Sprawozdanie

Projektowanie Efektywnych Algorytmów P

Etap 1

Opracował: Patryk Uzarowski

Informatyka Techniczna, Wydział Informatyki i Telekomunikacji

Termin oddania projektu: 10 listopada 2022

1. Wstęp

Celem pierwszego zadania projektowego było opracowanie algorytmów rozwiązujących deterministycznie problem Komiwojażera (Travelling Salesman Problem). Problem ten opiera się na znalezieniu najbardziej optymalnej drogi z i do wierzchołka początkowego poprzez wszystkie pozostałe wierzchołki. Problem polega więc na odnalezieniu cyklu Hamiltona o najmniejszej sumie wag krawędzi.
Założenia dotyczące problemu Komiwojażera:

a) Asymetryczny – drogi docelowe oraz powrotne nie muszą być sobie równe ((i => j) != (j => i)).
b) Droga z miasta (i => i) może być reprezentowana jako 0 / 9999 / -1.
c) Wierzchołek 0 jest punktem startowym.

Założenia dotyczące struktury projektu:

a) Program napisany jest w podejściu Obiektowym wykorzystując język C++.
b) Program wyposażony jest w Meny Główne, w którym można dokonać wyboru algorytmu, wczytać macierz sąsiedztwa rozpatrywanego grafu lub przeprowadzić test 100 instancji problemu dla podanej ilości wierzchołków.
c) Algorytmy egzekwowane są przez klasy:
   TSP_BF – realizacja algorytmu Przeglądu Zupełnego
   TSP_DP – realizacja algorytmu Programowania Dynamicznego
   TSP_BnB – realizacja algorytmu Podziału i Ograniczeń
d) W przypadku każdego algorytmu argumentem wejściowym jest macierz sąsiedztwa reprezentująca rozpatrywany graf np. :

