Politechnika Wrocławska
26.11.2019r

Projektowanie Efektywnych Algorytmów

Temat:
Rozwiązanie TSP za pomocą metod przeglądu zupełnego, programowania dynamicznego, metodą podziału i ograniczeń i metodą zachłanną

Prowadzący
dr inż. Tomasz Kapłon

Autor
Paweł Parczyk

Wstęp

Początki problemu Komiwojażera (TSP - Travelling salesman problem) sięgają XIX wieku, kiedy to William Rowan Hamilton opisał problem istnienia cyklu n-elementowego w grafie o n wierzchołkach. Za pierwszego autora, który sformalizował matematycznie problem komiwojażera, uznaje się Karla Mengera, który w 1930 zwrócił szczególną uwagę na trudność w obliczeniu optymalnego rozwiązania. Z uwagi na prostotę opisu problemu i trudność optymalizacji algorytmów służących do jego rozwiązania, problem stał się bardzo popularny.

Celem projektu było stworzenie kilku algorytmów rozwiązujących TSP, a następnie określenie ich złożoności czasowych i pamięciowych na podstawie przeprowadzonych testów, a także porównanie różnych podejść pod względem szybkości, zapotrzebowania zasobów i skuteczności wyszukiwania optymalnego rozwiązania.

W ramach projektu zostały zaimplementowane cztery algorytmy: metoda przeszukiwania zupełnego (brute force), metoda programowania dynamicznego (algorytm Helda-Karpa), metoda zachłanna oraz metoda B&B.

Program został napisany w języku c++ w wersji 11 z wykorzystaniem technik programowania obiektowego. Kod został skompilowany przez kompilator g++, a następnie uruchomiony na komputerze zarządzanym przez system Linux Ubuntu 16.04, sama maszyna wyposażona była w 8GB pamięci ram i procesor intel core i5.

Algorytmy zostały napisane w wersjach iteracyjnych w celu zminimalizowania czasu wykonania.

Realizacja i testy

Wszystkie algorytmy zostały przetestowane dla 8 grafów: 6-, 8-, 10-, 12-, 11-, 13-, 14 i 15-wierzchołkowego. Każdy pomiar został powtórzony 3 razy a wyniki zostały uśrednione. Poprawność algorytmów została określona na podstawie znanych odpowiedzi podanych w pliku *wyniki.as.*

Grafy i optymalne rozwiązania do nich zostały pobrane ze strony dr inż. Jarosława Mierzwy.

Znalezione rozwiązania:

Dla grafów dr Mierzwy zostały poniżej zapisane uzyskane rozwiązania

- tsp_6: 0 – 1 - 2 – 3 - 4 – 5 – 0 dystans: 132,
- tsp_6_2: 0 – 5 – 1 – 2 – 3 – 4 – 0 dystans: 80,
- tsp_10: 0 – 3 – 1 – 9 – 6 – 7 – 8 – 2 – 4 – 5 – 0 dystans: 212,
- tsp_12: 0 – 1 – 8 – 4 – 6 – 2 – 11 – 9 – 7 – 5 – 3 – 10 – 0 dystans: 264,
- tsp_14: 0 – 12 – 1 – 8 – 4 – 6 – 2 – 11 – 13 – 9 – 7 – 5 – 3 – 10 – 0 dystans: 282,
- tsp_15: 0 – 12 – 1 – 14 – 8 – 4 – 6 – 2 – 11 – 13 – 9 – 7 – 5 – 3 – 10 – 0 dystans: 291.

Opis algorytmów

1. Przegląd zupełny

Metoda polegająca na sprawdzeniu wszystkich możliwych permutacji wierzchołków w grafie i wyboru takiej, w której sumaryczny koszt odwiedzenia wszystkich wierzchołków jest najmniejszy. Metoda ta gwarantuje znalezienie optymalnego rozwiązania.

Ilość przypadków do rozpatrzenia wyraża się wzorem n!. To sprawia że dla odpowiednio dużych grafów (w zależności od możliwości sprzętu) algorytm ten staje się bezużyteczny, a problemy nierozwiązywalne.

