Politechnika Wrocławska
Wydział Elektroniki

Wrocław, dn.09.11.2020r.

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

ZADANIE PROJEKTOWE NR.1

IMPLEMENTACJA I ANALIZA EFEKTYWNOŚCI ALGORYTMU PRZESZUKIWANIA ZUPEŁNEGO ORAZ ALGORYTMÓW OPARTYCH NA METODZIE PODZIAŁU I OGRANICZEŃ ORAZ PROGRAMOWANIA DYNAMICZNEGO

Autor:

Alicja Myśliwiec 248867

Termin zajęć:

Poniedziałek 15.15 – 16.55

Prowadzący:

Dr inż. Jarosław Mierzwa

SPIS TREŚCI

SPIS TABEL:

SPIS WYKRESÓW:

1 Wstęp teoretyczny

Celem projektu była implementacja oraz analiza efektywności algorytmów opartych o metodę przeglądu zupełnego, podziału i ograniczeń oraz programowania dynamicznego dla asymetrycznego problemu komiwojażera. Jest to zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość pomiędzy każdą parą miast. W asymetrycznym problemie odległość z miasta A do miasta B może być inna niż z miasta B do miasta A. Celem jest znalezienie najkrótszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej w określonym punkcie. Główną trudnością problemu jest liczba danych do analizy. W przypadku asymetrycznego problemu komiwojażera dla n miast liczba kombinacji wynosi $(n-1)!$. Rozwiązanie tego problemu przy użyciu naiwnego algorytmu wykorzystującego metodą przeglądu zupełnego wymaga wygenerowania wszystkich możliwych cykli oraz wybrania z nich tego o najmniejszej sumie wag krawędzi. Prowadzi to do wykładniczej klasy złożoności obliczeniowej $O(n!)$, zatem niemożliwe jest, aby uzyskać rozwiązanie dla dużych n w stosunkowo niewielkim czasie.

2 Algorytm podziału i ograniczeń

Metoda podziału i ograniczeń służy do rozwiązywania problemów optymalizacyjnych. Jej działanie opiera się na analizie drzewa przestrzeni stanów. Drzewo to reprezentuje wszystkie możliwe ścieżki jakimi może pójść algorytm rozwiązując dany problem. Algorytm zaczyna w korzeniu drzewa i przechodząc do któregoś liścia konstruuje rozwiązanie. Przeglądanie całego drzewa byłoby bardzo kosztowne ze względu na jego wykładniczy rozmiar, dlatego metoda podziału i ograniczeń w każdym węźle oblicza granicę (ograniczenie), 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 oraz liczenia granicy, zmniejszyć ilość odwiedzonych wierzchołków i szybciej znaleźć rozwiązanie problemu.

- ***Granica*** – liczba będąca ograniczeniem wartości rozwiązania jakie można uzyskać dzięki rozwinięciu danego węzła
- ***Węzeł obiecujący*** – węzeł, którego granica jest lepsza niż wartość najlepszego znalezionego dotąd rozwiązania
- ***Węzeł nieobiecujący*** – węzeł, którego granica jest gorsza niż wartość najlepszego znalezionego dotąd rozwiązania

Przy tworzeniu algorytmu należy zaprojektować dwa elementy:

- ***funkcję obliczającą granicę*** – wyznacza ona najlepsze rozwiązania, jakie można uzyskać z poddrzewa danego węzła,
- ***strategię odwiedzania wierzchołków*** – zależy od niej szybkość działania algorytmu, może być to np. strategia przeszukiwania wszerz.

W przypadku problemu komiwojażera, żeby obliczyć granicę węzła, musimy określić dolną granicę długości ścieżek, których prefiksem jest ścieżka w obecnym węźle. Długość trasy nie może być krótsza niż suma minimalnych długości krawędzi wychodzących z każdego wierzchołka. Minimalną długość krawędzi wychodzących z wierzchołka i możemy wyznaczyć przeglądając i-ty wiersz macierzy sąsiedztwa. Funkcja obliczająca granicę danego wierzchołka jest "sercem" metody podziału i ograniczeń. To od jej jakości w dużej mierze zależy szybkość działania algorytmu. Obliczone przez nią granice powinny być jak najbliższe optymalnej wartości dla podproblemów, żeby móc precyzyjnie identyfikować nieobiecujące węzły i przez to zmniejszyć ilość węzłów odwiedzonych. Nie powinna ona być jednak zbyt czasochłonna, gdyż jest obliczana często i może to odbić się na efektywności całego algorytmu. Często problem może mieć więcej niż jedną funkcję obliczającą granicę.