wierzchołek	0	1	2	3
0	INF	3	4	7
1	4	INF	6	9
2	2	5	INF	3
3	8	1	6	INF

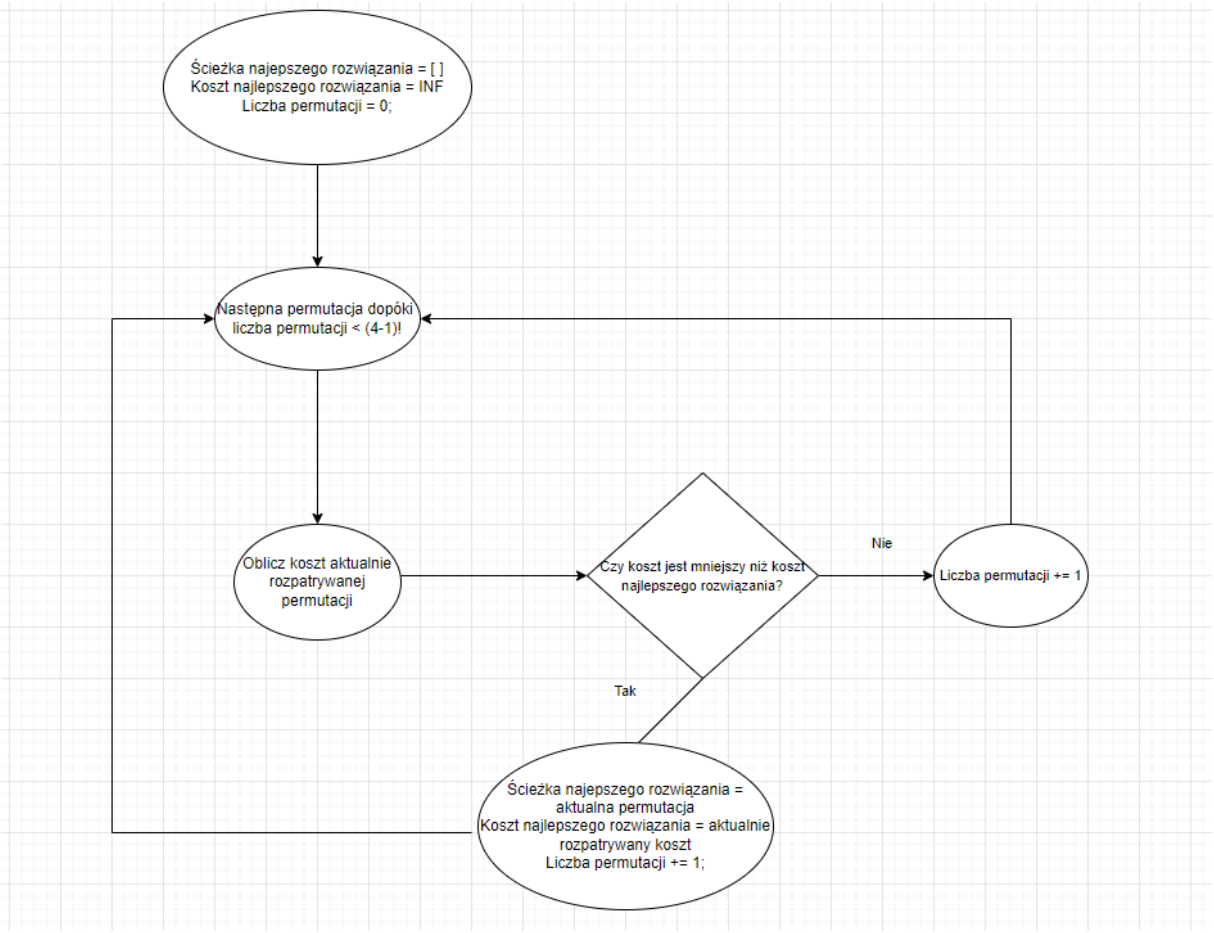
2. Przegląd Zupełny

Pierwszym algorytmem zaimplementowanym w zadaniu projektowym był algorytm Przeglądu zupełnego (Brute Force). Polega on na generacji (n-1!) permutacji wierzchołków (z węzłem startowym zawsze przodzie tablicy) grafu oraz dokonuje porównania aktualnie rozpatrzonego kosztu z tymczasowo najlepszym rozwiązaniem.
Złożoność obliczeniowa algorytmu to O((n-1)!).

Przykład praktyczny (Zostaje w nim wykorzystany graf z pkt 1.d))

Dla każdej permutacji wierzchołków zostaje obliczony koszt cyklu, który jeżeli jest mniejszy niż aktualnie najlepszy wynik, staje się najlepszym wynikiem.

Punktem wyjściowym algorytmy jest ustawienie najlepszego rozwiązania na maksymalnie dużą liczbę (inf).



Rysunek 1. Uproszczony schemat blokowy algorytmu Przeglądu Zupełnego dla problemu Komiwojażera

Tabela z rozwiązaniami bazująca na schemacie blokowym (Kolor zielony – Zmiana najlepszej ścieżki oraz rozwiązania na aktualnie rozpatrywane)

Liczba permutacji	Koszt najlepszego rozwiązania	Ścieżka najlepszego rozwiązania	Koszt rozpatrywanego rozwiązania	Ścieżka rozpatrywanego rozwiązania
0	-	-	20	0,1,2,3,0
1	20	0,1,2,3,0	20	0,1,3,2,0
2	20	0,1,2,3,0	26	0,2,1,3,0
3	20	0,1,2,3,0	12	0,2,3,1,0
4	12	0,2,3,1,0	16	0,3,1,2,0
5	12	0,2,3,1,0	22	0,3,2,1,0
Rozwiązanie algorytmu Przeglądu Zupełnego			12	0,2,3,1,0

3. Metoda Podziału i Ograniczeń

Algorytm opiera swe działanie na rozwijaniu węzłów drzewa przestrzeni stanów, gdzie każdy węzeł utożsamiany jest z danym wierzchołek grafu rozpatrywanego problemu Komiwojażera. Dziećmi każdego z nich są węzły reprezentujące możliwe połączenie z innymi wierzchołkami grafu (z wyłączeniem już przeanalizowanych). W zadaniu projektowym został zaimplementowany wariant algorytmu Best-first polegający na rozwijaniu tylko najlepszego węzła na danym poziomie przeszukiwań. Każdy węzeł algorytmu posiada swój koszt, który jest tutaj rozumiany jako [koszt rodzica + koszt krawędzi aktualnie rozpatrywanej oraz dolna granica ścieżki rozpoczynającej się w węźle. Dolna granica węzła jest rozumiana jako suma minimów wszystkich wierszy oraz kolumn aktualnie rozpatrywanej macierzy, która w następnych krokach będzie poddawana redukcji.