Złożoność obliczeniowa algorytmu, z uwagi na ilość przypadków do rozpatrzenia, wynosi O(n!)

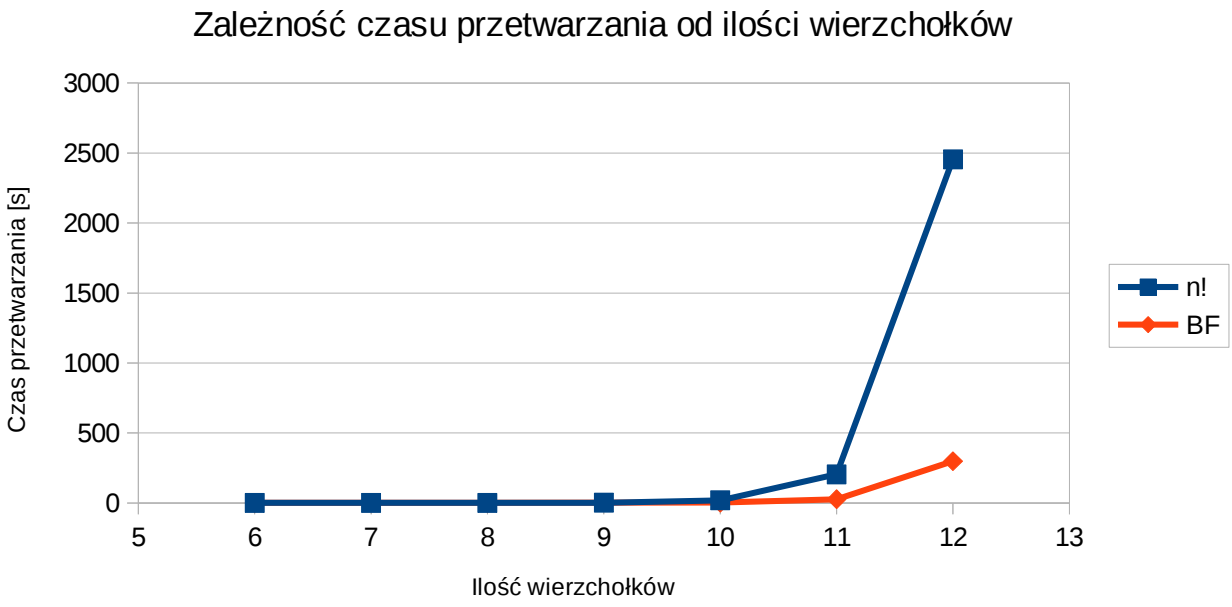
Algorytm:

W kolejnych iteracjach generowane są kolejne permutacje. Po obliczeniu długości ścieżki wynikającej z danej permutacji wierzchołków sprawdzana jest ona z aktualnie zapamiętaną najkrótszą ścieżką. Jeżeli aktualna jest krótsza to aktualna ścieżka zostaje zapamiętana.

