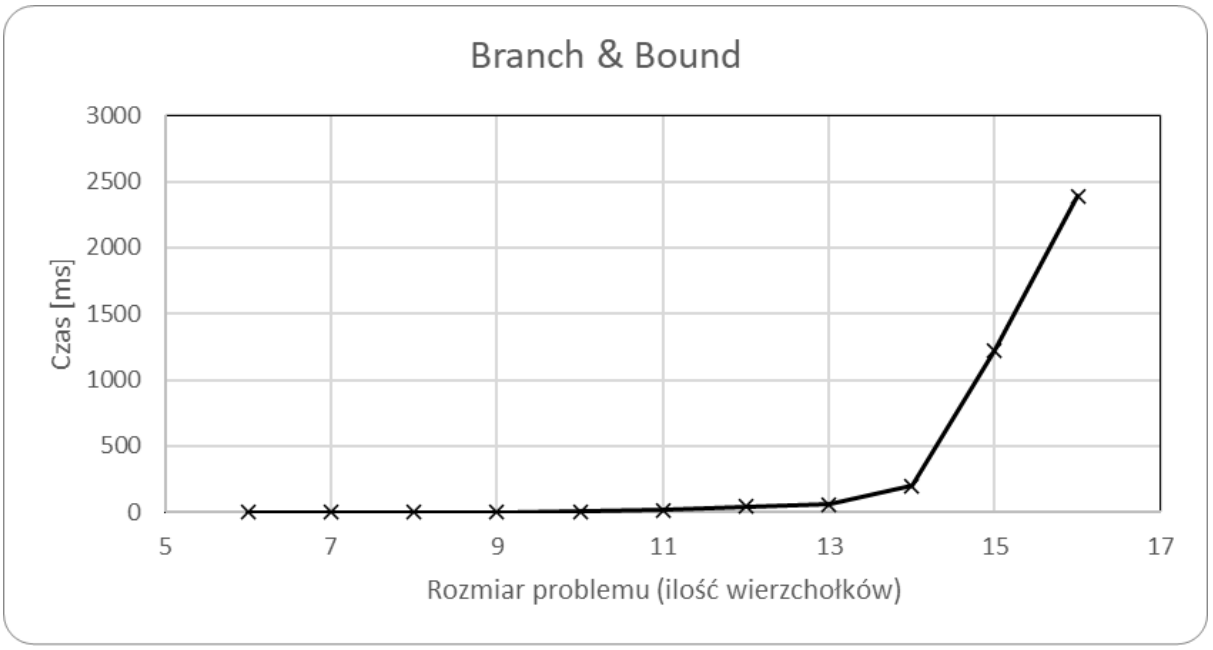
Czas wykonania dla danego algorytmu [ms]			
Rozmiar Problemu	Brute Force	Branch & Bound	Dynamic Programming
6	0,017	0,039	0,013
7	0,055	0,210	0,026
8	0,186	0,340	0,050
9	1,269	1,496	0,116
10	11,646	4,346	0,258
11	124,777	13,059	0,601
12	1431,746	43,758	1,433
13	18948,321	57,617	3,03
14	244360,553	196,637	9,240
15	-	1222,667	16,251
16	-	2387,887	37,422

* PC – oznacza że przekroczony został dopuszczalny czas wykonania

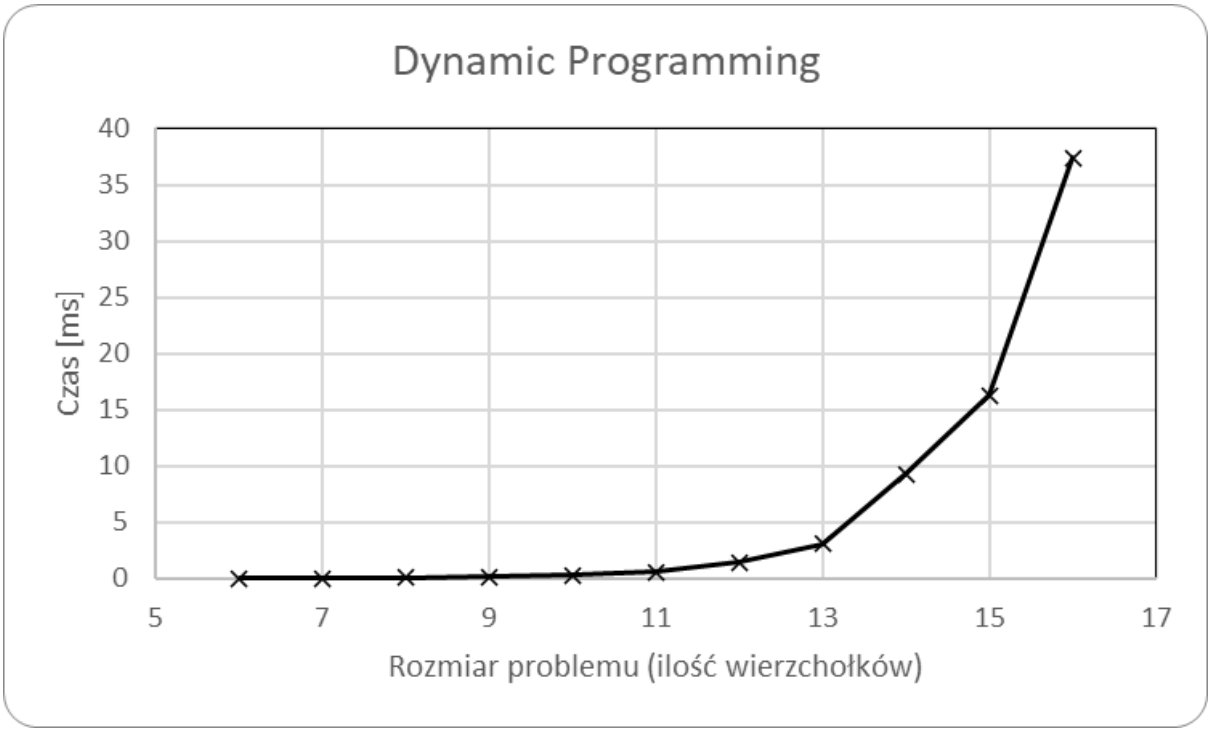
Wykres 1. Zależność czasu wykonania od rozmiaru problemu dla przeglądu zupełnego