2.1 Opis implementacji algorytmu

Metoda podziału i ograniczeń opiera się na przeszukiwaniu w głąb drzewa reprezentującego przestrzeń rozwiązań problemu. Plusem tej metody jest możliwość ograniczenia ilości przeglądanych rozwiązań, co prowadzi do redukcji czasu potrzebnego do odnalezienia rozwiązania. Oprócz samego przeszukiwania drzewa wykorzystujemy również:

- ***Podział*** – jest to dzielenie zbioru rozwiązań reprezentowanego przez węzeł drzewa na rozłączne podzbiory, reprezentowane przez następników tego węzła
- ***Ograniczenie*** – pomijanie w poszukiwaniu tych gałęzi drzewa, o których wiadomo, że nie zawierają optymalnego rozwiązania w swoich liściach

W bieżącym poddrzewie rozwiązań poszukiwane są rozwiązania znajdujące się pomiędzy dolnym oraz górnym ograniczeniem. Najlepsze znalezione (do tej pory) rozwiązanie to wartość górnego ograniczenia, natomiast dolne ograniczenie to heurystyczne oszacowanie najlepszego rozwiązania, które może zostać odnalezione w podzbiorze reprezentowanym przez następnika bieżącego węzła. W związku z tym musi ono zostać obliczone dla każdego z nich. Jeśli dolne ograniczenie węzła jest większe od wartości górnego ograniczenia, węzeł jest odrzucany. Wartość dolnego ograniczenia musi być obliczona w taki sposób, aby żadne możliwe do uzyskania w danym poddrzewie rozwiązanie nie było lepsze od wartości tego ograniczenia. Powinna być jak najbliższa istniejącemu rozwiązaniu, jednak może od niego odbiegać.

Algorytm dokonuje przeglądu drzewa w głąb i oblicza ograniczenia dolne dla następników badanego węzła. Przegląd jest kontynuowany w wierzchołku, który ma najmniejsze ograniczenie dolne i jednocześnie jest ono mniejsze od ograniczenia górnego. Po przejściu przez wierzchołek, jest on oznaczany jako odwiedzony i nie jest rozpatrywany w dalszej analizie. W dalszej części algorytm wykonywany jest rekurencyjnie, uaktualniania jest ścieżka oraz ograniczenie górne. Algorytm wykorzystuje następujące struktury danych:

- ***Tablice*** – dwuwymiarowa z reprezentacją grafu, jednowymiarowa z informacją o odwiedzonych wierzchołkach, jednowymiarowa przechowująca wyliczone dolne ograniczenia dla wierzchołków
- ***Stos*** – w jednej instancji stosu zapamiętana jest najlepsza ścieżka, a w drugiej przechowywana jest aktualna dla danego etapu wykonywania algorytmu
- ***Kolejka priorytetowa*** – przechowuje węzły ustawione według wyznaczonych dla nich ograniczeń dolnych

Aby obliczyć ograniczenie, należy wybrać z wierszy macierzy krawędzi krawędź o najniższej wadze. Krawędzie zapisane są w tablicy pomocniczej, której indeks odpowiada wierzchołkowi, z którego wychodzi dana krawędź. Suma wag krawędzi jest ograniczeniem dolnym dla wierzchołka początkowego. Ograniczenie jest następnie aktualizowane.

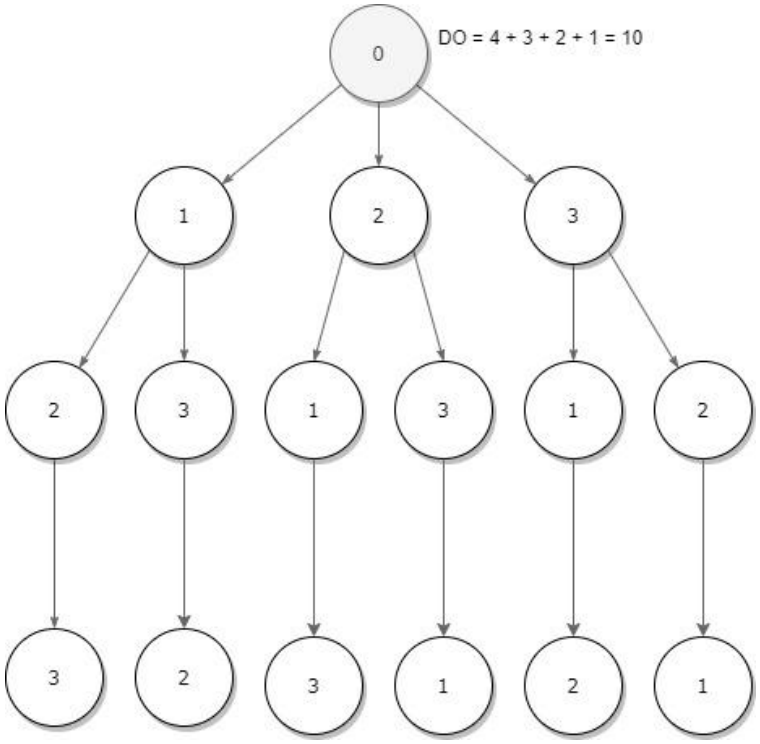
2.2 Przykład praktyczny

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

	0	**1**	**2**	**3**	**Tablica pomocnicza**
0	*-1*	14	10	4	4
1	7	*-1*	3	7	3
2	10	2	*-1*	9	2
3	6	1	2	*-1*	1