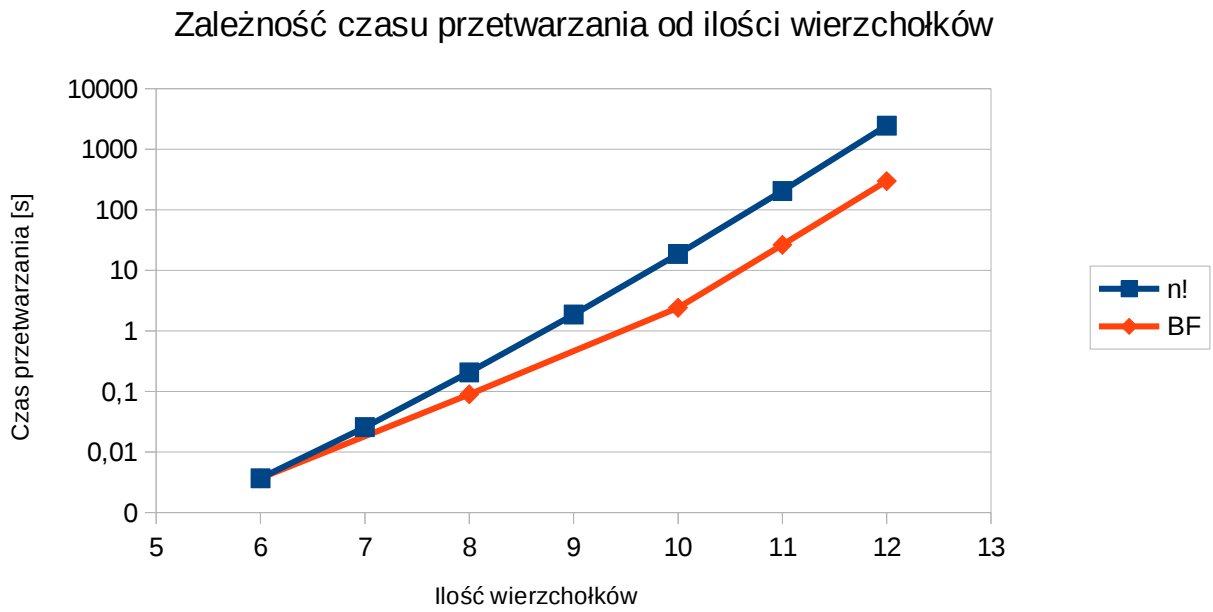
Realizacja:

Otrzymane wyniki czasowe pracy algorytmu:

Ilość wierzchołków	Przegląd zupełny [ms]
6	3.69
8	4.06
10	2406.84
12	298282.58
13	38265241.74
14	>15h
15	>226h



Wykres prezentuje porównanie uzyskanej złożoności czasowej algorytmu przeglądu zupełnego z zakładaną złożonością. Z wykresu wynika że zaimplementowany algorytm ma złożoność mniejszą niż O(n!). Poniżej ten sam wykres został zaprezentowany w skali logarytmicznej.



2. Programowanie dynamiczne

Metoda programowania dynamicznego cechuje się mniejszą złożonością czasową od metody przeglądu zupełnego, za to znacznie wyższą złożonością pamięciową. Polega ona na podzieleniu całego problemu na podproblemy i zapamiętywaniu wszystkich rozwiązanych podproblemów w celu późniejszego wykorzystania. Każdy podproblem może składać się z innych.

Algorytm:

Zastosowano algorytm Helda-Karpa, który opiera się na kolejnych wywoływaniach funkcji *H(V, d)*, gdzie *V* to zbiór wierzchołków $V=\{v_1, v_2, \ldots v_n\}$, przez które trzeba dojść, a d $d \in s$ to

wierzchołek docelowy, na którym należy zakończyć. Następnie po obliczeniu wartości funkcji jej wynik zostaje zapisany w tablicy kosztów. W sytuacji, gdy ponownie zachodzi potrzeba wyliczenia wartości funkcji H, to korzysta się z tablicy kosztów.

Działanie algorytmu określa wzór rekurencyjny:
$H(\{v_1, v_2, \ldots, v_n\}, V_0) = min\{H(\{v_2, v_3, \ldots v_n\}, v_1) + D(v_{1,} V_0), H(\{v_1, v_3, \ldots v_n\}, v_2) + D(v_{2,} v_0), \ldots H(\{v_1, v_2, \ldots v_{(n-1)}\}, v_n) + D(v_n, v_0)\}$,

gdzie $H(\{v_i\}, v_i) = D(v_{0,} v_i)$, a $D(v_i, v_j)$ to koszt dotarcia z wierzchołka v_i do wierzchołka v_j .

W programie wektor wierzchołków został zaprezentowany jako liczba Int, gdzie i-ty bit w tej liczbie oznacza czy i-ty+1 wierzchołek grafu zawiera się w V.

