Podstawy efektywnych algorytmów

Prowadzący zajęcia: dr inż. Jarosław Mierzwa
Termin zajęć: Piątek, godz. 11:15

Autor projektu: Mateusz Gawłowski 264463	**Rodzaj grupy:** Późniejsza
Etap projektu: 1	**Data oddania sprawozdania:** 24.11.2023

Spis treści

1. Wstęp teoretyczny 2
2. Rozpatrywane podejścia do rozwiązania problemu TSP. 2
   2.1. Podział i ograniczenia - Branch & Bound. 3
      2.1.1. Implementacja algorytmu opartego o strategię Branch & Bound. 3
      2.1.2. Przykład. 5
   2.2. Programowanie dynamiczne – Dynamic Programming. 10
      2.2.1. Implementacja algorytmu wykorzystującego programowanie dynamiczne 11
3. Plan eksperymentu. 11
4. Analiza danych pomiarowych. 12
5. Wnioski. 14
6. Bibliografia 15

1. Wstęp teoretyczny

Projekt ten obejmował implementację i ocenę efektywności trzech algorytmów: metodę przeglądu zupełnego (Brute Force), algorytm Branch & Bound oraz programowanie dynamiczne, stosowane w rozwiązywaniu asymetrycznego problemu komiwojażera (ATSP). Celem było znalezienie minimalnego cyklu Hamiltona w grafie ważonym, pełnym i skierowanym. Cykl Hamiltona definiuje się jako ścieżkę, która rozpoczyna się i kończy w tym samym wierzchołku, przechodząc dokładnie raz przez wszystkie inne wierzchołki. Szukany minimalny cykl Hamiltona charakteryzuje się najniższą możliwą sumą wag krawędzi w porównaniu z innymi cyklami Hamiltona w grafie.

W pełnym grafie G(V, E) z n wierzchołkami, gdzie każda krawędź ekp łącząca wierzchołki vk i vp (dla k, p ∈ {1; n}) może mieć tę samą wagę co krawędź epk lub inną, rozwiązanie problemu sprowadza się do ATSP lub symetrycznego problemu komiwojażera (STSP). Kluczowym wyzwaniem w rozwiązywaniu ATSP jest ogromna liczba potencjalnych kombinacji krawędzi tworzących cykl Hamiltona, wynosząca $(n-1)!$, co przy wykorzystaniu algorytmu Brute Force skutkuje koniecznością generowania wszystkich możliwych cykli i selekcji cyklu o najniższej sumie wag. Taka metoda charakteryzuje się wykładniczą złożonością obliczeniową $O(n!)$, co przy dużych wartościach n czyni rozwiązanie problemu niepraktycznym w rozsądnym czasie.

2. Rozpatrywane podejścia do rozwiązania problemu TSP.

W niniejszej sekcji przedstawiono strategie rozwiązywania problemu komiwojażera, które posłużyły do opracowania algorytmów zaimplementowanych w ramach projektu. Metoda przeglądu zupełnego, ze względu na jej szczególny charakter, została opisana we wstępie i nie będzie dalej szczegółowo omawiana w treści sprawozdania.

2.1. Podział i ograniczenia - Branch & Bound.

Metoda Branch & Bound, podobnie jak przegląd zupełny, bazuje na przeszukiwaniu drzewa reprezentującego przestrzeń rozwiązań problemu. Główną cechą odróżniającą ją od metody Brute Force jest zdolność do ograniczenia liczby przeglądanych rozwiązań, co przekłada się na skrócenie czasu potrzebnego do osiągnięcia globalnego optimum. Efektywność tej metody opiera się na dwóch kluczowych procedurach:

1. Podział (Branching): polega na dzieleniu zbioru rozwiązań reprezentowanego przez dany węzeł drzewa na mniejsze, rozłączne podzbiory, które są następnie reprezentowane przez węzły potomne.

2. Ograniczanie (Bounding): polega na pomijaniu tych gałęzi drzewa, które na pewno nie zawierają optymalnego rozwiązania w swoich liściach.

Podczas przeszukiwania konkretnego poddrzewa, poszukiwane są rozwiązania zawierające się między zdefiniowanym dolnym a górnym ograniczeniem. Górne ograniczenie jest aktualną najlepszą wartością rozwiązania, natomiast dolne ograniczenie stanowi heurystyczne oszacowanie najlepszego rozwiązania możliwego do znalezienia w danym poddrzewie. Dolne ograniczenie musi być obliczone dla każdego węzła potomnego, a węzły, dla których dolne ograniczenie przewyższa górne, są odrzucane.

Dolne ograniczenie powinno być wyznaczone tak, aby żadne potencjalne rozwiązanie w danym poddrzewie nie było lepsze od tego ograniczenia. Chociaż wartość ta może różnić się od faktycznej wartości najlepszego możliwego rozwiązania, powinna być jak najbliżej realnej wartości, co przyczynia się do zwiększenia efektywności metody poprzez eliminację większej liczby nieoptymalnych ścieżek.

2.1.1. Implementacja algorytmu opartego o strategię Branch & Bound.

W ramach projektu wykonano implementację algorytmu rozwiązującego problem komiwojażera w języku C++, opartego na metodzie Branch & Bound. Algorytm ten przeszukuje drzewo rozwiązań w głąb, obliczając dolne ograniczenia dla następników danego węzła. Kolejny etap przeglądu koncentruje się na węźle, którego dolne ograniczenie jest najniższe spośród dostępnych i jednocześnie niższe od obecnego ograniczenia górnego. Po eksploracji węzeł ten zostaje wykluczony z dalszego przeszukiwania. Proces ten jest realizowany rekurencyjnie, z jednoczesną modyfikacją bieżącej ścieżki oraz ograniczenia górnego.

Do implementacji tego algorytmu użyto następujących struktur danych:

1. Tablice:
   - Dwuwymiarowa tablica reprezentująca graf.
   - Dwie jednowymiarowe tablice, jedna przechowująca informacje o odwiedzonych wierzchołkach, a druga służąca do obliczania dolnych ograniczeń.