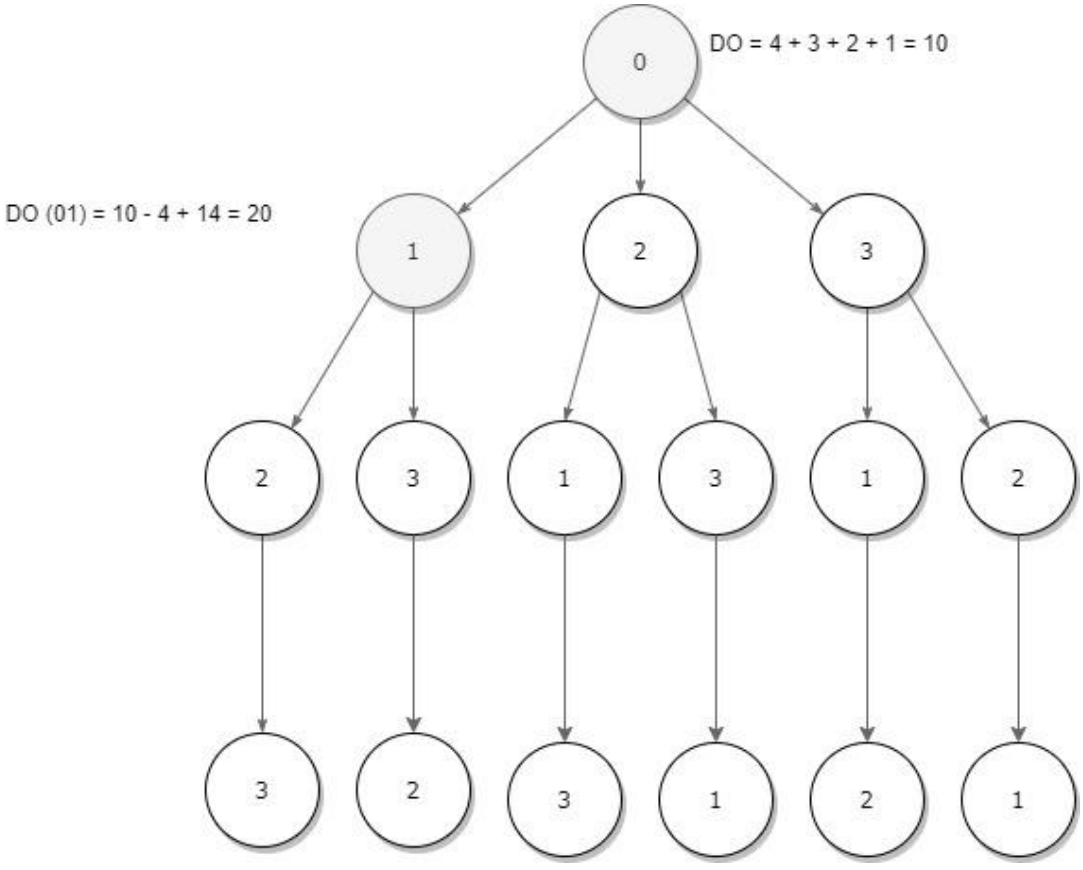
Dolne ograniczenie dla wierzchołka początkowego (kiedy nie przeszliśmy jeszcze żadnej drogi) stanowi suma elementów tablicy pomocniczej.

DO = 4 + 3 + 2 + 1 = 10



Następnie obliczamy górne ograniczenia dla następników. W tym celu od przekazanej do następnika wartości dolnego ograniczenia odejmujemy wartość wagi odpowiedniej krawędzi z tablicy pomocniczej oraz dodać wartość wagi krawędzi przebytej drogi do następnika:

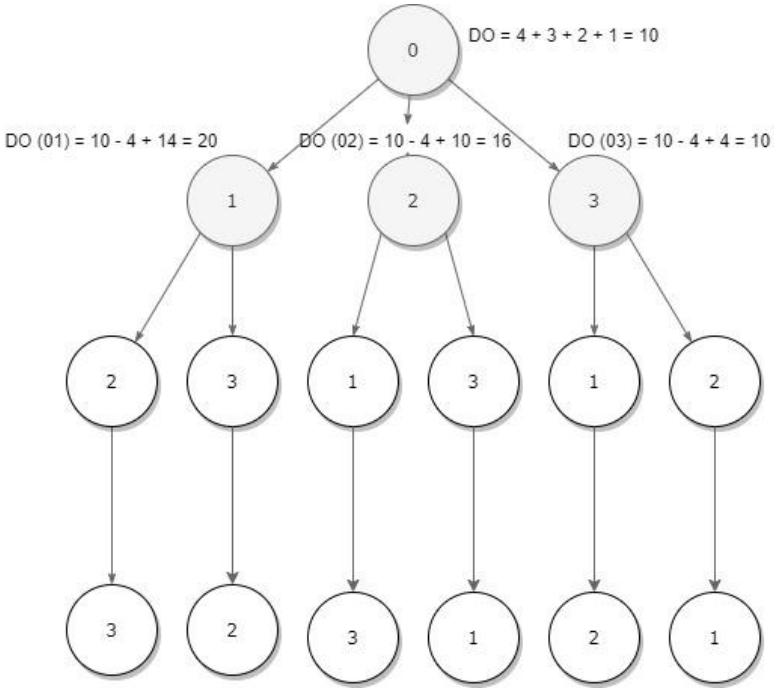
DO (01) = DO – tablica[0] + waga[0][1] = 10 – 4 + 14 = 20



Zgodnie z tą regułą obliczamy dolne ograniczenia pozostałych następników:

DO (02) = DO – tablica[0] + waga[0][2] = 10 – 4 + 10 = 16

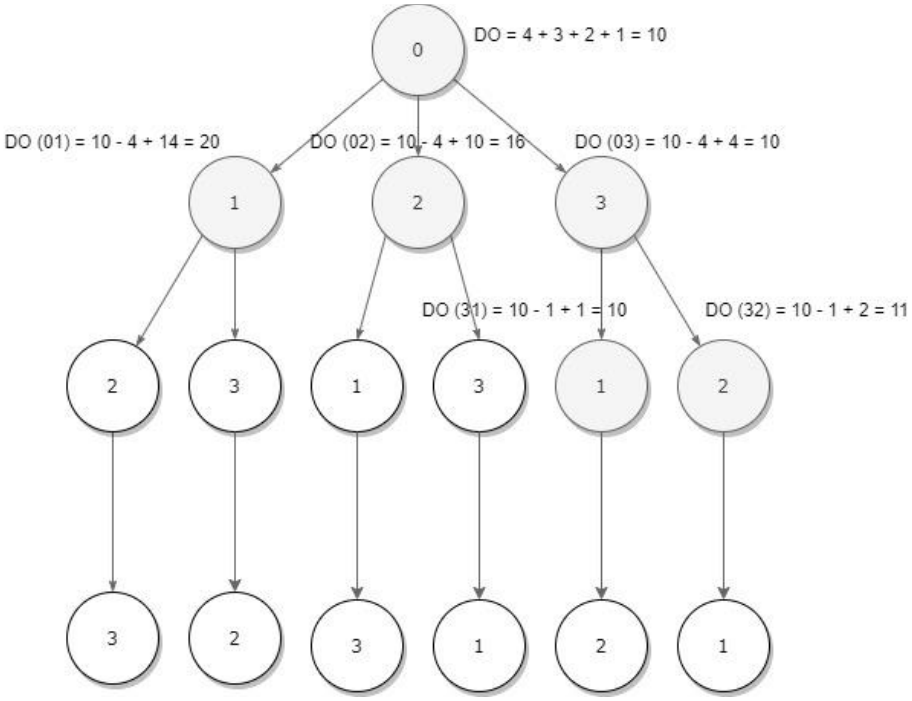
DO (03) = DO – tablica[0] + waga[0][3] = 10 – 4 + 4 = 10



Następnie wybieramy węzeł o najniższej wartości ograniczenia i jednocześnie mniejszej od górnego ograniczenia, które jest określone jako nieskończoność do momentu znalezienia rozwiązania. Należy zatem wybrać węzeł numer 3 i dokonać analogicznej czynności:

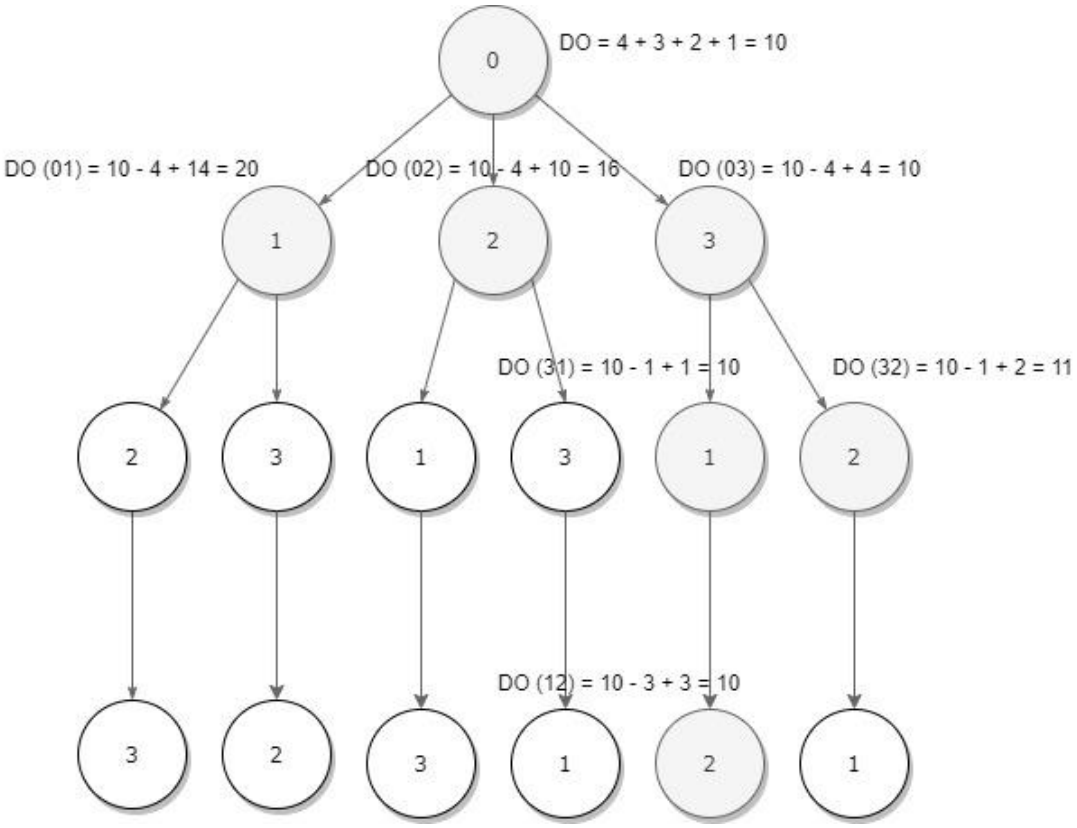
DO (31) = DO – tablica[3] + waga[3][1] = 10 – 1 + 1 = 10

DO (32) = DO – tablica[3] + waga[3][2] = 10 – 1 + 2 = 11



Do dalszej analizy wybieramy węzeł o numerze 1 i liczymy ograniczenie dolne dla jego następnika:

DO (12) = DO – tablica[1] + waga[1][2] = 10 – 3 + 3 = 10



Węzeł o numerze 2 jest liściem, zatem należy sprawdzić, czy suma wag utworzonej drogi po zamknięciu jej w wierzchołku startowym będzie mniejsza od górnego ograniczenia:

DO (20) = 10 – tablica[2] + waga[2][0] = 10 – 2 + 10 = 18

Jest to pierwsze znalezione rozwiązanie, zatem spełniony zostaje warunek DO < GO. Można zatem zaktualizować wartość górnego ograniczenia:

GO = 18

Po ustanowieniu górnego ograniczenia przechodzimy do niższych poziomów drzewa w poszukiwaniu węzłów o spełniającym warunku DO < GO i analizujemy je w taki sam sposób, aż do momentu, w którym żaden z węzłów nie będzie spełniał tego warunku. Nastąpi wtedy koniec algorytmu.

3 Programowanie dynamiczne

Jest to technika lub strategia projektowania algorytmów stosowana przeważnie do rozwiązywania zagadnień optymalizacyjnych, w tym wypadku problemu komiwojażera. Jest alternatywną dla niektórych zagadnień rozwiązywanych za pomocą algorytmów zachłannych. Opiera się na podziela rozwiązywanego problemu na podproblemy względem kilku parametrów, które nie są rozłączne, ale musi cechować je własność optymalnej podstruktury, czyli optymalnym rozwiązaniem jest funkcją optymalnych rozwiązań podproblemów. Sama liczba podproblemów jest wielomianowa. Zazwyczaj jednym z parametrów definiujących podproblemy jest liczba elementów znajdujących się w rozpatrywanym problemie, drugim jest pewna wartość liczbowa. Programowanie dynamiczne znajduje optymalną wartość funkcji celu dla całego zagadnienia, rozwiązując podproblemy od najmniejszego do największego i zapisując optymalne wartości w tablicy. Pozwala to zastąpić wywołania rekurencyjne odwołaniami do odpowiednich komórek wspomnianej tablicy i gwarantuje, że każdy podproblemy jest rozwiązywany tylko raz. Rozwiązanie ostatniego z rozpatrywanych podproblemów jest na ogół wartością rozwiązania zadanego zagadnienia. W przypadku problemu komiwojażera wiadomo, że każda droga będąca potencjalnym rozwiązaniem, zaczyna się i kończy w tym samym wierzchołku. Nie jest istotne w jakiej kolejności zostały odwiedzone wierzchołki, istotna jest jedynie suma wag krawędzi łączących te wierzchołki. Pojedynczy stan dla problemu komiwojażera określamy jako nieuporządkowany podzbiór odwiedzonych wierzchołków z zaznaczonym początkiem oraz końcem.