Przykład: *H(v, d)*, gdzie $s = (bin)0...000100101, d = (bin)0.0000000100$ oznacza, że należy policzyć koszt dotarcia z wierzchołka 0 do wierzchołka 3 (trzeci bit ustawiony na jeden) przechodząc przez wierzchołki 1, 3, 6. Zapis tak obliczonego kosztu odbywa się do tablicy kosztów *k[v][d]*.

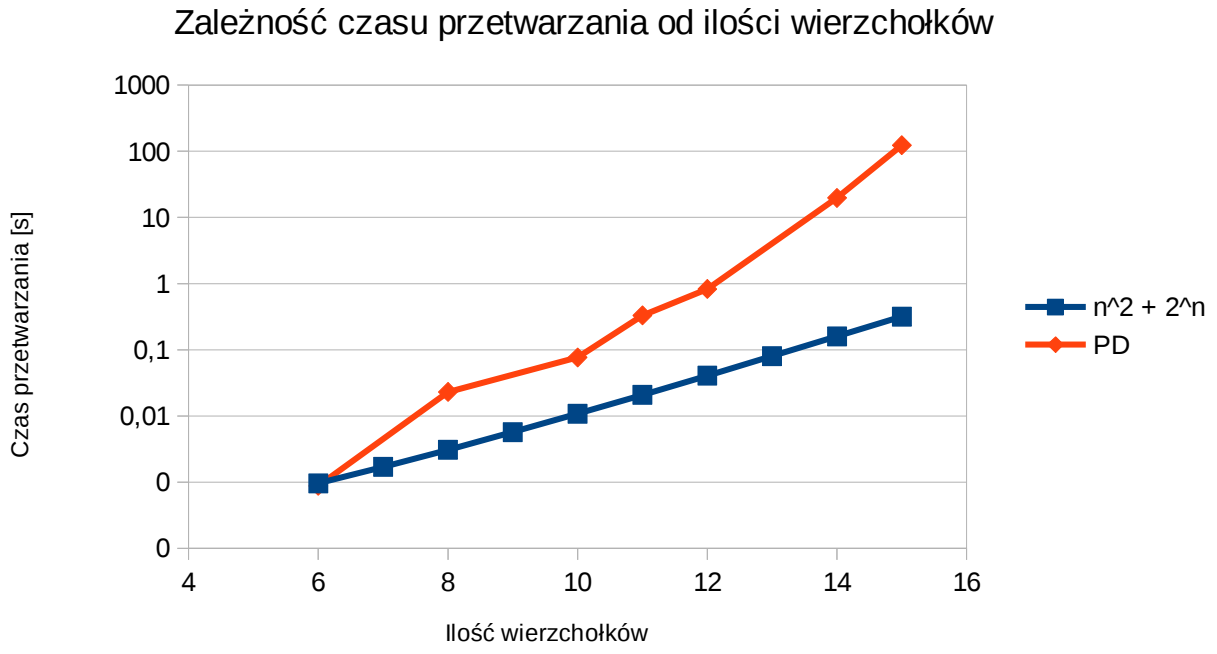
Rozpatrując funkcje H od przypadków podstawowych tzn *rozmiar(V) = 1* można uzyskać algorytm w wersji iteracyjnej.
Złożoność obliczeniowa algorytmu jest uzależniona od zastosowanej metody indeksowania rozwiązań zapamiętywanych w tablicy kosztów. Przy zastosowaniu indeksowania o złożoności O(1) – złożoność równa dostępowi do tablicy przez indeks – złożoność obliczeniowa algorytm Helda-Karpa wynosi $O(n^2 + 2^n)$. Znaczącą wada tego algorytmu jest zapotrzebowanie pamięciowe, które rośnie wraz z powiększaniem się problemu. Złożoność pamięciowa wynosi $O(n \cdot 2^n)$. Cecha ta nie stanowi problemu dopóki pamięci w komputerze wystarcza, ale gdy pamięć się kończy system operacyjny zaczyna stosować pamięć wymiany co spowoduje zwolnienie realizacji.