2. Stos:
   - Instancja stosu zapamiętująca najlepsze dotychczasowe rozwiązanie.
   - Instancja stosu przechowująca tymczasową ścieżkę na danym etapie algorytmu.

3. Kolejka Priorytetowa:
   - Zawiera węzły, których priorytet jest określony przez wartość dolnego ograniczenia. Najwyższy priorytet przyznawany jest wierzchołkowi z najniższym dolnym ograniczeniem.

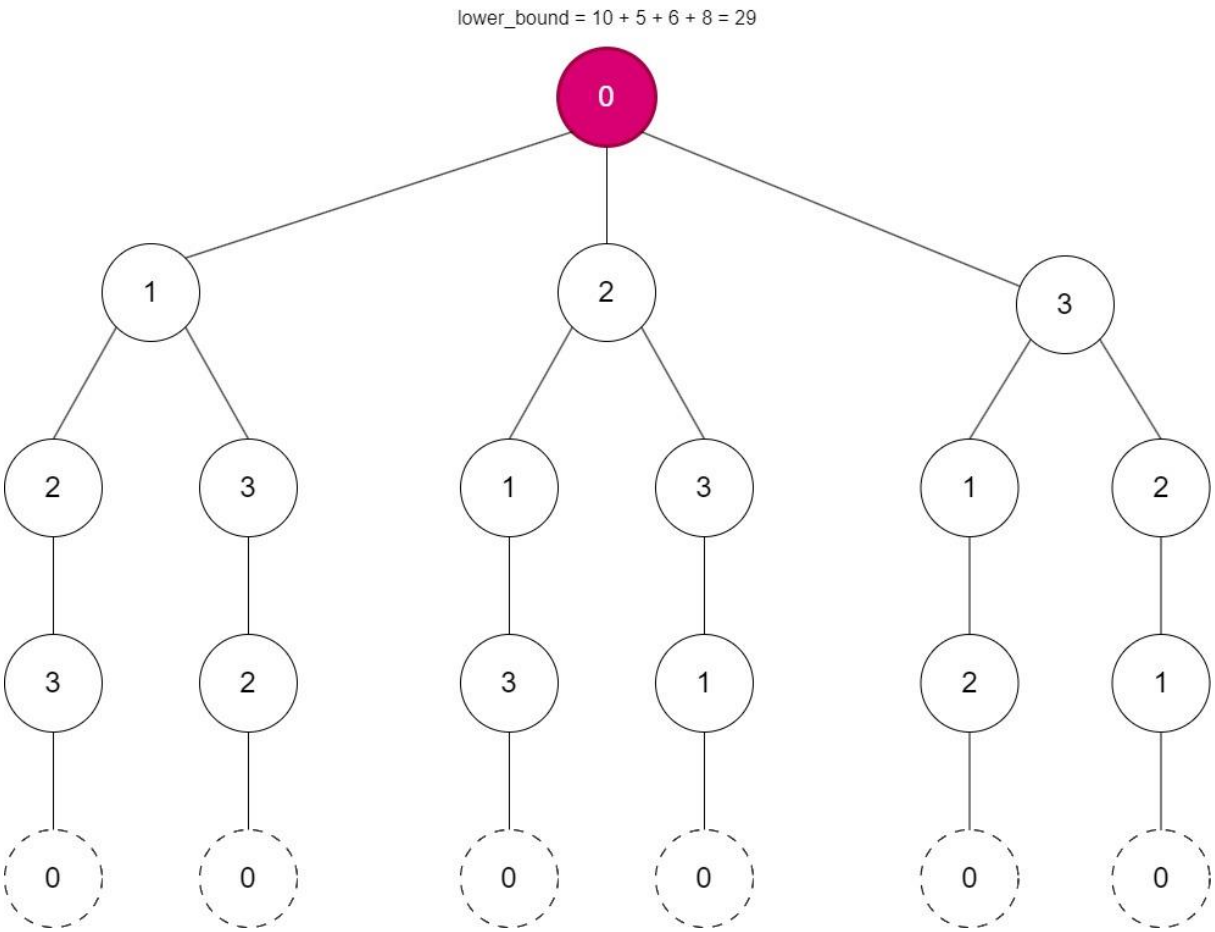
Obliczanie dolnego ograniczenia odbywa się poprzez wybór krawędzi o najniższej wadze z każdego wiersza macierzy reprezentującej graf. Wybrane krawędzie są rejestrowane w tablicy, gdzie ich indeksy odpowiadają wierzchołkom początkowym. Suma wag tych krawędzi definiuje dolne ograniczenie dla wierzchołka początkowego. Ograniczenie to jest następnie aktualizowane i przypisywane do każdego z następników obecnego węzła, zgodnie z określoną regułą.

2.1.2. Przykład.

W strukturze danych projektu znajduje się pełny graf ważony, reprezentowany przez macierz sąsiedztwa. Dodatkowo zaimplementowano tablicę pomocniczą, która przechowuje najniższe wagi krawędzi wychodzących z poszczególnych wierzchołków grafu.

	0	**1**	**2**	**3**	Zawartość tablicy pomocniczej
0	-1	10	15	20	10
1	5	-1	10	9	5
2	6	13	-1	12	6
3	8	8	9	N	8

W procesie obliczania dolnego ograniczenia dla wierzchołka początkowego grafu, przed przebyciem jakiejkolwiek krawędzi, dolne ograniczenie jest wyznaczane jako suma wartości z tablicy pomocniczej, przechowującej najniższe wagi krawędzi wychodzących z każdego wierzchołka.



