Patryk Skowroński 237454 Wrocław, dn. 12.11.2019

Zadanie projektowe nr 1.

Implementacja i analiza efektywności algorytmu wykorzystującego metodę podziału i ograniczeń oraz programowania dynamicznego.

Sprawozdanie z zadania projektowego przedmiotu „ Projektowanie efektywnych algorytmów”.

Rok akademicki 2019/2020, kierunek Informatyka.

Prowadzący:

Dr inż. Zbigniew Buchalski

Termin zajęć:

Wtorek, 9.15

1. Wstęp teoretyczny

Celem wykonania zadania projektowego była implementacja oraz późniejsza analiza efektywności algorytmów opartych o metodę przeglądu zupełnego (ang. Brute Force), podziału i ograniczeń (ang. Branch & Bound) oraz programowania dynamicznego (ang. Dynamic Programming) dla asymetrycznego problemu komiwojażera (ATSP). Rozpatrywany problem polega na odnalezieniu minimalnego cyklu Hamiltona w pełnym, skierowanym, grafie ważonym. Cykl Hamiltona to droga rozpoczynająca się w danym wierzchołku grafu, przechodząca przez każdy z pozostałych wierzchołków dokładnie raz i kończąca się w wierzchołku początkowym. Minimalny cykl Hamiltona jest cyklem składającym się z tych krawędzi, których suma wag jest najmniejsza spośród sum wag innych cykli Hamiltona znajdujących się w grafie. Odnalezienie takiego cyklu w grafie pełnym (założenie pesymistyczne) G(V, E) o n wierzchołkach, w którym krawędź e_{kp} łącząca wierzchołek v_k z v_p (gdzie k, p ∈ {1; n}, v_k, v_p ∈ V oraz e_{pk}, e_{kp} ∈ E) ma taką samą wagę jak krawędź e_{pk} łącząca wierzchołek v_p z v_k lub różną od niej jest równoznaczne kolejno z rozwiązaniem symetrycznego problemu komiwojażera (STSP) lub asymetrycznego problemu komiwojażera (ATSP).

Główną trudnością związaną z poszukiwaniem rozwiązania dla problemu komiwojażera jest bardzo duża liczba danych do analizy. Liczba wszystkich możliwych kombinacji krawędzi tworzących cykl Hamiltona w grafie o n wierzchołkach dla przypadku ATSP wynosi $(n-1)!$. Rozwiązanie danej instancji tego problemu przy użyciu naiwnego algorytmu wykorzystującego metodę przeglądu zupełnego wymaga wygenerowania wszystkich możliwych cykli (kombinacji) oraz wybrania z nich tego o najmniejszej sumie wag budujących go krawędzi. Prowadzi to do wykładniczej klasy złożoności obliczeniowej O(n!), zatem niemożliwym jest uzyskanie rozwiązania dla dużych n w akceptowalnym czasie.

2. Rozpatrywane podejścia do rozwiązania problemu TSP.

W poniższych podpunktach zostały zestawione wymienione we wstępie strategie znajdowania optymalnych (dokładnych) rozwiązań dla problemu komiwojażera, które zostały wykorzystane do sformułowania algorytmów zaimplementowanych w ramach zadania projektowego. Przegląd zupełny, ze względu na swoją specyfikę został przedstawiony we wstępie i nie będzie szczegółowo przedstawiany w dalszej części sprawozdania.

2.1. Podział i ograniczenia - Branch & Bound.

Metoda podziału i ograniczeń, podobnie jak metoda przeglądu zupełnego opiera się na przeszukiwaniu w głąb drzewa reprezentującego przestrzeń rozwiązań problemu. Kluczową własnością tej metody, odróżniającą ją od metody naiwnej jest możliwość ograniczenia ilości przeglądanych rozwiązań, co w konsekwencji może prowadzić do redukcji ilości czasu potrzebnego do znalezienia optimum globalnego. Jest to możliwe dzięki wzbogaceniu przeglądania drzewa rozwiązań o następujące procedury:

- **Podział** (lub rozgałęzianie, ang. branching) - dzielenie zbioru rozwiązań reprezentowanego przez węzeł drzewa na rozłączne podzbiory, reprezentowane przez następników tego węzła.
- **Ograniczanie** (ang. bounding) - pomijanie w przeszukiwaniu 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. Górne ograniczenie stanowi wartość najlepszego znalezionego do tej pory rozwiązania, natomiast dolne ograniczenie jest heurystycznym oszacowaniem 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. Podczas przeszukiwania drzewa odrzucane są te węzły dla których dolne ograniczenie jest wyższe od wartości górnego ograniczenia.