3.1 Opis implementacji algorytmu

Głównym zadaniem implementowanego algorytmu jest wypełnianie tablicy stanów. W tym celu zostały użyte maski bitowe, aby można było przedstawić reprezentację każdego z nieuporządkowanych podzbiorów. Maski bitowe to słowa bitowe, w których waga bitu odpowiada numerowi wierzchołka. Jeżeli bit ma wartość równą jeden, oznacza to, że wierzchołek o numerze równym wadze tego bitu znajduje się w reprezentowanym podzbiorze. Analogicznie jeżeli bit ma wartość zero, danego wierzchołka nie znajdziemy w podzbiorze. Przykładowo maska bitowa 101 oznacza, że w podzbiorze znajdują się wierzchołki o numerach 0 i 2, a wierzchołek o numerze 1 się w nim nie znajduje. W odniesieniu do problemu komiwojażera, wierzchołki znajdujące się w podzbiorze oznaczają miasta należące do ścieżki. Tablica stanów jest tablicą dwuwymiarową o rozmiarach 2^n x n, gdzie n jest równe liczbie wierzchołków w grafie. Ilość wierszy jest równa ilości możliwych kombinacji, a liczba kolumn odpowiada ilości wierzchołków, do których można przejść z danego podzbioru. Z tego względu klasa złożoności obliczeniowej wynosi $O(n^2 2^n)$. Struktury wykorzystane przy tworzeniu algorytmu:

- ***Tablica dwuwymiarowa*** – przechowuje rozwiązania podproblemów
- ***Stos*** – struktura wykorzystywana do odnalezienia minimalnej ścieżki