Złożoność obliczeniowa algorytmu to: Pesymistycznie O((n-1)!), Optymistycznie O($n^2 \ x \ 2^n$)

Przykład praktyczny (Zostaje w nim wykorzystany graf z pkt 1.d))

Kolejne kroki algorytmu:

Pierwszym etapem algorytmu jest określenie kosztu korzenia drzewa przestrzeni stanów (wierzchołka początkowego rozpatrywanego grafu), który z powodu braku rodzica będzie wyłącznie jego dolną granicą.

Kolorem zielonym zostaną oznaczone minima wierszy.

wierzchołek	0	1	2	3
0	INF	3	4	7
1	4	INF	6	9
2	2	5	INF	3
3	8	1	6	INF

Kolejnym etapem jest zredukowanie macierzy, będzie ona macierzą wyjściową dla wszystkich dzieci danego węzła. Poprzez zredukowanie rozumiemy tutaj odjęcie od wszystkich elementów danego wiersza wartości minimalnej.

wierzchołek	0	1	2	3
0	INF	0	1	4
1	0	INF	2	5
2	0	3	INF	1
3	7	0	5	INF

Następnie obliczamy minima kolumn oraz ponownie redukujemy macierz.

Kolorem niebieskim zostały oznaczone minima kolumn.

wierzchołek	0	1	2	3
0	INF	0	0	3
1	0	INF	1	4
2	0	3	INF	0
3	7	0	4	INF

Dolna granica (a zarazem koszt korzenia) wynosi 3 + 4 + 2 + 1 + 1 + 1 = 12

Kolejnym krokiem algorytmu przy założeniu przeglądu wszerz, jest sprawdzenie wszystkich możliwych węzłów reprezentujących wierzchołki grafu, gdzie przejście jest możliwe. W naszym przypadku są to wierzchołki 1,2,3.

a) Połączenie 0 -> 1
   Pierwszym elementem rozważanego połączenia będzie wykreślenie wszystkich dróg wychodzących z wierzchołka 0 oraz dróg docierających do wierzchołka 1.
   W przypadku rozważanego zadania projektowego poprzez wykreślenie rozumiane jest wstawienie dużej wartości w kolumnach / wierszach (INF). Dodatkowo wykreślane jest połączenie powrotne (1 => 0). Wszystkie operacje dokonywane są na macierzy zredukowanej odziedziczonej po węźle rodzicu.
   Kolorem czerwonym zaznaczono elementy do wykreślenia.

wierzchołek	0	1	2	3
0	INF	0	0	3
1	0	INF	1	4
2	0	3	INF	0
3	7	0	4	INF

wierzchołek	0	1	2	3
0	INF	INF	INF	INF
1	INF	INF	1	4
2	0	INF	INF	0
3	7	INF	4	INF

   Na podstawie nowo otrzymanej macierzy dokonujemy redukcji oraz obliczenia dolnej granicy (Podejście jak w poprzednim kroku)