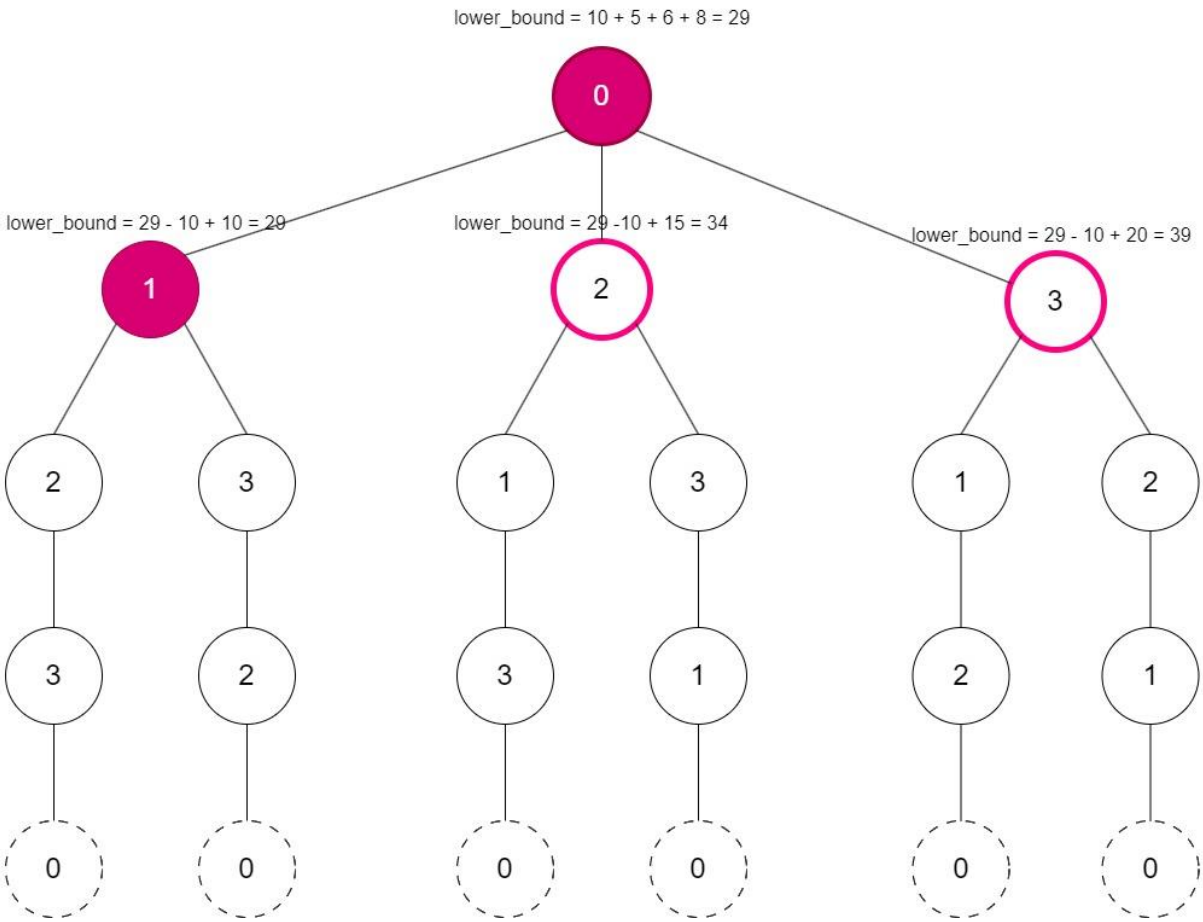
W kolejnym kroku procesu algorytmicznego oblicza się dolne ograniczenia dla następników danego węzła. Aby to zrobić, od wartości dolnego ograniczenia węzła poprzedzającego (poprzednika) odejmuje się wagę odpowiedniej krawędzi z tablicy pomocniczej, odpowiadającą temu poprzednikowi (w tym przypadku o indeksie 0), i dodaje się wagę krawędzi prowadzącej do następnika. Na przykładzie, dla wierzchołka 1, obliczenie to przedstawia się następująco:

$lower\_bound_{01}$ = 29 – helper_array[0] + matrix[0][1] = 29

Zgodnie z tą regułą należy postąpić dla pozostałych następników, zatem dla węzła 2 oraz 3 będzie to kolejno:

$lower\_bound_{02}$ = 29 – helper_array[0] + matrix[0][2] = 34

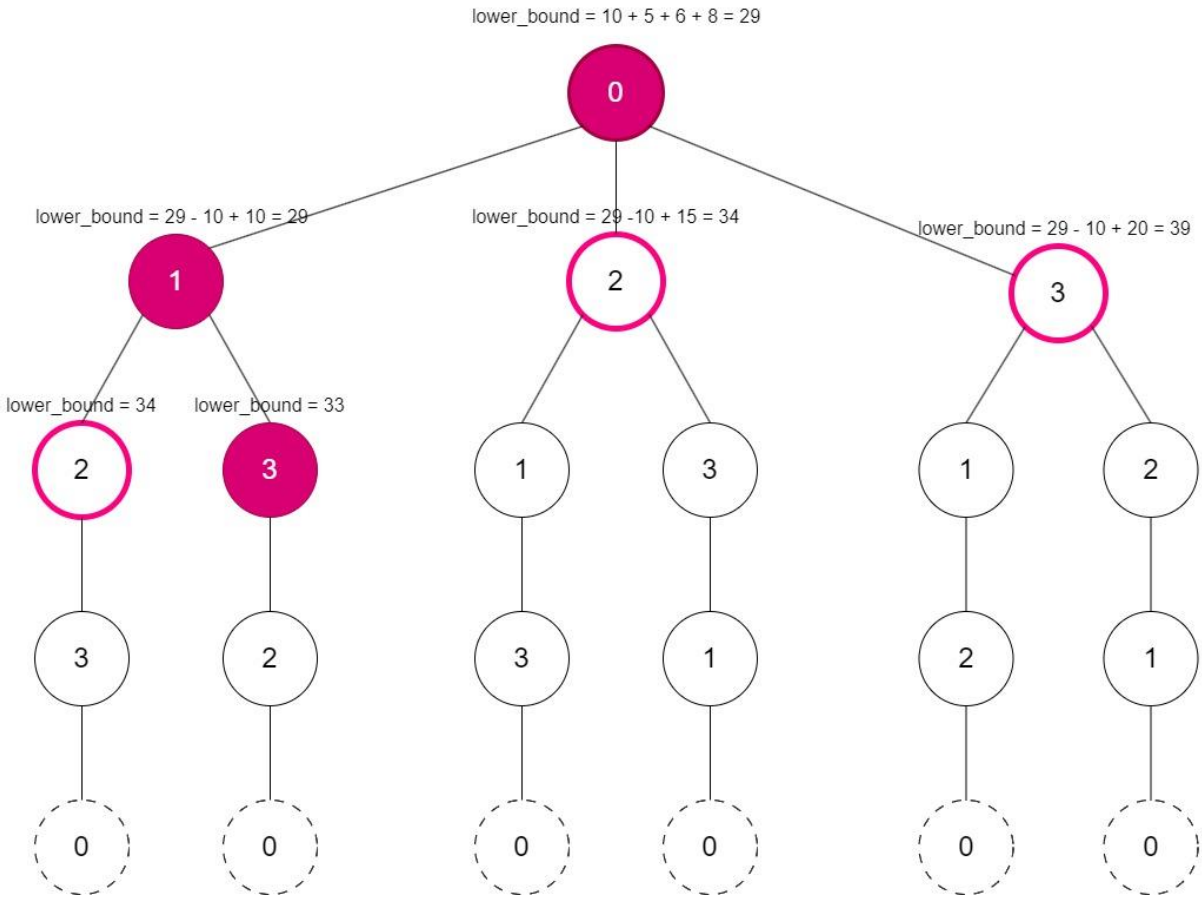
$lower\_bound_{02}$ = 29 – helper_array[0] + matrix[0][3] = 39