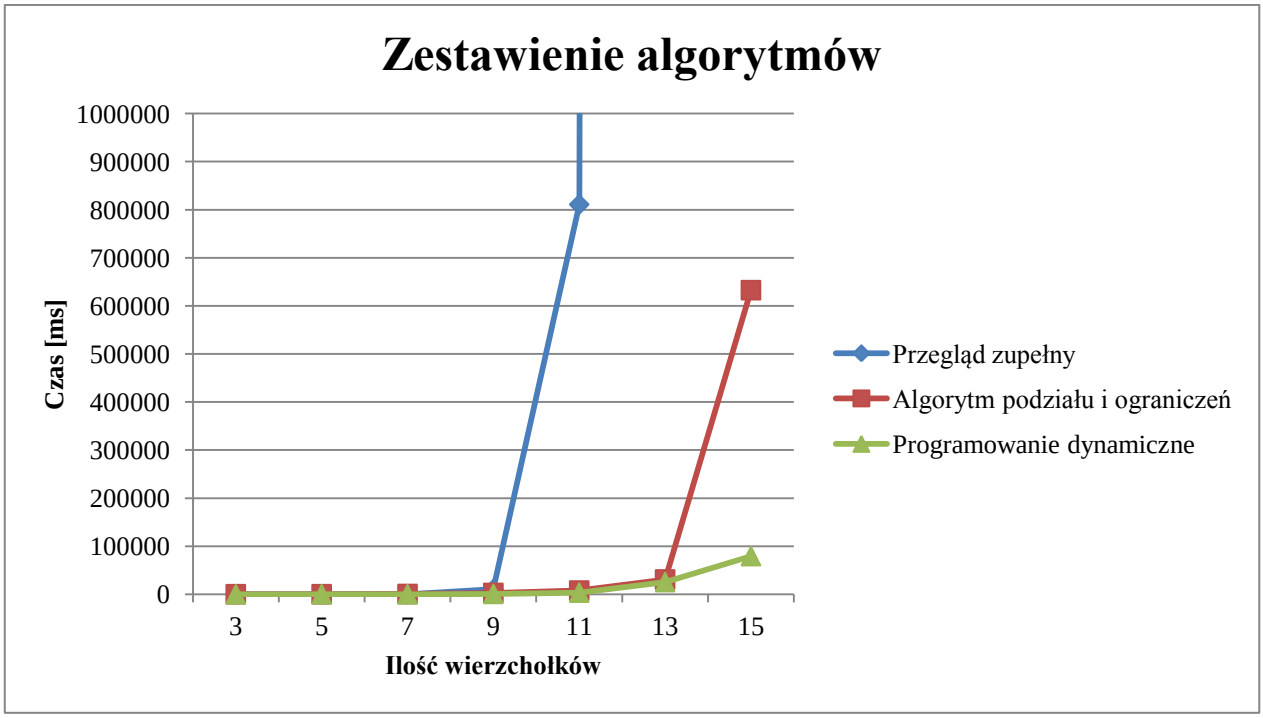
4 Plan projektu

Projekt został napisany w języku C++. Zadanie polegało na pomiarze czasu wykonywania poszczególnych algorytmów w celu analizy efektywności metod. Ponieważ pojedynczy pomiar może być obarczony znacznym błędem oraz otrzymane wyniki mogą zależeć także od rozkładu danych, to pomiary dla konkretnej pod względem wielkości instancji problemu komiwojażera zostały wykonane wielokrotnie. Na reprezentatywne wartości zostały wybrane liczby : 3, 5, 7, 9, 11, 13, 15. Grafy wejściowe były generowane jako macierze zawierające losowe liczby z przedziału 1 do 1000, co gwarantowało asymetryczność problemu. Dane znajdujące się na przekątnych zostały oznaczone wartością -1. Za każdym razem program generował nowy zestaw danych. Łącznie dla każdego rozmiaru macierzy przeprowadzono po 100 pomiarów, a wyniki zostały uśrednione i podane w mikrosekundach z dokładnością do części setnych. Do dokładnego pomiaru czasu w systemie Windows w C++ została wykorzystana funkcja QueryPerformanceCounter.

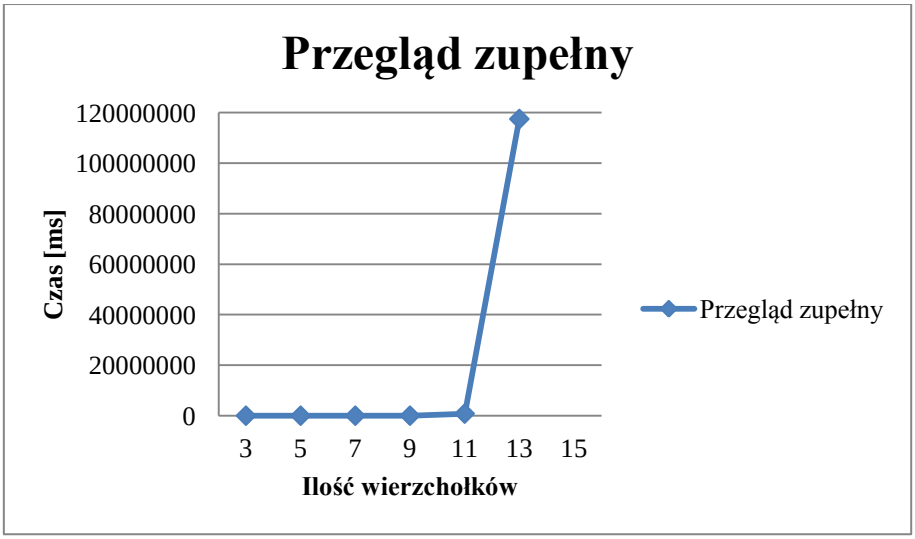
5 Analiza wyników

Tabela 1 : Wyniki pomiarów

	Czas wykonywania dla danego algorytmu [s]		
Rozmiar problemu	Przegląd zupełny	Algorytm podziału i ograniczeń	Programowanie dynamiczne
3	**78.03**	**80.12**	**61.09**
5	**99.07**	**110.88**	**74.65**
7	**214.51**	**521.39**	**170.71**
9	**10273.77**	**2793.29**	**782.87**
11	**810797**	**7691.83**	**3412.54**
13	**117479000**	**30729.2**	**25318.5**
15	***Przekroczony dopuszczalny czas***	**632524**	**79404.1**