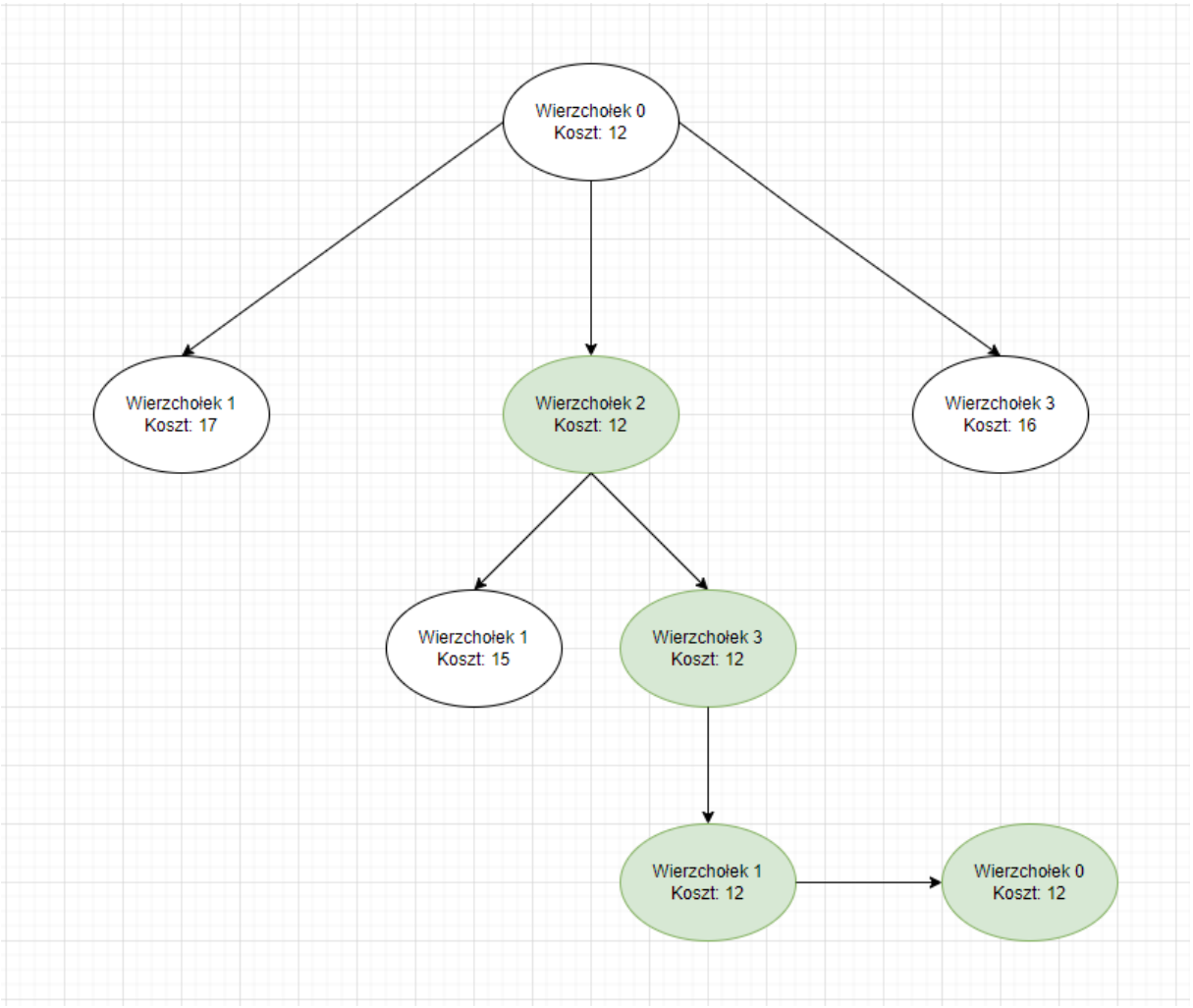
wierzchołek	0	1	2	3
0	INF	INF	INF	INF
1	INF	INF	1	4
2	0	INF	INF	0
3	7	INF	4	INF

wierzchołek	0	1	2	3
0	INF	INF	INF	INF
1	INF	INF	0	3
2	0	INF	INF	0
3	3	INF	0	INF

   Dolna granica = 5
   Koszt węzła obliczamy z => koszt węzła 0 + koszt przejścia (0 => 1) + dolna granica węzła 1.
   Koszt węzła(1) wynosi 12 + 0 + 5 = 17

Algorytm postępuje w jednakowy sposób dla kolejnych węzłów w przypadku zakończenia przeszukiwania danej głębokości wybierany zostaje najlepszy węzeł (cechujący się najniższym kosztem) oraz zostaje on rodzicem kolejnych możliwych połączeń. Węzły dzieci dziedziczą po rodzicu jego aktualną macierz redukcji.

Rozwiązanie dla przykładu z punktu 1d.



Ścieżka 0,2,3,1,0. Wynik algorytmu: 12

4. Programowanie Dynamiczne

Algorytm programowania dynamicznego opiera się na podzieleniu problemu Komiwojażera na mniejsze pod-problemy rozumiane tutaj jako obliczone wartości ścieżek od dołu grafu rozwijające rozwiązanie w stronę korzenia. Obliczone koszty są zapamiętywane w pomocniczych macierzach, przez co jednakowy problem jest liczony maksymalnie jeden raz. Algorytm został zaimplementowany w postaci rekurencyjnej, z kolei przejścia do kolejnych wierzchołków grafu są sprawdzane oraz manipulowane na podstawie maski bitowej.