Wartość dolnego ograniczenia musi być obliczona tak, żeby żadne, możliwe do uzyskania w danym poddrzewie rozwiązanie nie było lepsze od wartości tego ograniczenia. Może ona zatem odbiegać od rzeczywistej wartości możliwej do uzyskania ale jednocześnie powinna być ona możliwie jak nabliższa istniejącemu rozwiązaniu, co będzie się wiązało dokonywaniem większej ilości pominięć.

2.1.1. Implementacja algorytmu opartego o strategię Branch & Bound.

Jednym z elementów zadania projektowego była implementacja w języku C++ algorytmu rozwiązującego problem komiwojażera, opartego o metodę podziału i ograniczeń. 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 naniższej wartości dolnego ograniczenia.

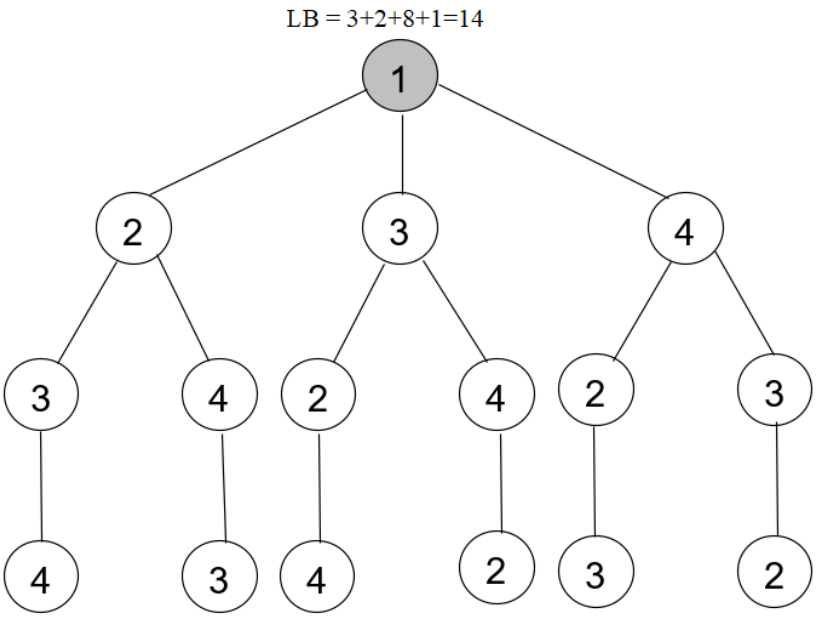
Obliczanie ograniczenia dolnego odbywa się poprzez wybór z wierszy macierzy krawędzi o jak najniższej wadze. Krawędzie te są zapamiętywane w tablicy której indeks odpowiada wierzchołkowi z którego wychodzi dana krawędź. Suma wag tych krawędzi wyznacza dolne ograniczenie dla wierchołka początkowego. Ograniczenie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

2.1.2. Przykład.

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

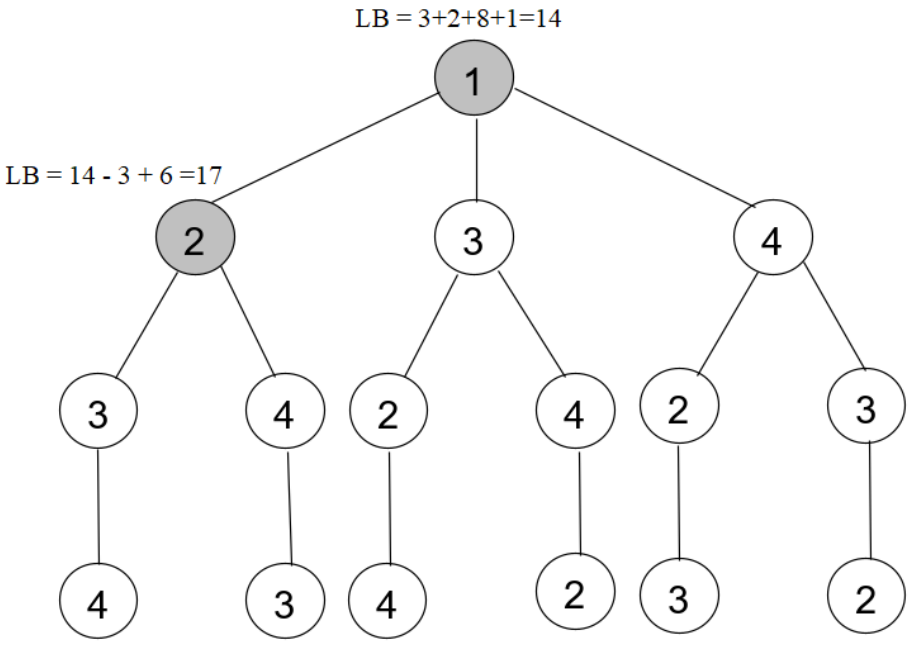
	1	**2**	**3**	**4**	Zawartość tablicy pomocniczej
1	N	6	3	9	3
2	2	N	5	4	2
3	10	8	N	9	8
4	5	1	8	N	1

