Raport 1

Przedmiot: Projektowanie Efektywnych algorytmów - Projekt

Data złożenia: 21.11.2024

Temat zadania: Rozwiązanie problemu komiwojażera (TSP)

Treść zadania:
1. Rozwiązanie problemu komiwojażera (TSP – Travelling Salesman Problem) metodami: Brute-force, nearest-neighbour, random, DFS, BFS oraz Lowest Cost.

Piotr Strycharczyk, 272995

Punkt 1 - Opis problemu

1. Problem komiwojażera.

Zadanie polega na znalezieniu ścieżki, dla której suma wag będzie miała minimalną wartość w grafie, tak aby każdy wierzchołek został odwiedzony dokładnie raz. Dodatkowym założeniem jest fakt, że testowane grafy posiadają cykle Hamiltona. Ścieżka ma się zakończyć w tym samym wierzchołku, w którym zaczynaliśmy naszą podróż.

2. Jakie wyniki chcemy uzyskać

Wyniki, które chcemy uzyskać mają być dokładnym rozwiązaniem problemu komiwojażera lub w przypadku algorytmów takich jak nearest-neighbour i random jak najbliższe dokładnemu rozwiązaniu. Dodatkowo chcemy aby wyniki otrzymane w wyniku algorytmów były uzyskiwane w jak najkrótszym czasie.

3. Implementacja wybranych metod rozwiązywania problemu TSP:

W każdej z moich metod posiadam implementację zegara, który sprawdza czy nie upłynął maksymalny czas jaki ma wykonywać się dany algorytm – 30 minut, jeśli tak się stanie to algorytm jest niezwłocznie kończony.

a. Metoda brute-force
   Do generowania permutacji zbioru wierzchołków grafu korzystam w niej z funkcji next_permutation, która jest dostępną funkcją z biblioteki algorithm. Funkcja ta przeszukuje wektor od końca, szukając pierwszego elementu, który jest mniejszy od swojego sąsiada po prawej stronie. Elementy te są zamieniane miejscami i później, wszystkie elementy na prawo od miejsca zamiany są sortowane w porządku rosnącym (poprzez odwrócenie kolejności). Gdy zbiór będzie posortowany w porządu malejącym to oznacza że osiągnięto już każdą możliwą permutację zbioru.
b. Metoda nearest-neighbour
   W mojej implmentacji algorytm nn startuje od każdego wierzchołka grafu czyniąc go bardziej efektywnym w porównaniu do wersji, w której algorytm startuje od sztywno ustalonego, bądź losowego wierzchołka. Dodatkowo gdy z danego wierzchołka wychodzą krawędzie o tych samych wagach, to algorytm rekurencyjnie sprawdzi wszystkie możliwe opcje.
c. Metoda random
   służy do rozwiązania problemu komiwojażera (TSP) poprzez podejście heurystyczne oparte na losowym generowaniu ścieżek w grafie. Dane losowania ścieżek mogą się powtarzać – nie sprawdzam, czy dana permutacja została już wygenerowana. Metoda korzysta z żywa generatora liczb pseudolosowych, który jest inicjalizowany za pomocą wartości pobranej z urządzenia losowego.
   Do wyboru wierzchołka z listy wykorzystywany jest rozkład równomierny, co zapewnia jednakowe prawdopodobieństwo dla każdego dostępnego wierzchołka. W moim algorytmie obiekt std::random_device służy do inicjalizacji generatora pseudolosowego std::mt19937, który jest jednym z najlepszych dostępnych generatorów, charakteryzującym się bardzo długim okresem (2^19937 − 1), szybkim działaniem i doskonałą jakością generowanych liczb losowych.
d. DFS

W mojej implementacji działa rekurencyjnie poprzez „schodzenie” na coraz to niższe poziomy grafu i jednoczesne sprawdzanie przy każdym zejściu czy waga obecnej ścieżki jest większa od górnego ograniczenia, jeśli tak się zdarzy to algorytm odcina daną gałąź i nie rozwija jej dalej co przyśpiesza algorytm. Dodatkowo w przypadku górnego ograniczenia zamiast przypisywać mu początkowo bardzo wysoką wartość (INT_MAX), to ja wykonuję algorytm nearest-neighbour i przypisuje jego wynik do właśnie do niego by jeszcze szybciej odcinać gałęzie.