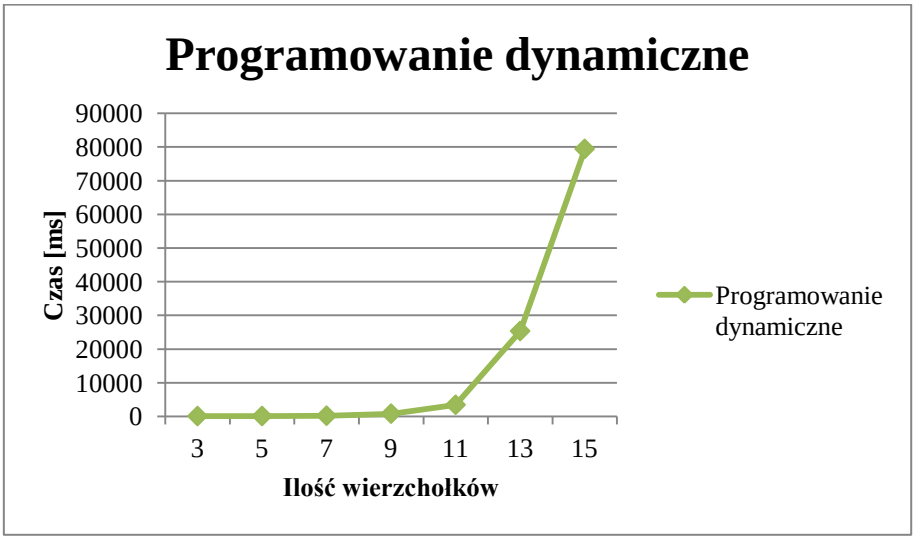


Wykres 1 : Zestawienie algorytmów



Wykres 2 : Przegląd zupełny

Wykres 3 : Algorytm podziału i ograniczeń



Wykres 4 : Programowanie dynamiczne

6 Wnioski

Eksperyment pozwolił zaobserwować, jak przedstawiają się wyniki rzeczywistych pomiarów w porównaniu z wcześniejszymi założeniami co do ich złożoności. Powyższe wykresy ukazują, że te teorie są słuszne. Oczywiście czas obliczeń zależy w dużym stopniu od wydajności komputera, na którym pracujemy. Oprócz eksperymentu, ma on jeszcze inne zadania do wykonania w tym samym czasie. Procesy działające w tle mogły wpływać na chwilowe spadki mocy obliczeniowej tym samym powodować wolniejsze wykonywanie operacji. Innym źródłem błędów mógł być fakt, że program generował losowy zestaw danych, zatem algorytmy mogły różnie radzić sobie z różnymi zestawami. Należy również pamiętać, że w wykonywanych zadaniach nie została uwzględniona złożoność pamięciowa, która dałabym jeszcze szerszy obraz wyników eksperymentu.

Zgodnie z przewidywaniami, algorytm przeglądu zupełnego nie sprawdza się w przypadku większych instancji problemu, ponieważ czas wykonania algorytmu jest długi już w przypadku 13 miast. Jest on jednak najbardziej intuicyjny i najprostszy w zaimplementowaniu – najmniej abstrakcyjny.

Algorytm oparty o metodę podziału i ograniczeń działał nieco sprawniej. Mimo podobnej klasy złożoności co algorytm przeglądu zupełnego. Jednak wprowadzenie rozgałęzień i ograniczeń przeszukiwania nieco usprawnił czas wykonywania tego algorytmu. Istotny jest tutaj dobór funkcji liczącej ograniczenia oraz dobór metody przeszukiwania drzewa, gdyż czas algorytmu będzie się różnił w zależności od dokładności wybranych metod. W niektórych węzłach zaimplementowana funkcja daje lepsze wyniki, w innych gorsze, zależy to również od wartości odległości między węzłami (miastami).

Najlepiej sprawdził się algorytm oparty na programowaniu dynamicznym. Działał on sprawnie dla tych instancji, dla których poprzednie algorytmu wymagały więcej czasu. Uzyskane pomiary czasów są całe rzędy niższe, zatem występuje zgodność teoretyczna z klasą złożoności algorytmu.

7 Bibliografia

- https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met_podz_ogr.opr.pdf
- https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm?fbclid=IwAR01i1NgXOjPtD7PWokhD9po1KBzfT4JJpynNYRtJrPwcKNGX4oeDtLUGTY
- http://mst.mimuw.edu.pl/lecture.php?lecture=op1&part=Ch12
- https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met_podz_ogr.pdf
- https://cs.pwr.edu.pl/zielinski/lectures/om/mow10.pdf
- https://pl.wikipedia.org/wiki/Programowanie_dynamiczne
- http://algorytmy.ency.pl/artykul/programowanie_dynamiczne
- http://www.algorytm.org/kurs-algorytmiki/programowanie-dynamiczne.html
- http://smurf.mimuw.edu.pl/node/297
- https://pl.wikipedia.org/wiki/Problem_komiwoja%C5%BCera
- http://www.mini.pw.edu.pl/MiNIwyklady/grafy/prob-komiw.html
- https://eduinf.waw.pl/inf/alg/001_search/0140.php
- https://pl.wikipedia.org/wiki/Przeszukiwanie_grafu
- https://pl.wikipedia.org/wiki/Przeszukiwanie_wszerz
- https://pl.wikipedia.org/wiki/Przeszukiwanie_w_g%C5%82%C4%85b