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:

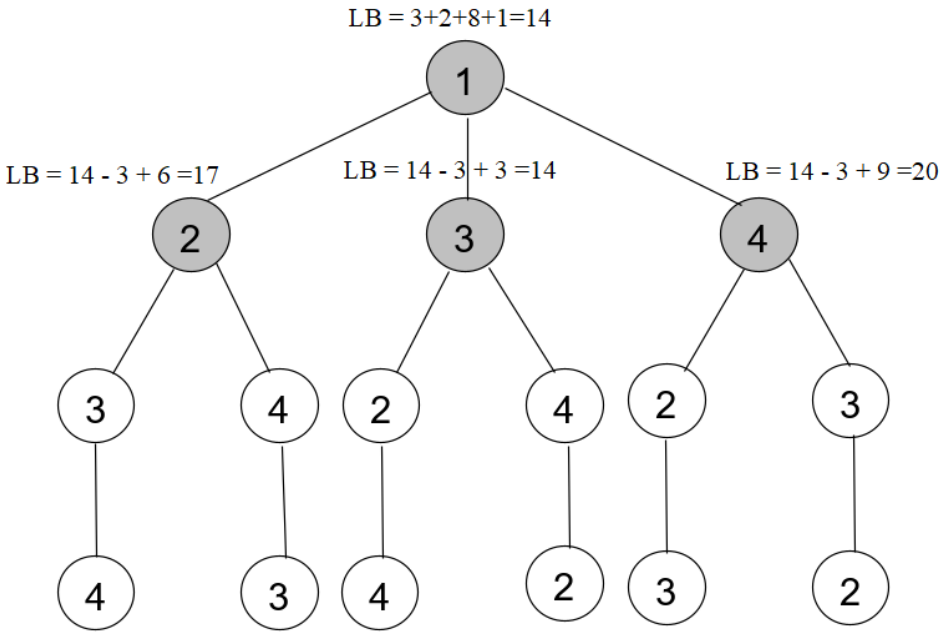
LB_{12} = 14 – tablica[1] + krawędź[1][2]



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

LB_{13} = 14 – tablica[1] + krawędź[1][3]

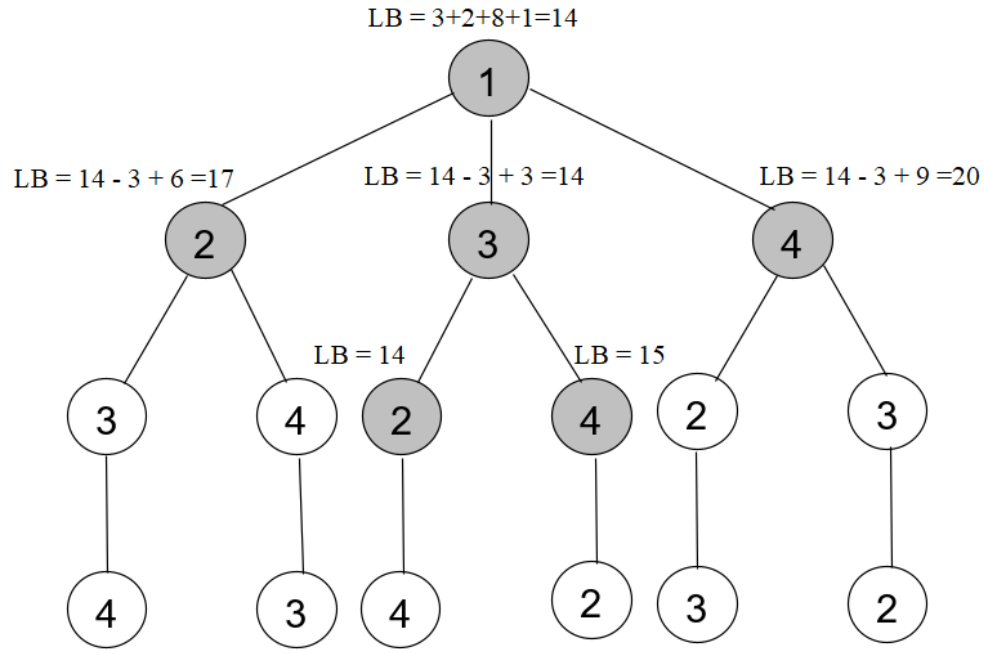
LB_{14} = 14 – tablica[1] + krawędź[1][4]