Realizacja:

Otrzymane wyniki czasowe pracy algorytmu:

Ilość wierzchołków	**Dynamiczne [ms]**
6	0.879
8	61.51
10	76.65
12	829.55
13	3581.26
14	19745.35
15	286082.53

Na wykresie porównano uzyskane wyniki czasowe z wynikami funkcji złożoności obliczeniowej algorytmu. Zaprojektowany algorytm okazał się być nieoptymalny a jego złożoność jest wyższa od $O(n^2+2^n)$.



Algorytm Helda-Karpa z uwagi na swoją złożoność pamięciową, na maszynie testowej, poprawnie działał (wystarczało pamięci operacyjnej) do grafu o 15 wierzchołkach. Powyżej tej wartości było potrzebne znacznie więcej pamięci, niż było zainstalowane na komputerze, co powodowało wyłączenie się programu.

3. Metoda zachłanna

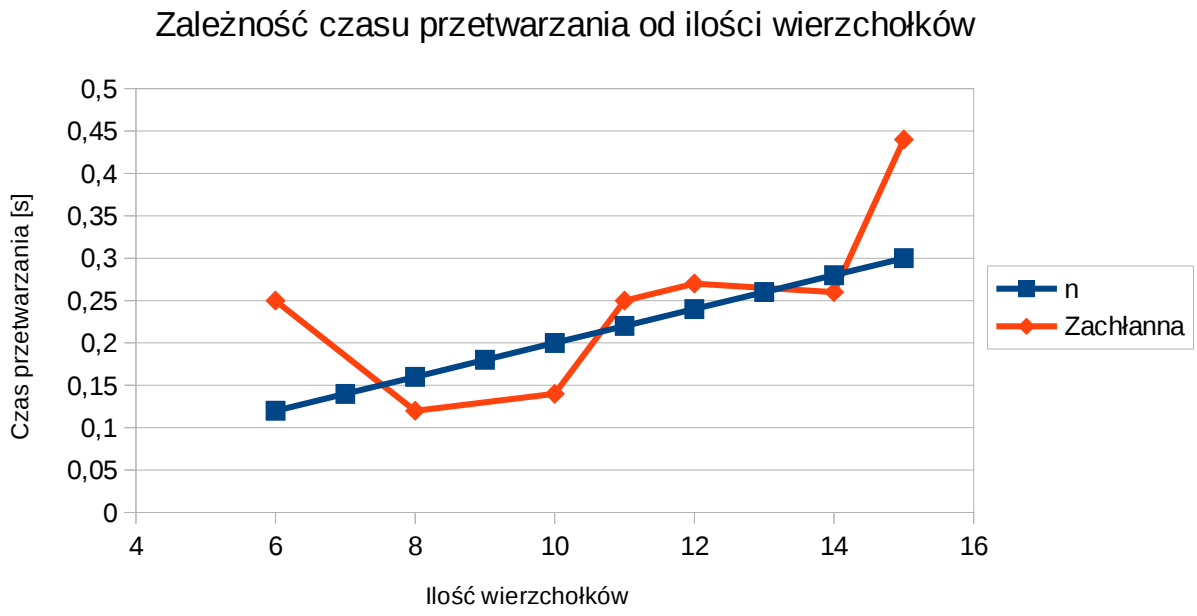
Metoda zachłanna, polega na wybieraniu przy każdym kroku bieżącego najlepszego rozwiązania.

Algorytm:

Zastosowano algorytm najbliższego sąsiada, algorytm ten ma liniową złożoność obliczeniową O(n), ale nie gwarantuje znalezienia optymalnego rozwiązania. Polega na przechodzeniu z wierzchołka zawsze do tego wierzchołka, do którego koszt przejścia jest najniższy

Realizacja:

Ilość wierzchołków	Metoda zachłanna [ms]
6	0.15
8	0.10
10	0.25
12	0.26
13	0.29
14	0.44
15	0.17



Na podstawie wykresu można stwierdzić, że czas realizacji algorytmu jest w znacznym stopniu losowy, ale ogólna tendencja jest liniowa.