Złożoność obliczeniowa algorytmu to: O($n^2\ x\ 2^n$)

Maski bitowe służą nam w celu określenia aktualnego stanu całkowitej ścieżki dla danego pod-problemu. Dla przedstawionego powyżej grafu posiadającego 4 wierzchołki maksymalna maska bitowa wynosi 1111 [b] co jest równoważne z zakończeniem cyklu oraz rozwiązaniem danego zbioru.
Interpretacja masek:

Maska	0001	0011	0111	1111
Wierzchołki rozpatrzone	0	0,1	0,1,2	0,1,2,3

Sprawdzanie ścieżek oraz manipulacja trasami została zaimplementowana poprzez operacje logiczne (AND i OR) wykonywane na aktualnie rozpatrywanej masce.
Przykład:

Przejście	Maska	Operacja logiczna	Cel	Maska wynikowa
0 -> 1	0001	OR	0010	0011
1 -> 3	0011	OR	1000	1011
3 -> 2	1011	OR	0100	1111

Przykład praktyczny (Zostaje w nim wykorzystany graf z pkt 1.d))
Kolejne kroki algorytmu:

1. Stworzenie macierzy pomocniczej przechowującej rozwiązania konkretnych pod-problemów (połączeń wierzchołków)
2. Jeżeli maska bitowa jest równa $2^n - 1\ [b]$, gdzie n – liczba wierzchołków rozpatrywanego grafu, to zwracamy połączenie z aktualnej pozycji do wierzchołka startowego.
3. Jeżeli algorytm został już wywołany dla danych parametrów zwracamy wartość przechowywaną w macierzy pomocniczej.
4. Dla wszystkich przejść (liczba wierzchołków) sprawdzamy czy możliwe jest przejście, dokonujemy tego poprzez operację AND na aktualnej masce oraz wierzchołku docelowym (reprezentowanym jako 2^x [b] gdzie x – numer wierzchołka docelowego) np.

Maska	Potencjalne przejście	Reprezentacja przejścia jako przesunięcia bitowego	Porównanie (AND logiczny) musi być równe 0	Czy przejście możliwe?
0001	0 -> 3	2^3 = 1000	0001 & 1000	= 0, Tak
1011	3 -> 1	2^1 = 0010	1011 & 0010	= 1, Nie

5. Jeżeli przejście jest możliwe wykonujemy rekurencyjnie wszystkie pod-problemy dla danej permutacji wierzchołków i otrzymujemy koszt dróg wierzchołków, które znajdują się w zbiorze pod-problemu.

6. Wartość porównujemy z aktualnie najlepszą dla danego zbioru i jeżeli jest mniejsza to ją aktualizujemy.
7. Końcowo zwracamy najmniejszy wynik dla najbardziej zewnętrznego pod-problemu – dla zbioru wierzchołków wychodzących z korzenia drzewa.

Rozwiązanie dla grafu 1d.

1. Rozruch algorytmu:

	0	1	2	3
0				
1	4			
2	2			
3	8			

2. Pod-problem ze zbiorem 1-elementowym

	0	1	2	3
0				
1	4	-	8	17
2	2	9	-	11
3	8	5	8	-

3. Pod-problem ze zbiorem 2-elementowym

	0	1	2	3	2,1	3,1	3,2
0							
1	4	-	8	17			17
2	2	9	-	11		8	
3	8	5	8	-	15		

4. Najbardziej zewnętrzny pod-problem, zbiór 3 elementowy

	0	1	2	3	2,1	3,1	3,2	2,3,1
0								12
1	4	-	8	17			17	
2	2	9	-	11		8		
3	8	5	8	-	15			

Wynik algorytmu: 12.