Następnym krokiem jest wybór węzła o najniżej 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 ekploracji gałęzi którą reprezentuje. Stąd:

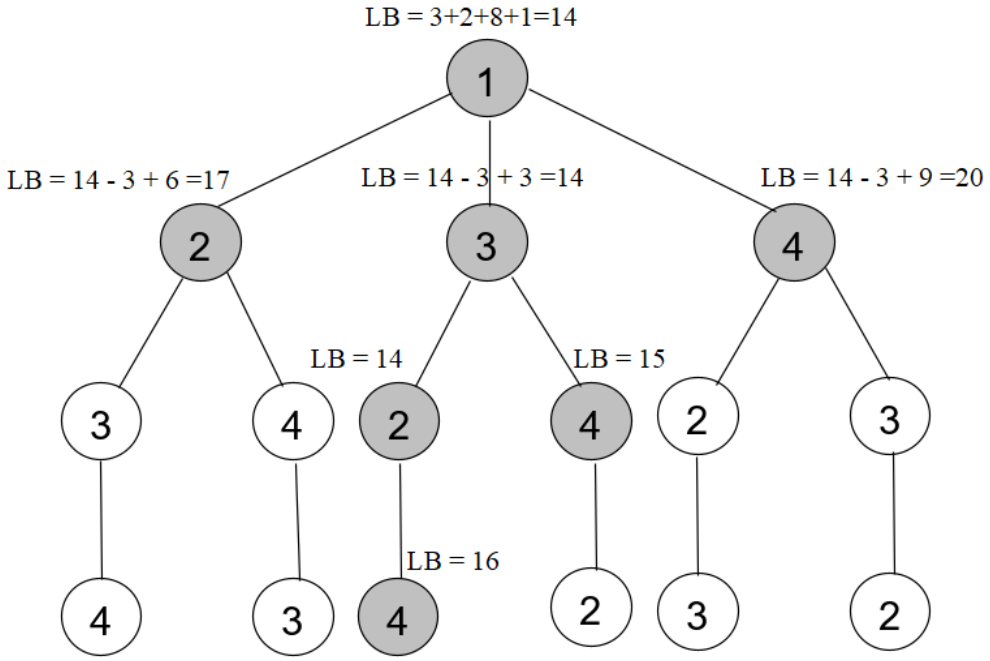
LB_{32} = 14 – tablica[3] + krawędź[3][2]

LB_{34} = 14 – tablica[3] + krawędź[3][4]



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

LB_{24} = 14 – tablica[2] + krawędź[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 – tablica[4] + krawędź[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.

2.2. Programowanie dynamiczne – Dynamic Programming.

Drugą startegią, rozpatrywaną w ramach zadania projektowego było programowanie dynamiczne. Polega ona na rozkładaniu w miarę możliwości danego problemu na wielomianową liczbę podproblemów oraz rozwiązywania ich począwszy od tych najprostszych i wykorzystywaniu ich wyników do rozwiązywania kolejnych, bardziej złożonych. Nie potrzebne staje się zatem wykorzystywanie rekurencji, gdyż rozwiązania podproblemów są zapamiętywane w tablicy stanów. W przypadku problemu komiwojażera wiadomym jest że każda droga będąca potencjalnym rozwiązaniem, zaczyna się i kończy w tym samym wierzchołku. Nieistotne zatem jest w jakiej kolejności zostały na jej trasie odwiedzone wierzchołki. Istotna jest jedynie suma wag przebytych krawędzi. W związku z powyższym, pojedynczy stan dla problemu komiwojażera można określić jako nieuporządkowany podzbiór odwiedzonych wierzchołków z zaznaczonym początkiem oraz końcem. Ogólną regułę rozwiązywania podproblemu można sformułować następująco:

$$d[\mathbf{K}][\mathbf{w}] = \min\{ d[\mathbf{K}/\mathbf{w}][\mathbf{v}] + l[\mathbf{v}][\mathbf{w}] \mid v \in \mathbf{K}/\{\mathbf{w},1\}\} \text{ dla } \mathbf{w} \neq 1$$

gdzie:

d[**K**][**w**] to suma wag krawędzi wchodzących w skład ścieżki rozpoczynającej się w wierzchołku pierwszym, przechodząca przez podbiór wierzchołków **K** i kończącej się w wierzchołku **w**.

l[**v**][**w**] to waga krawędzi biegnącej od wierzchołka **v** do **w**.