4. B&B

Metoda przeglądu z ograniczeniami jest modyfikacją metody przeglądu zupełnego. Z uwagi na pewne zależności pomiędzy kolejnymi rozwiązaniami można próbować wcześniej podzielić ścieżki dojścia do rozwiązania na takie, które dają nadzieję uzyskania lepszego rezultatu od znalezionego wcześniej i na takie, które nie dają takiej nadziej. Aby móc zastosować tą metodę trzeba znaleźć w problemie zależności.

Algorytm: W procesie występują dwa ograniczenia górne i dolne. Górna granica jest obliczana jako koszt drogi przebytej do aktualnie przetwarzania wierzchołka. Dolna granica to z kolei najbardziej optymistyczna droga potrzebna do osiągnięcia wierzchołka przeznaczenia.

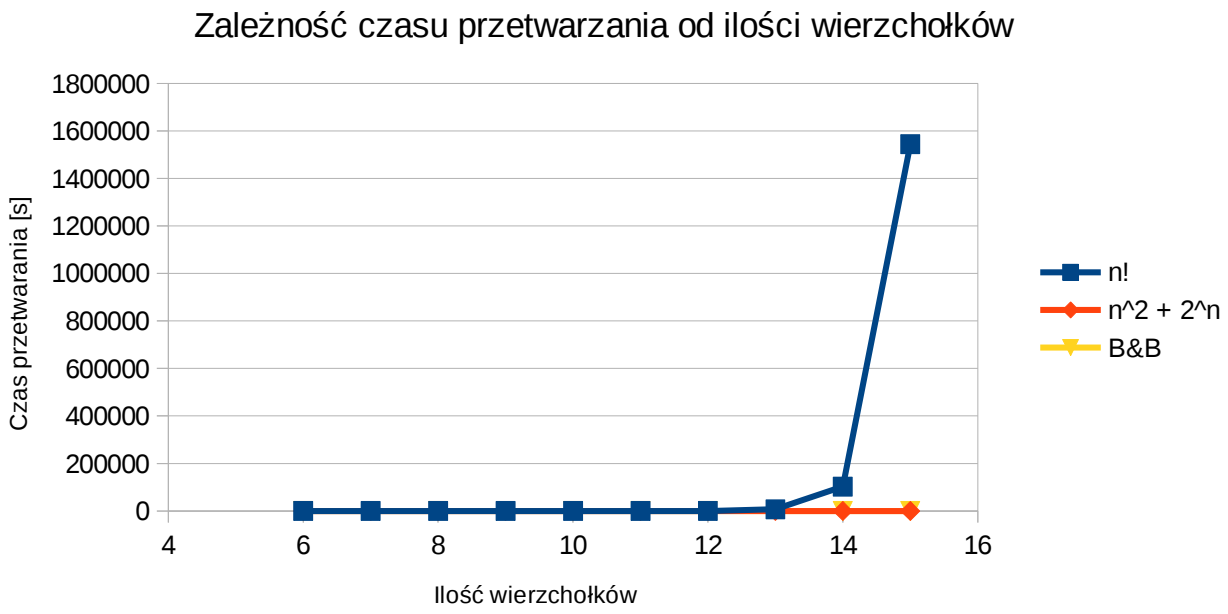
Podczas decyzji o rozwinięciu kolejnego wierzchołka sprawdzane są ograniczenia górne i dolne. Górne porównywane jest z kosztem dotarcia do kolejnego wierzchołka, i jeżeli ten koszt będzie większy od aktualnego minimalnego kosztu, to wierzchołek taki nie będzie rozwinięty. Podobnie badane jest ograniczenie dolne. Jeżeli koszt dotarcia do następnego wierzchołka, a następnie koszt dotarcia z niego do końca jest wyższy niż aktualnie najlepsze rozwiązanie, to wierzchołek taki nie zostanie rozwinięty.