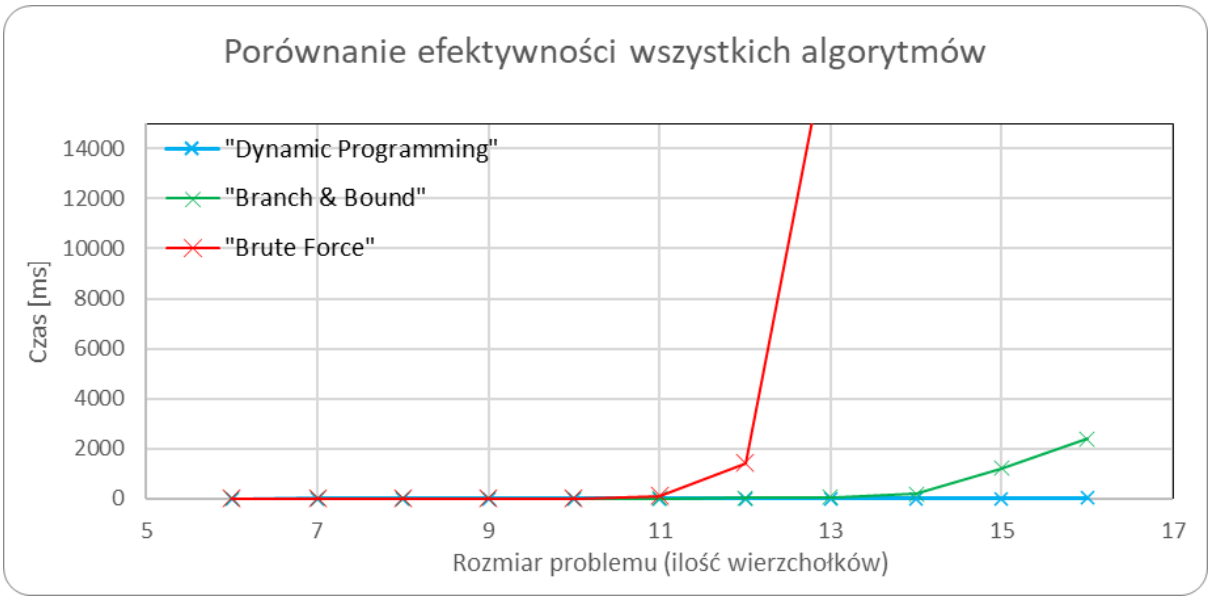
Wykres 2. Zależność czasu wykonania od rozmiaru problemu dla metody podziału i ograniczeń



Wykres 3. Zależność czasu wykonania od rozmiaru problemu dla programowania dynamicznego



Wykres 4. Ogólne zestawienie efektywności



4. Wnioski.

Analiza danych uzyskanych z przeprowadzonych pomiarów wykazuje zgodność wyników z klasami złożoności obliczeniowej, jakie zostały oszacowane we wstępie dla każdej z metod. Algorytm przeglądu zupełnego, implementowany w ramach eksperymentu, wykazał teoretyczną klasę obliczeniową $O(n!)$. Należy zaznaczyć, że pomiary zostały przerwane podczas zastosowania metody naiwnej dla instancji problemu, których wielkość przekroczyła 14 miast. To wynikało z nieakceptowalnie długiego czasu potrzebnego na znalezienie rozwiązania dla takich przypadków. Metoda naiwna zdaje się być skuteczna jedynie dla problemów o niewielkich rozmiarach.

Kolejny z implementowanych algorytmów opierał się na metodzie podziału i ograniczeń. Podobnie jak w przypadku przeglądu zupełnego, jego złożoność obliczeniowa jest ograniczona funkcją wykładniczą $n!$. Jednak dzięki zastosowaniu procedur rozgałęziania oraz ograniczania, czas potrzebny do znalezienia rozwiązania dla badanych instancji został znacznie zredukowany w porównaniu do metody naiwnej. Kluczowym elementem jest precyzyjne dostosowanie funkcji obliczającej dolne ograniczenie oraz skuteczne wykorzystanie procedur przeglądania możliwych rozwiązań. Niewłaściwy dobór funkcji ograniczającej może prowadzić do nieefektywnego odcinania gałęzi i skutkować czasem wykonania zbliżonym do przeglądu zupełnego.

Ostatni z zaimplementowanych algorytmów korzystał z metody programowania dynamicznego. W tym przypadku również zaobserwowano zgodność z teoretyczną klasą złożoności $O(n^2 * 2^n)$, osiąganą poprzez ograniczenie działania algorytmu do procedury wypełniania tablicy stanów. Ten algorytm okazał się najbardziej efektywny, a uzyskane czasy były znacznie niższe niż te osiągnięte w przypadku metody Branch & Bound.