8. Ścieżka zostaje wypisana poprzez przejście macierzy pomocniczej przechowującej rozwiązania problemu z uwzględnieniem ścieżek poprzedzających dane obliczenie.
   a) Znajdując się w wierzchołku startowym rozważamy pod-problem ze zbiorem {1,2,3}
      Tymczasowy wynik znajduje się w macierzy pomocniczej tab[0011][1], Kolejny w tab[0101][2], następny w tab[1001][3]. Do każdego wyniku dodajemy ścieżkę poprzedzającą dany krok (w tym przypadku 0). Znajdujemy najmniejszy i zapisujemy jego numer indeksu tab[][i].
   b) Znajdując się w wierzchołku 2 rozważamy pod-problem ze zbiorem {1,3}
      Tymczasowy wynik znajduje się w macierzy tab[0111][1], kolejny w tab[1101][3]. Do każdego wyniku dodajemy ścieżkę poprzedzającą dany krok

(0 => 2). Znajdujemy najmniejszy i zapisujemy jego numer indeksu tab[][i].

c) Znajdując się w wierzchołku 3 rozważamy pod-problem ze zbiorem {1}. Zapisujemy ostatni wierzchołek wraz z wierzchołkiem startowym.
Wynik algorytmu: Ścieżka 0,2,3,1,0.

5. Przebieg eksperymentu

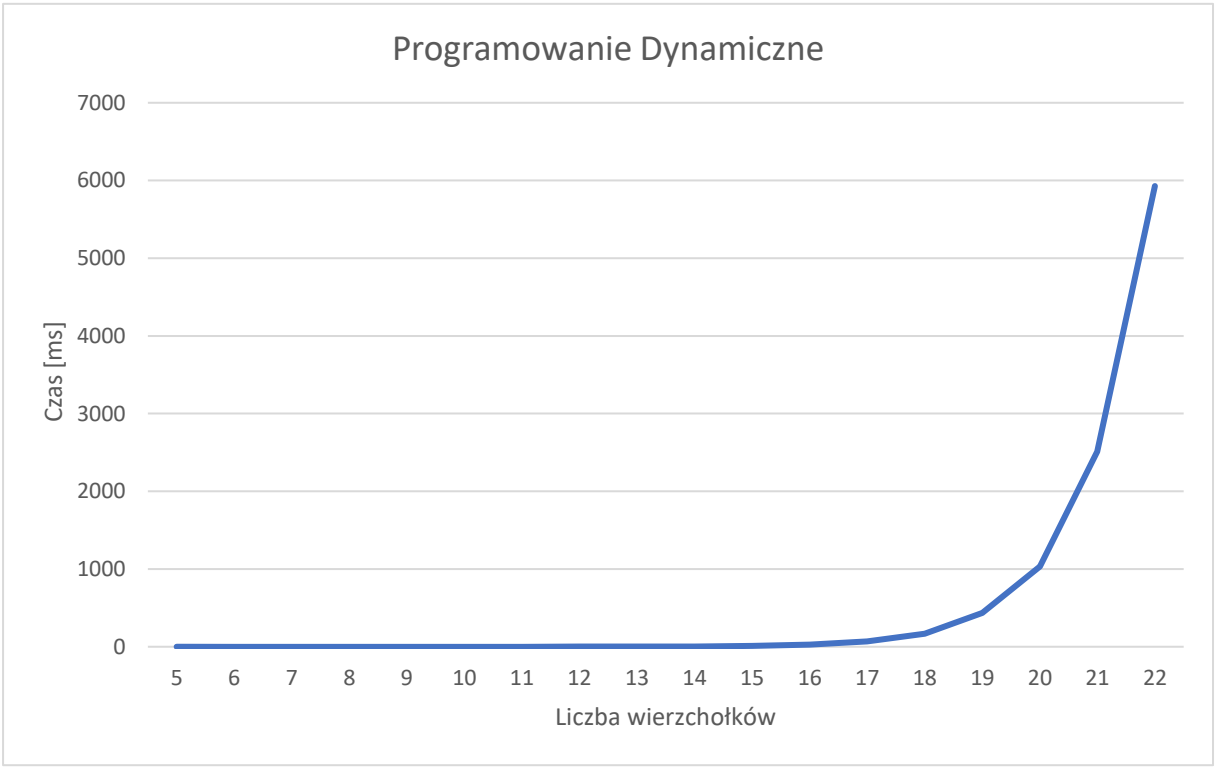
W eksperymencie przetestowano działanie algorytmów na grafach o rozmiarach od 5 do 22 wierzchołków. Wagi krawędzi zostały wygenerowane losowo z przedziału [0:100]. Pomiar czasu przebiegu algorytmów został zmierzony za pomocą std::chrono::steady_clock, wyniki w poniższej tabeli stanowią wartość uśrednioną czasu ze 100 instancji problemu dla każdego grafu o **N** wierzchołkach.