2.2.1. Implementacja algorytmu wykorzystującego programowanie dynamiczne.

Głównym zadaniem implementowanego algorytmu jest wypełnianie tablicy stanów. Do reprezentacji każdego z nieuporządkowanych podzbiorów zostały wykorzystane maski bitowe. Są 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. Jeżeli bit ma wartość zero, wierzchołek wskazywany przez jego wagę nie znajduje się w podzbiorze.

Tablica stanów jest tablicą dwuwymiarową, posiadającą $2^{\mathbf{n}}$ wierszy oraz **n** kolumn, gdzie n jest równe liczbie wierzchołków zadanego grafu. Ilość wierszy jest równa ilości możliwych kombinacji (podzbiorów). Odwoływanie do wierszy odbywa się poprzez podanie maski bitowej danego zbioru. Liczba kolumn odpowiada ilości wierzchołków do których można przejść z danego podzbioru. Skutkuje to osiągnięciem klasy złożoności obliczeniowej $O(n^n 2^n)$. Do implementacji algorytmu zostały wykorzystane takie struktury danych jak:

- **Tablica dwuwymiarowa** – przechowująca rozwiązania podproblemów
- **Stos** – jest to struktura pomocnicza, wykorzystywana przy identyfikacji drogi stanowiącej rozwiązanie problemu

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.

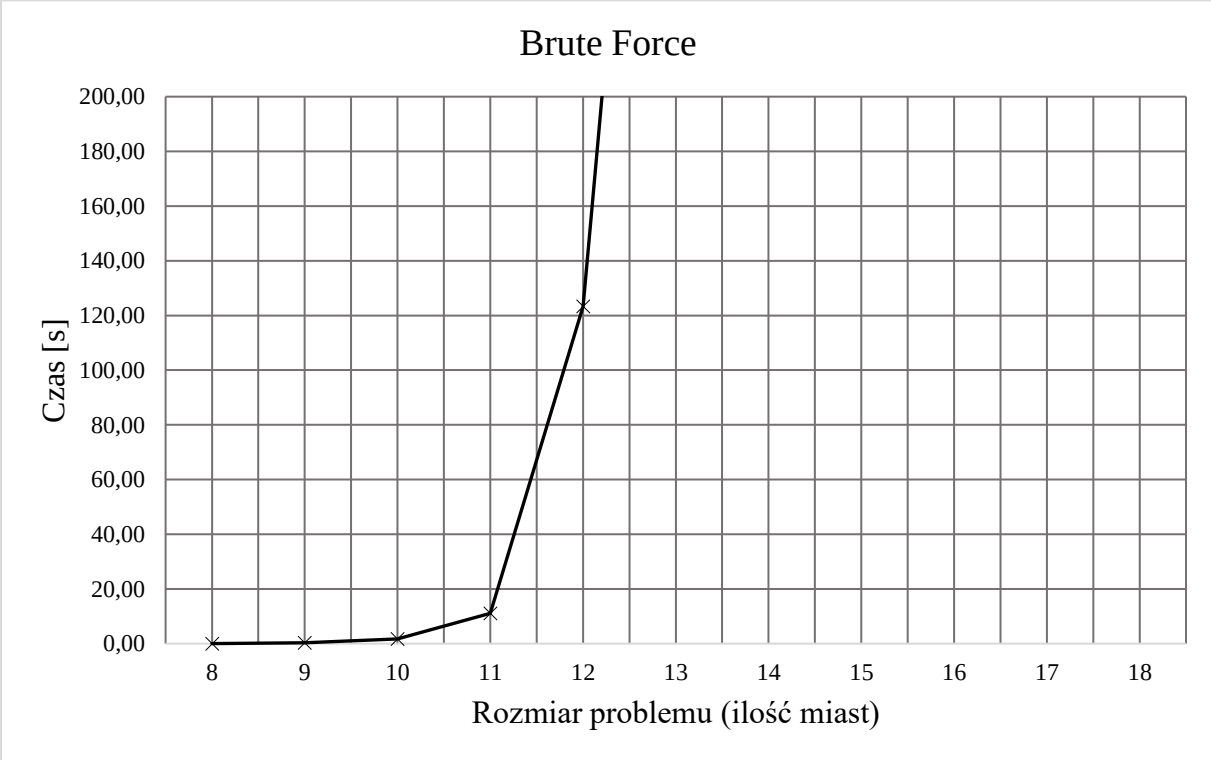
4. Analiza danych pomiarowych.