Kolejnym etapem algorytmu jest wybór węzła, który ma najniższą wartość dolnego ograniczenia, jednocześnie będącą niższą od aktualnego górnego ograniczenia. Początkowo górne ograniczenie ustawione jest jako nieskończoność, dopóki nie zostanie znalezione pierwsze rozwiązanie. W tej sytuacji należy wybrać węzeł 1 i kontynuować eksplorację gałęzi, którą reprezentuje, analogicznie do przeprowadzonej dla poprzedniego węzła. Dla wskazanego węzła proces ten przedstawia się w następujący sposób:

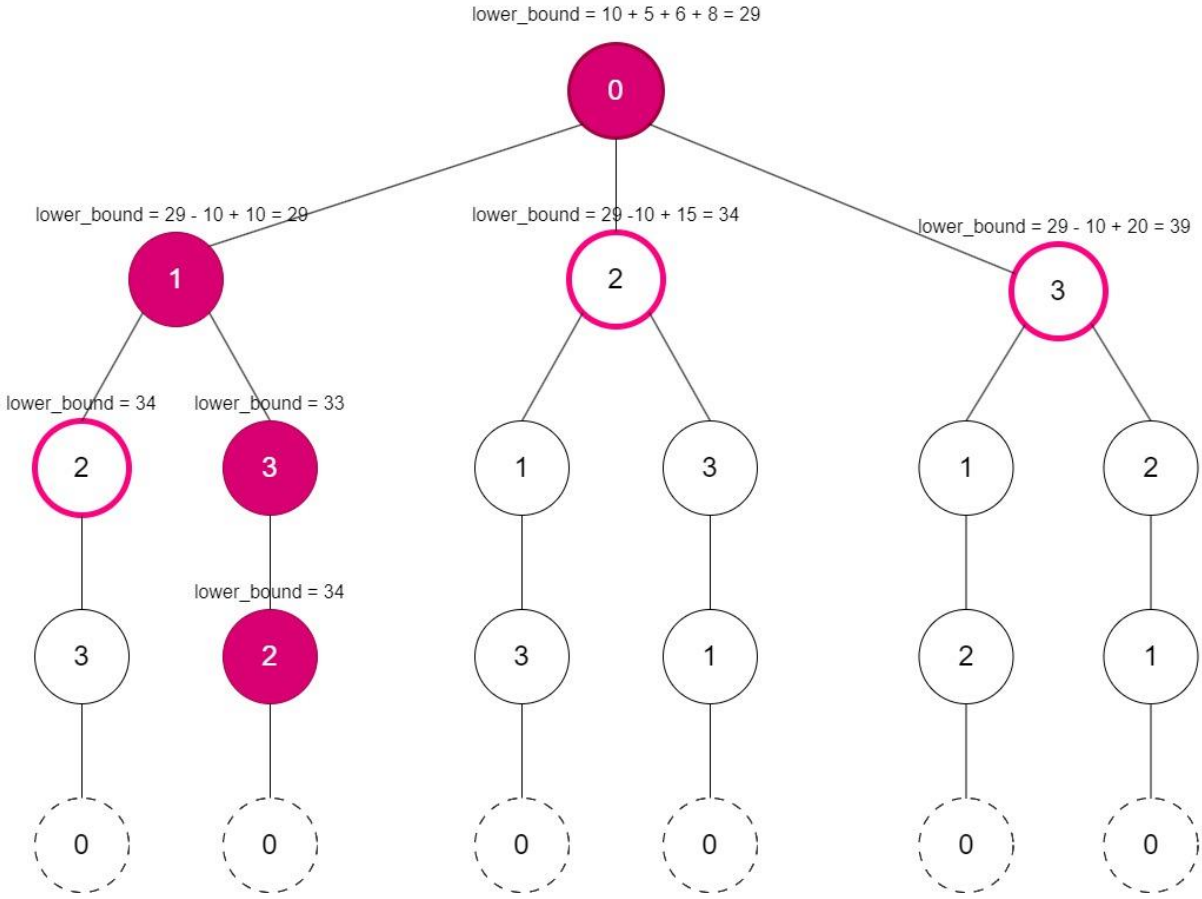
$lower\_bound_{12}$ = 29 – helper_array[1] + matrix[1][2] = 34

$lower\_bound_{13}$ = 29 – helper_array[1] + matrix[1][3] = 33



Do dalszej eksploracji zostaje wybrany węzeł 3. Policzone zostaje ograniczenie dolne dla jego następnika:

$lower\_bound_{32}$ = 33 – helper_array[3] + matrix[3][2] = 34

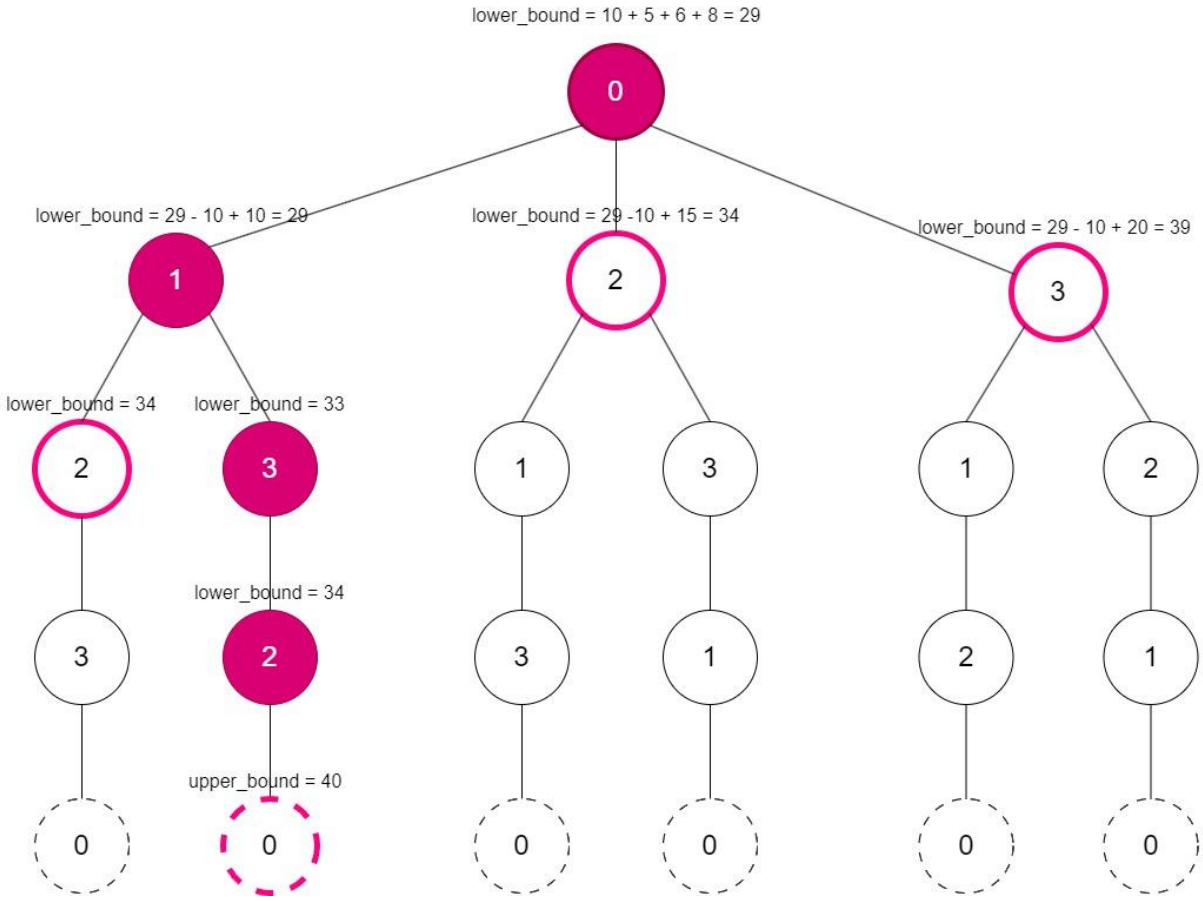


Na etapie, gdy dotarto do węzła 2, który jest liściem drzewa, algorytm osiąga przypadek podstawowy. W tej sytuacji należy sprawdzić, czy suma wag ścieżki, po zamknięciu jej w wierzchołku startowym, będzie mniejsza od obecnego górnego ograniczenia. Dolne ograniczenie dla tej ścieżki ($lower\_bound_{20}$) jest obliczane poprzez odjęcie od 34 wartości z tablicy pomocniczej dla węzła 2 i dodanie wagi krawędzi z węzła 2 do 0:

$lower\_bound_{20} = 34 - helper\_array[2] + matrix[2][0] = 34$

Ponieważ jest to pierwsze znalezione rozwiązanie, spełniony jest warunek lower_bound<upper_bound (dolne ograniczenie mniejsze od górnego). W związku z tym, możliwe jest zaktualizowanie wartości górnego ograniczenia, przypisując mu wartość obliczonego dolnego ograniczenia:

Upper_bound = 40



Po ustaleniu nowego górnego ograniczenia, algorytm kontynuuje eksplorację drzewa, szukając węzłów, które spełniają warunek lower_bound < upper_bound. Eksploracja ta odbywa się analogicznie do wcześniejszych etapów, aż do momentu, gdy żaden z węzłów nie spełni tego kryterium. W takim przypadku następuje zakończenie działania algorytmu.

2.2. Programowanie dynamiczne – Dynamic Programming.

W ramach projektu zastosowano również strategię programowania dynamicznego. Metoda ta bazuje na dekompozycji danego problemu na wielomianową liczbę podproblemów, które są rozwiązywane poczynając od najprostszych, a ich wyniki wykorzystywane są do rozwiązania bardziej skomplikowanych zagadnień. Rekurencja staje się zbędna, ponieważ rozwiązania podproblemów są przechowywane w tablicy stanów. W kontekście problemu komiwojażera, kluczowym założeniem jest fakt, że każda potencjalna ścieżka rozpoczyna i kończy się w tym samym wierzchołku, a kolejność odwiedzania wierzchołków nie jest istotna – liczy się jedynie suma wag przebytych krawędzi.

W związku z tym, pojedynczy stan w problemie komiwojażera definiowany jest jako nieuporządkowany podzbiór odwiedzonych wierzchołków z określonym początkiem i końcem. Ogólna zasada rozwiązywania podproblemu jest następująca:

$d[K][w] = \min\{ d[K\backslash w][v] + l[v][w] \mid v \in K \{w,1\}\}$ dla $w \neq 1$, gdzie:

$d[K][w]$ reprezentuje sumę wag krawędzi tworzących ścieżkę, która rozpoczyna się w wierzchołku pierwszym, przechodzi przez podzbiór wierzchołków K i kończy się w wierzchołku w.
$l[v][w]$ to waga krawędzi łączącej wierzchołek v z w.