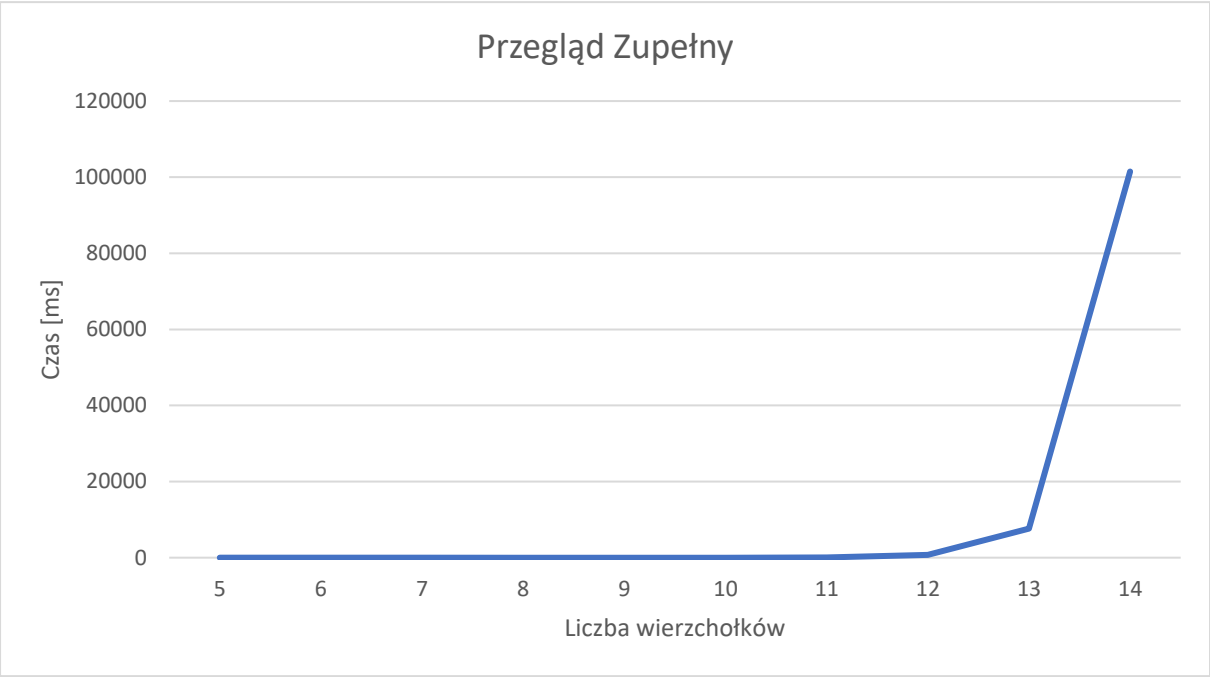
Wyniki pomiarów

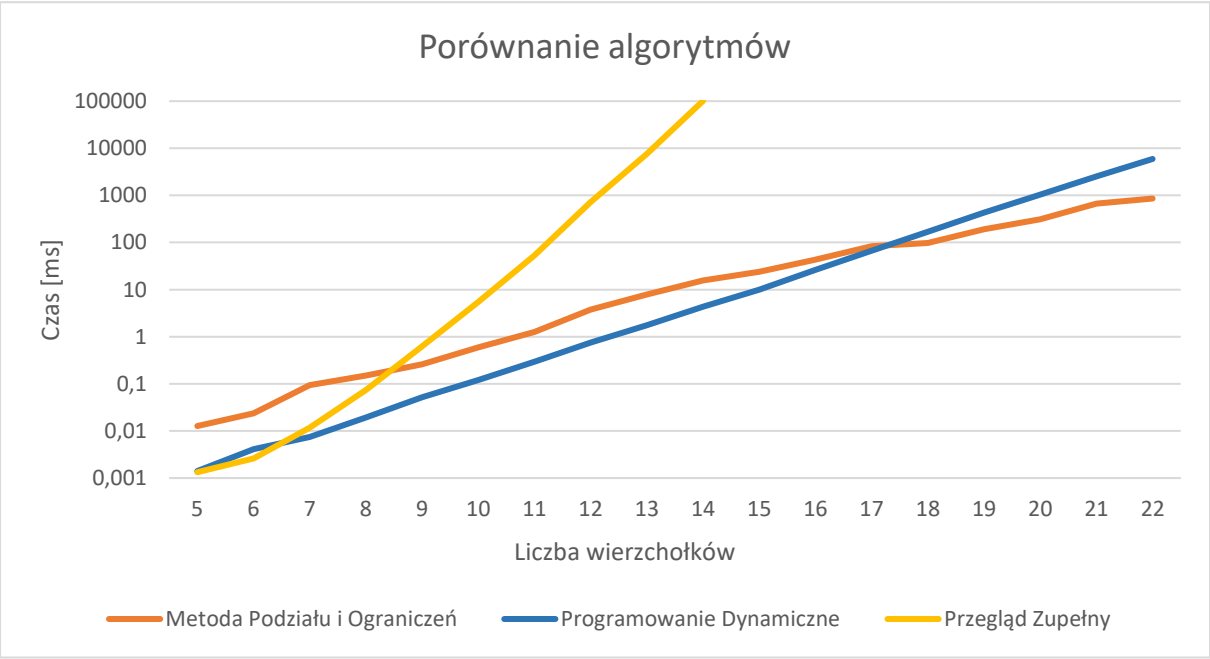
	Czas [ms]		
N	Przegląd zupełny	Metoda Podziału i ograniczeń	Programowanie Dynamiczne
5	0,00133	0,0127	0,00141
6	0,00259	0,0235	0,00413
7	0,0117	0,094	0,00747
8	0,07447	0,1507	0,0192
9	0,620	0,26	0,0521
10	5,504	0,594	0,121
11	53,331	1,254	0,296
12	724,811	3,759	0,750
13	7594	7,9	1,749
14	101486	15,723	4,343
15	-	23,943	9,870
16	-	43,563	26,013
17	-	84,024	66,80
18	-	96,937	168,901
19	-	190,064	434,323
20	-	312,512	1031,25
21	-	665,923	2510,53
22	-	857,67	5928,386

Wykresy:


Metoda Podziału i Ograniczeń
1000
900
800
700
600
500
400
300
200
100
0
Czas [ms]
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Liczba wierzchołków


Programowanie Dynamiczne
7000
6000
5000
4000
3000
2000
1000
0
Czas [ms]
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Liczba wierzchołków





6. Wnioski