Tabela 1. Wyniki pomiarów czasu wykonania zaimplementowanych algorytmów.

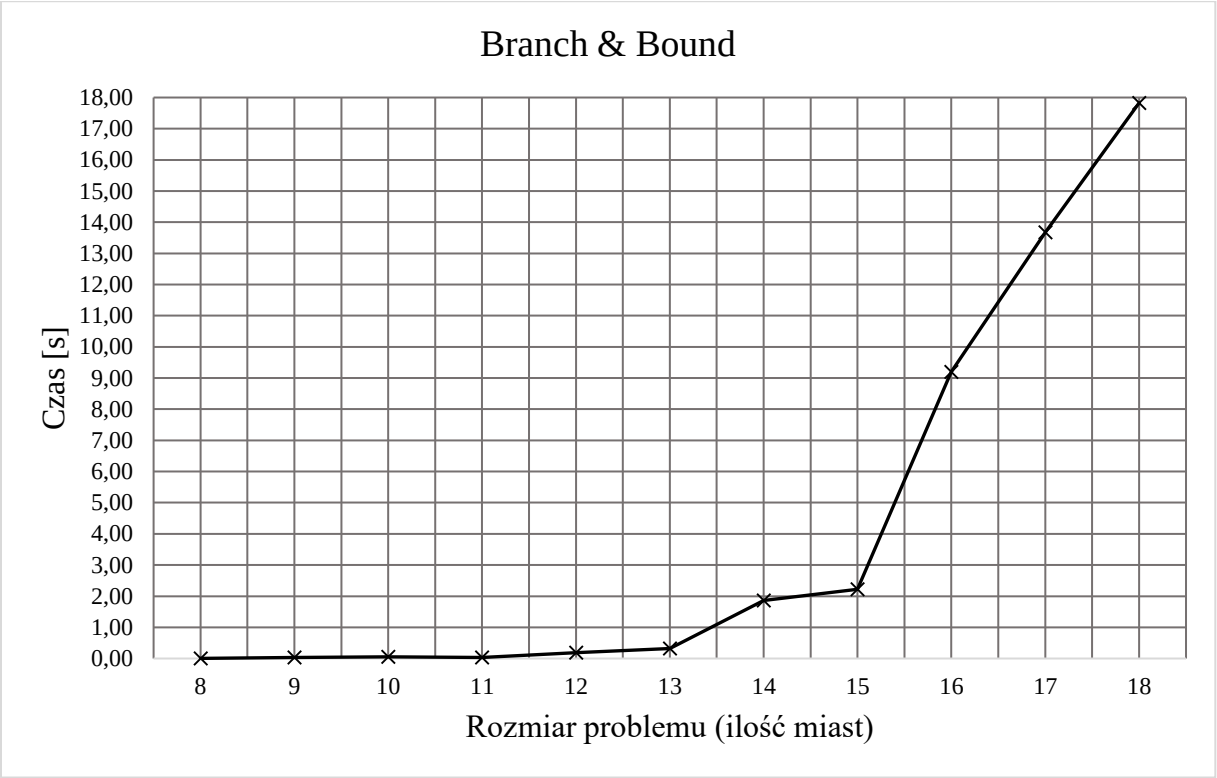
	Czas wykonania dla danego algorytmu [s]		
Rozmiar problemu	Brute Force	Branch & Bound	Dynamic Programming
8	0,02238	0,00482	0,00013
9	0,27035	0,03196	0,00034
10	1,74962	0,05525	0,00078
11	11,10182	0,03559	0,00258
12	123,30792	0,18600	0,00481
13	PC	0,32282	0,01083
14	PC	1,85850	0,02032
15	PC	2,21952	0,03669
16	PC	9,19265	0,08019
17	PC	13,67220	0,16307
18	PC	17,82609	0,35942