Nie można jednoznacznie określić złożoności obliczeniowej z uwagi na dużą zależność czasu wykonania od rodzaju problemu. W najgorszym przypadki metoda przeglądu z ograniczeniami sprowadza się do przeglądu zupełnego. Pesymistyczna złożoność wynosi O(n!)

Realizacja:

Otrzymane wyniki czasowe pracy algorytmu:

Ilość wierzchołków	**B&B [ms]**
6	0.68
8	0.88
10	99.60
12	1864.80
13	3310.59
14	22450.10
15	62526.20

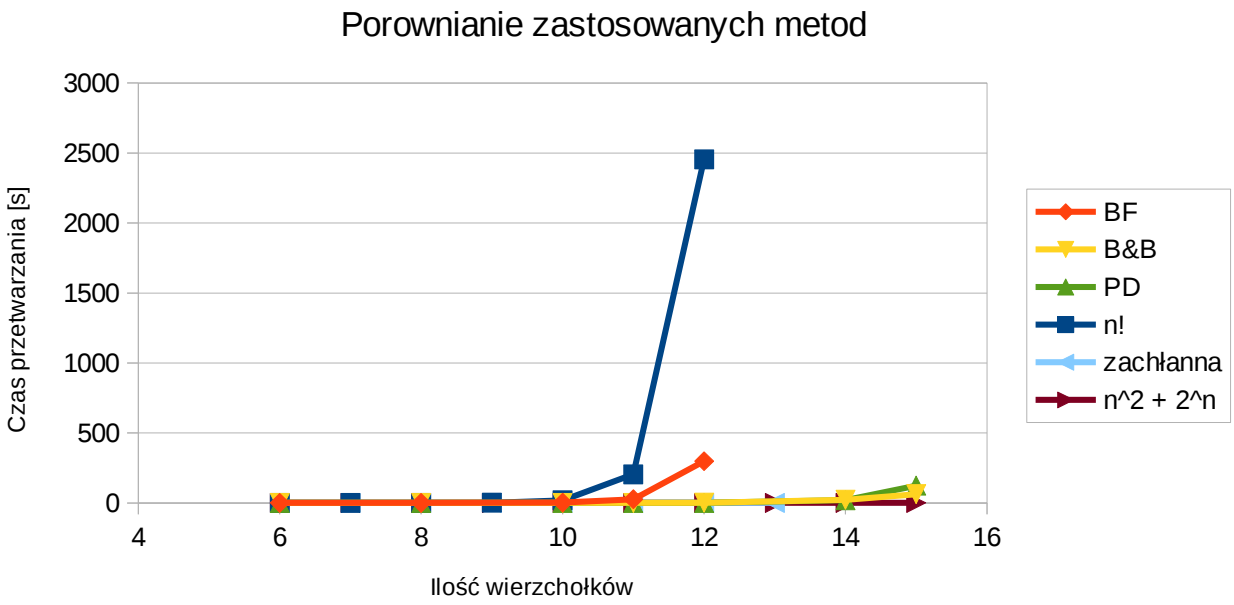


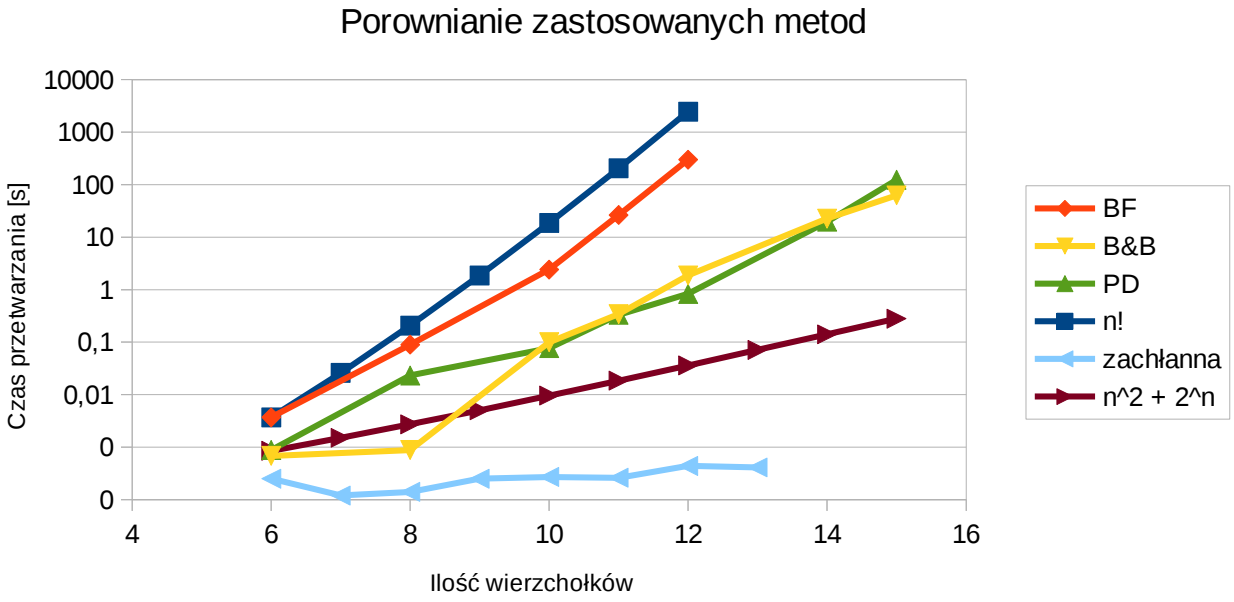
Z uwagi na trudność określenia złożoności obliczeniowej metody algorytmu przeglądu można jedynie określić że mieści się ona pomiędzy O(n!) a $O(n^2+2^n)$. W zależności od określonych ograniczeń można uzyskiwać różne rezultaty. Na czas realizacji algorytmu ma również wpływ układ problemu jaki trzeba rozwiązać.

Porównanie

Poniżej została zaprezentowana tabela prezentująca porównanie czasu realzacji różnych algorytmów

Ilość wierzchołków	Przegląd zupełny [ms]	Dynamiczne [ms]	Metoda zachłanna [ms]	B&B [ms]
6	3.69	0.879	0.15	0.68
8	4.06	61.51	0.10	0.88
10	2406.84	76.65	0.25	99.60
12	298282.58	829.55	0.26	1864.80
13	38265241.74	3581.26	0.29	3310.59
14	>15h	19745.35	0.44	22450.10
15	>226h	286082.53	0.17	62526.20