2.2.1. Implementacja algorytmu wykorzystującego programowanie dynamiczne.

Kluczowym zadaniem zaimplementowanego algorytmu programowania dynamicznego jest wypełnianie tablicy stanów. Do reprezentacji każdego z nieuporządkowanych podzbiorów wierzchołków zastosowano maski bitowe. Są to słowa bitowe, w których poszczególne bity mają przypisaną wagę odpowiadającą numerom wierzchołków. Bit o wartości jeden wskazuje, że wierzchołek o numerze równym wadze tego bitu jest obecny w reprezentowanym podzbiorze. Bit o wartości zero oznacza, że wierzchołek nie znajduje się w podzbiorze.

Tablica stanów to dwuwymiarowa struktura danych składająca się z 2^n wierszy i n kolumn, gdzie n to liczba wierzchołków w grafie. Liczba wierszy odpowiada ilości możliwych kombinacji (podzbiorów) wierzchołków. Dostęp do poszczególnych wierszy uzyskuje się poprzez maski bitowe reprezentujące odpowiednie podzbiory. Liczba kolumn odpowiada liczbie wierzchołków, do których można przejść z danego podzbioru. Taka struktura tablicy stanów prowadzi do osiągnięcia złożoności obliczeniowej O(n*n*2^n).

W implementacji tego algorytmu wykorzystano następujące struktury danych:

- Tablica Dwuwymiarowa: Przechowuje rozwiązania poszczególnych podproblemów.

3. Plan eksperymentu.

W celu oceny efektywności analizowanych metod rozwiązania problemu komiwojażera, przeprowadzono pomiary czasu wykonania każdego z algorytmów na dziesięciu różnych instancjach problemu, różniących się wielkością. Do generowania grafów wejściowych wykorzystano macierze, które wypełniano losowymi liczbami z zakresu 1 do 100, co zapewniało asymetrię instancji problemu. Elementy znajdujące się na przekątnej macierzy zostały zdefiniowane jako nieskończoności. Rozmiary użytych struktur danych odpowiadały rozmiarom macierzy sąsiedztwa reprezentujących grafy o liczbie wierzchołków od 6 do 20. Pomiary czasu wykonania algorytmów przeprowadzono za pomocą zegara o wysokiej rozdzielczości z biblioteki <chrono>. Wyniki eksperymentu zostaną przedstawione w następnym punkcie, zarówno w formie tabelarycznej, jak i za pomocą wykresów.

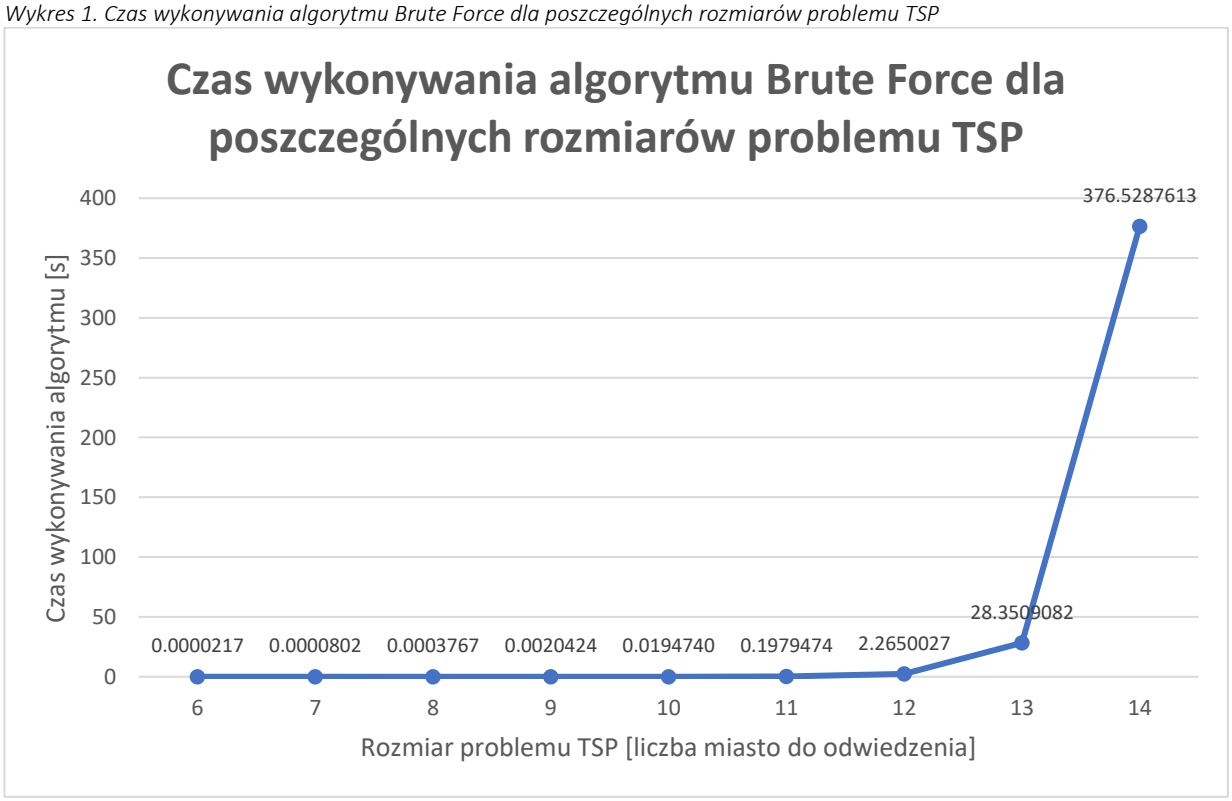
4. Analiza danych pomiarowych.