* PC – oznacza że przekroczony został dopuszczalny czas wykonania

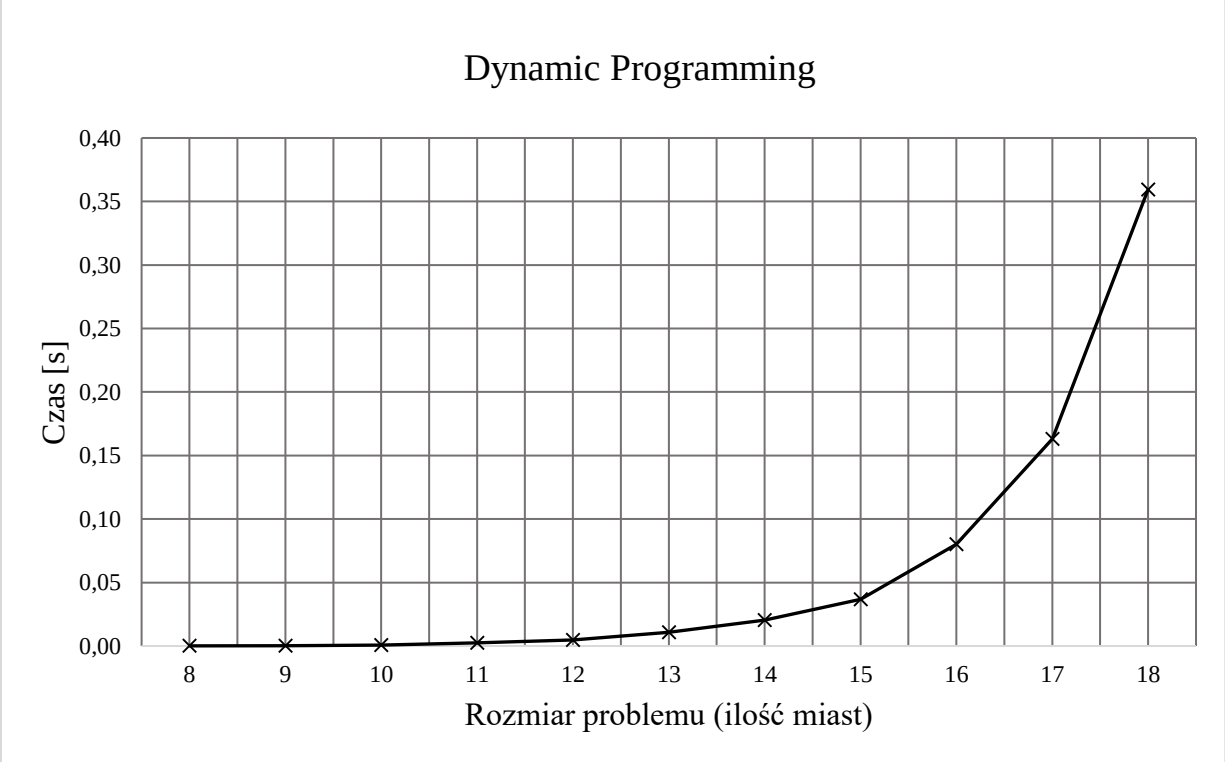
Wykres 1. Zależność czasu wykonania od rozmiaru problemu dla przeglądu zupełnego