Wykres prezentuje porównanie wszystkich zastosowanych algorytmów ze sobą. Wszystkie 3 algorytmy cechują się złożonością obliczeniową pomiędzy O(n!) a $O(n^2+2^n)$. Można zauważyć że implementacje B&B i PD cechują się podobnym złożonościami czasowymi, jednak w B&B można zaobserwować wpływ losowości związanej z rozpatrywanymi problemami na czas znajdowania rozwiązania. Algorytm zastosowany w metodzie zachłannej okazał się być najszybszym ze wszystkich algorytmów, ale nie gwarantuje otrzymania właściwego rozwiązania.

Wnioski

1. W projekcie udało się zaimplementować algorytm o optymalnej złożoności obliczeniowej.
2. Implementacja algorytmu Helda-Karpa okazała się być nieoptymalna pod względem czasowym – algorytm ten pracował podobnie do B&B – a także nieoptymalna pod względem pamięciowym, gdyż już 15-wierzchołkowy graf okazał się być za duży dla maszyny na której odbywały się testy.
3. Algorytm zastosowany w metodzie zachłannej rozwiązuje zadane problemy w czasie znacznie krótszym od pozostałych algorytmów ale przy tym nie gwarantuje uzyskania poprawnego wyniku. Dla niektórych problemów może to nie być problem, więc ten algorytm również może znaleźć zastosowanie.
4. Podsumowując złożoności czasowe i pamięciowe zaimplementowanych algorytmów najlepszy, który znajduje poprawne rozwiązanie okazał się być algorytm przeglądu z ograniczeniami.