Tabela 1 Zestawienie czasów pomiarów dla opracowywanych algorytmów

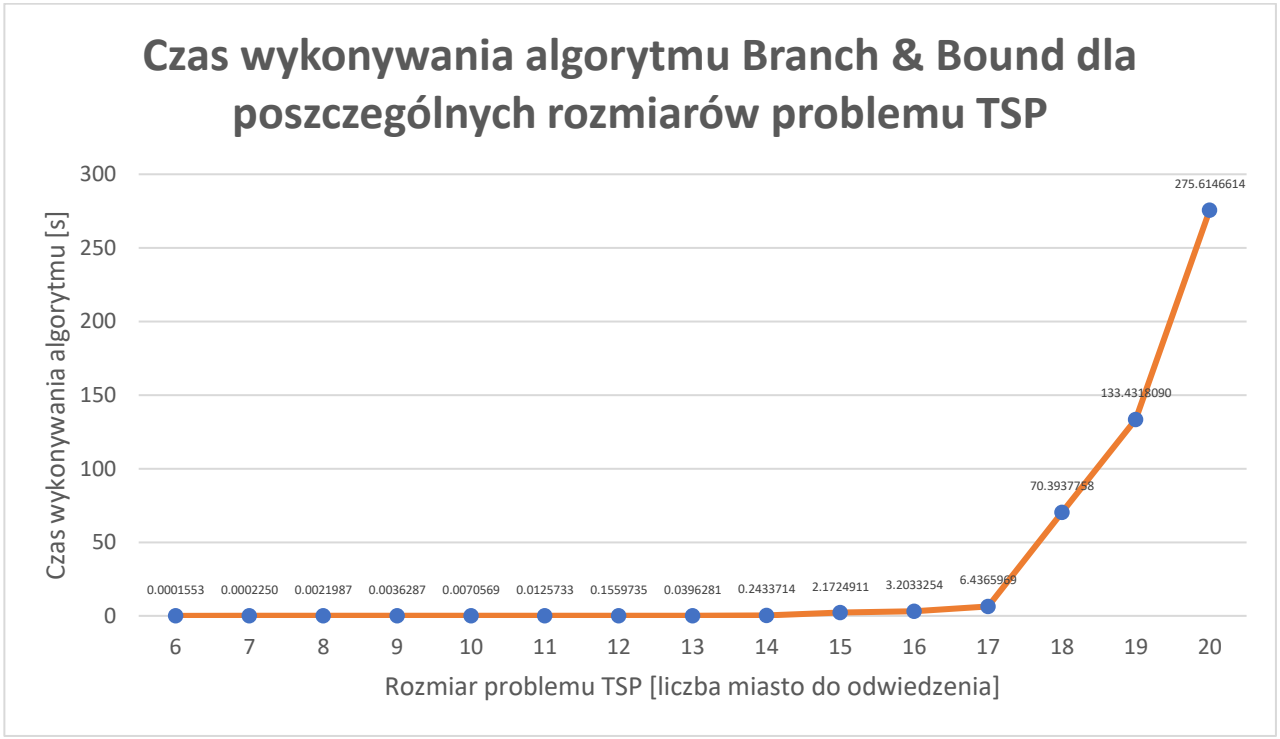
	Czas wykonywania algorytmu [s]		
Rozmiar problemu TSP	Brute Force	Branch & Bound	Dynamic Programming
6	0.0000217	0.0001553	0.0000168
7	0.0000802	0.0002250	0.0000413
8	0.0003767	0.0021987	0.0001080
9	0.0020424	0.0036287	0.0001728
10	0.0194740	0.0070569	0.0003684
11	0.1979474	0.0125733	0.0007648
12	2.2650027	0.1559735	0.0016980
13	28.3509082	0.0396281	0.0038383
14	376.5287613	0.2433714	0.0083850
15	PCW	2.1724911	0.0197929
16	PCW	3.2033254	0.0451019
17	PCW	6.4365969	0.1067257
18	PCW	70.3937758	0.2515264
19	PCW	133.4318090	0.6099978
20	PCW	275.6146614	1.4277571

* PCW – oznacza że przekroczony został dopuszczalny czas wykonania

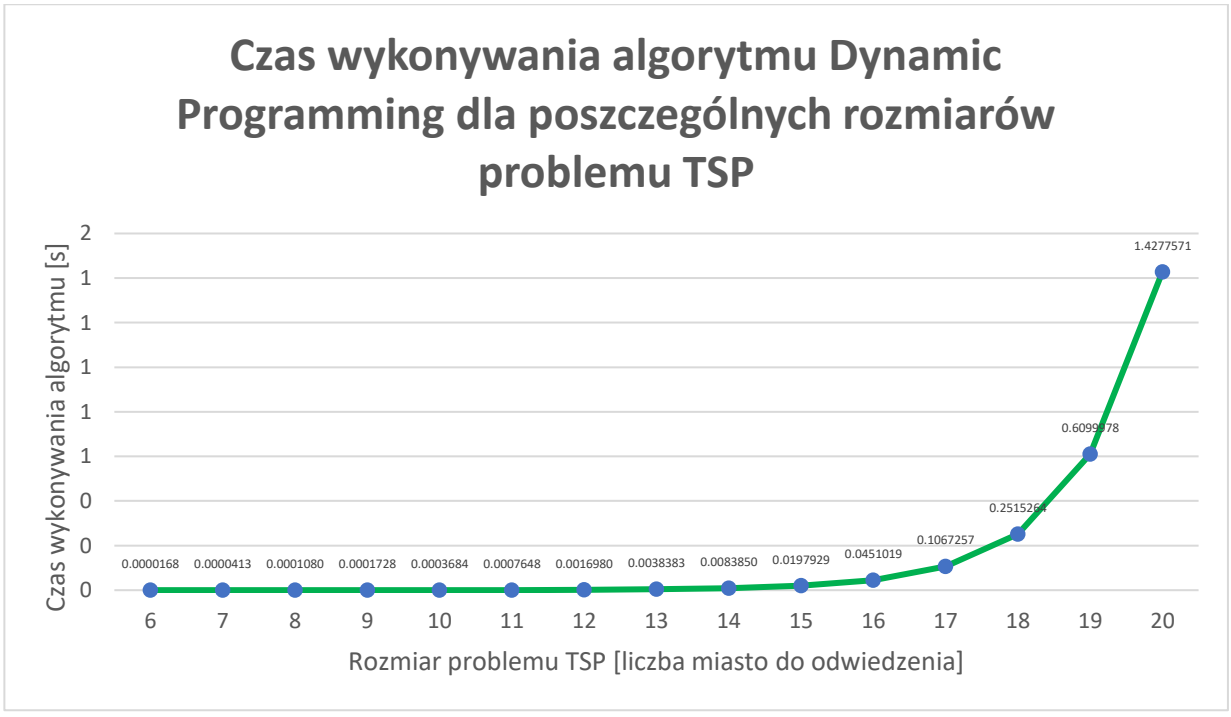
Wykres 1. Czas wykonywania algorytmu Brute Force dla poszczególnych rozmiarów problemu TSP