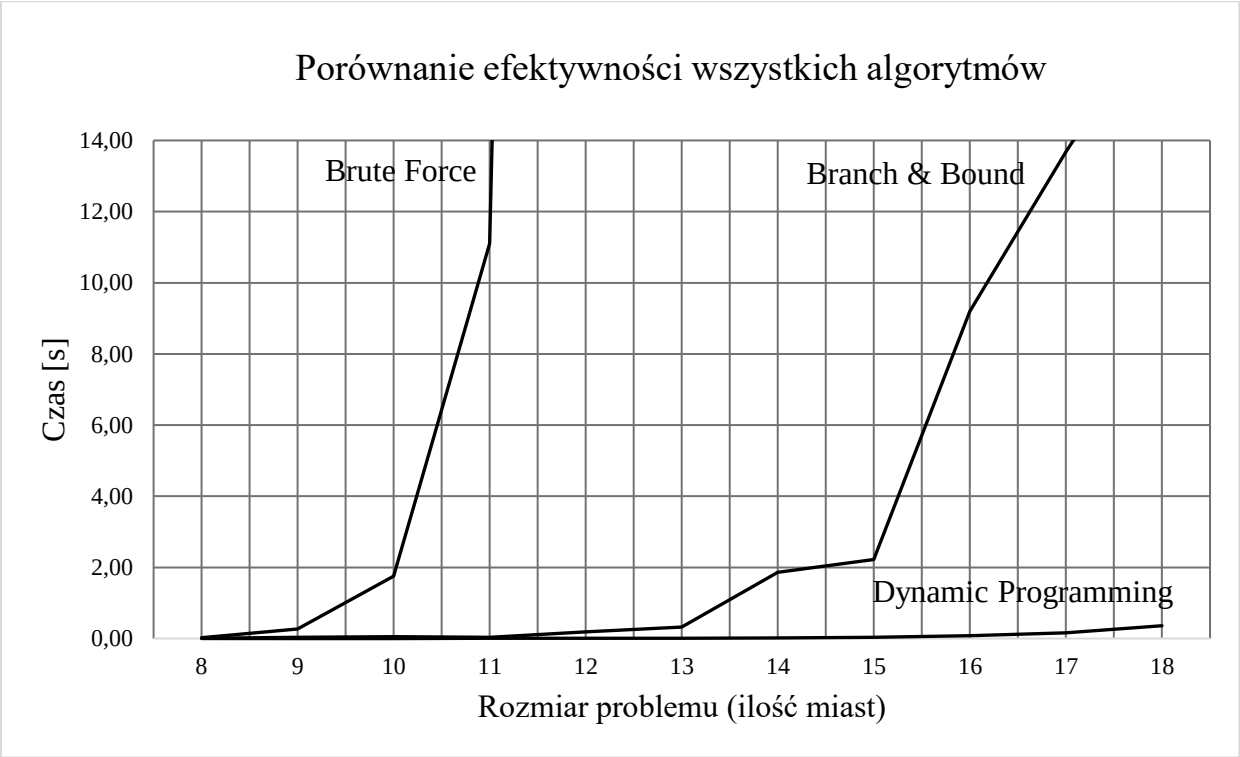
Wykres 2. Zależność czasu wykonania od rozmiaru problemu dla metody podziału i ograniczeń



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



Wykres 4. Ogólne zestawienie efektywności



5. Wnioski.

Analiza uzyskanych na drodze pomiarów danych wykazuje zgodność wynikających z nich zależności z oszacowanymi we wstępie klasami złożoności obliczeniowej dla każdej z metod. Zaimplementowany algorytm przeglądu zupełnego wykazał jego teoretyczną, wykładniczą klasę obliczeniową O(n!). Podczas przeprowadzania eksperymentu dla metody naiwnej, przerwane zostały pomiary dla instancji problemu których wielkość przekraczała 12 miast. Wynikało to z nieakceptowalnej ilości czasu potrzebnego na znalezienie dla takich instancji rozwiązań. Metoda ta nadaje się jedynie do rozwiązywania instancji problemu o małych rozmiarach.

Kolejny zaimplementowany algorytm został oparty o metodę podziału i ograniczeń. Podobnie jak w przypadku przeglądu zupełnego, jego złożoność obliczeniową ogranicza funkcja wykładnicza n!. Jednakże dzięki zastosowaniu procedur rozgałęziania oraz ograniczania przeglądu możliwych rozwiązań oraz dobrania dokładnej funkcji liczącej ograniczenie dolne, czas potrzebny do znalezienia rozwiązania dla każdej z badanych instancji został względem metody naiwnej wyraźnie zredukowany. Niedokładny dobór funkcji ograniczającej może prowadzić do nieefektywnego odcinania gałęzi i skutkować czasem wykonania zbliżonym do przeglądu zupełnego.

Ostatni z zaimplementowanych algorytmów wykorzystywał metodę programowania dynamicznego. Również w tym przypadku została zaobserwowana zgodność z teoretyczną klasą złożoności $O(n^2 2^n)$ która jest możliwa do osiągnięcia dzięki ograniczeniu działania całego algorytmu do procedury wypełniania tablicy stanów. Algorytm ten okazał się najbardziej

efektywnym. Uzyskane pomiary czasów są całe rzędy niższe od tych uzyskanych w przypadku metody Branch & Bound.

6. Bibliografia.

1. http://www.cs.put.poznan.pl/mkasprzak/zp/ZP-wyklad5.pdf
2. https://eduinf.waw.pl/inf/alg/001_search/0140.php
3. https://pl.wikipedia.org/wiki/Problem_komiwojażera
4. http://smurf.mimuw.edu.pl/node/297
5. http://informatyka.wroc.pl/node/227