Przegląd zupełny relatywnie szybko pokazuje swoje ograniczenia względem większych instancji problemu TSP. Stała złożoność czasowa algorytmu O((n-1)!) jest dobrze zauważalna już na etapie porównań dla instancji problemu 9-10 wierzchołków, kiedy to czas realizacji Przeglądu Zupełnego zaczyna diametralnie rosnąć. Metoda podziału i ograniczeń jest algorytmem, który jest bardzo trudny do uśrednienia, przy podejściu Best-first czas egzekucji algorytmu jest silnie uzależniony od wylosowanych wartości wag krawędzi rozpatrywanego grafu. W przypadku grafu umożliwiającego sukcesywne redukcje macierzy węzłów drzewa przestrzeni stanów, algorytm wykonywał się zdecydowanie szybciej niż

Programowanie Dynamiczne. Jednak w przypadku grafów trudnych do redukcji złożoność czasowa algorytmu w skrajnie pesymistycznym przypadku może być nawet porównywalna do Przeglądu Zupełnego O((n-1)!). Programowanie Dynamiczne wykazywało się znacznie stabilniejszym rozkładem wyników jednak w przypadku dużych instancji czas wykonania algorytmu spadał względem Brounch and Bound. Należy tutaj jednak uwzględnić duże rozbieżności wynikające z charakterystyki algorytmu Podziału i Ograniczeń opisanej powyżej.