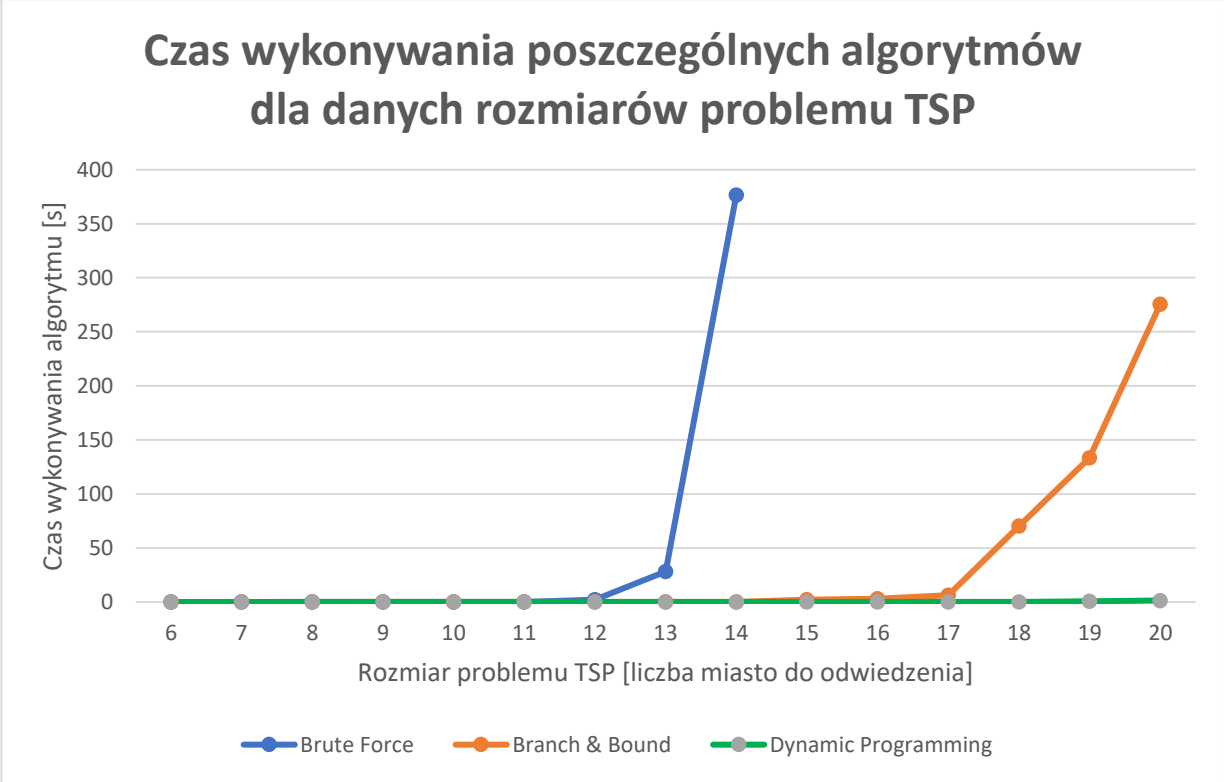
Wykres 2 Czas wykonywania algorytmu Branch & Bound dla poszczególnych rozmiarów problemu TSP



Wykres 3. Czas wykonywania algorytmu Dynamic Programming dla poszczególnych rozmiarów problemu TSP



Wykres 4. Czas wykonywania algorytmu Dynamic Programming dla poszczególnych rozmiarów problemu TSP



5. Wnioski.

Analiza wyników pomiarów potwierdza teoretyczne klasy złożoności obliczeniowej przewidziane dla każdej z rozpatrywanych metod. Zaimplementowany algorytm przeglądu zupełnego (Brute Force) wykazał przewidywaną wykładniczą złożoność obliczeniową O(n!), co zostało potwierdzone przez eksperymentalne wyniki. Dla instancji problemu komiwojażera o liczbie miast przekraczającej 14, pomiary czasu wykonania algorytmu zostały przerwane z powodu zbyt długiego czasu obliczeń. Wynika z tego, że metoda ta jest efektywna jedynie dla małych instancji problemu.

Algorytm bazujący na metodzie podziału i ograniczeń (Branch & Bound) również charakteryzuje się teoretyczną wykładniczą złożonością obliczeniową O(n!). Dzięki zastosowaniu technik rozgałęziania oraz ograniczania przeglądu możliwych rozwiązań, a także odpowiedniego doboru funkcji obliczającej dolne ograniczenie, czas potrzebny do znalezienia rozwiązania w badanych instancjach został znacząco zredukowany w porównaniu do metody Brute Force. Jednakże sam czas wykonywania algorytmu uzależniony jest również w znaczący sposób od rozkładu badanego grafu.

Metoda programowania dynamicznego wykazała zgodność z teoretyczną klasą złożoności O(n*2^n). Dzięki ograniczeniu algorytmu do procedury wypełniania tablicy stanów, okazał się on najbardziej efektywny z testowanych. Uzyskane pomiary czasu wykonania dla tej metody były znacznie niższe niż w przypadku metody Branch & Bound

6. Bibliografia.

1. https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met_podz_ogr.opr.pdf
2. http://cs.pwr.edu.pl/zielinski/lectures/om/mow10.pdf
3. https://www.youtube.com/watch?v=-cLsEHP0qt0
4. https://www.youtube.com/watch?v=nN4K8xA8ShM&t=927s
5. https://www.youtube.com/watch?v=JE0JE8ce1V0
6. https://www.youtube.com/watch?v=XaXsJJh-Q5Y