e. BFS
Zamiast działać rekurencyjnie korzysta z kolejki i przeszukuje graf po szerokości. W kolejce przechowuję strukturę State – przechowującą informację o tym na jakim poziomie w danej chwili jesteśmy, obecne miasto, które odwiedzamy, dotychczasowy koszt by porównywać go z górnym ograniczeniem oraz ścieżkę – gdyby to rozwiązanie okazało się tym optymalnym. W miarę przeszukiwania grafu kolejka posiada coraz więcej stanów ale i może niektóre stany odrzucać(gdy obecny koszt ścieżki przekroczy górne ogarniczenie). Ostatecznie dzieje się tak że kolejka jest pusta a my znaleźliśmy optymalne rozwiązanie. Kolejka nie działa priorytetowo jak ma to miejsce w Lowest Cost tylko jest to kolejka FIFO. Po kolei przeszukiwane są kolejne wierzchołki grafu.

f. Lowest Cost
Korzysta z kolejki priorytetowej. W odróżnieniu od wersji BFS, zamiast przeszukiwać graf w sposób równomierny poziomami, metoda ta zawsze najpierw eksploruje stany o najniższym koszcie dotychczasowej ścieżki. Dzięki temu algorytm bardziej skoncentrowany jest na potencjalnie najlepszych rozwiązaniach i szybciej może odrzucać mniej obiecujące stany. Struktura, którą przechowuje w kolejce poza tym, że posiada priorytet nie różni się niczym.

4. Generowanie instancji

Moje instancje symetryczne są wygenerowane w sposób losowy i takie, że $10 \leq Waga\ krawędzi \leq 40$. Są również takie, które pochodzą ze biblioteki TSPlib oraz kilka grafów asymetrycznych.

Dodatkowo instancje, które są symetryczne (jest ich dokładnie dziewięć) oraz $5 \leq V \leq 13, V \in \mathbb{C}$. Czas potrzebny do rozwiązania problemu komiwojażera dla V = 5 jest niski aczkolwiek zauważalny i dlatego od takiej instacji zdecydowałem się badać grafy. Natomiast dla instancji o V = 14 czas trwania algorytmu np. brute-force to ponad 30min, dlatego testowałem grafy do V = 13. Dla algorytmu nn jest możliwe przetestowanie większych instancji grafów ze względu na jego mniejszą złożoność obliczeniową w porównaniu do pozostałych algorytmów.

5. Hipotezy badawcze:

1. Czas wykonania algorytmów wzrośnie wykładniczo wraz z liczbą wierzchołków, dla algorytmów brute-force i random oraz wszystkich algorytmów Branch and Bound.
2. Algorytmy o złożoności $O(n!)$ nie nadają się do testowania dużych instancji ze względu na złożoność obliczeniową.
3. Metoda nearest neighbour (startowana od każdego wierzchołka) o złożoności $O(n^3)$może być wykonywana dla znacznie większych instancji niż pozostałe algorytmy.
4. Algorytmy DFS, BFS oraz Lowest Cost, ze względu na swoje strategie eksploracji przestrzeni rozwiązań, osiągają wyniki w czasie wzrastającym wykładniczo, jednak szybciej od brute-force.

Punkt 2 – instancje i specyfikacja

TABELA NR.1.1 ORAZ 1.2 PRZEDSTAWIA NAZWĘ INSTANCJI DLA GRAFU SYMETRYCZNEGO I ASYMETRYCZNEGO, LICZBĘ WIERZCHOŁKÓW ORAZ WARTOŚĆ OPTYMALNĄ.

Nazwa instancji(sym)	Liczba wierzchołków	Wartość optymalna
file_5.txt	5	98
file_6.txt	6	118
file_7.txt	7	90
file_8.txt	8	115
file_9.txt	9	105
file_10.txt	10	126
file_11.txt	11	156
file_12.txt	12	177
file_13.txt	13	174

Nazwa instancji (asym)	Liczba wierzchołków	Wartość optymalna
file_5a.txt	5	200
file_10a.txt	10	1985
data34a.txt	34	1286
data65a.txt(tsplib ftv64)	65	1839
data100a.txt(tsplib kro 124)	100	36230
data140a.txt	140	-

Specyfikacja sprzętowa:

Procesor: AMD Ryzen 5 5500U

Liczba rdzeni: 6

Maksymalne taktowanie zegara: 4,2 GHz

Wielkość pamięci cache: 19MB

Wielkość pamięci RAM: 16GB

Prezentacja programu odbywała się na tym samym sprzęcie, co jego testy.

Laptop miał odpalone testy podczas zasilania go prądem.

W trakcie działania testów nie działały inne programy na komputerze.

Punkt 3 – Opis procedury badawczej

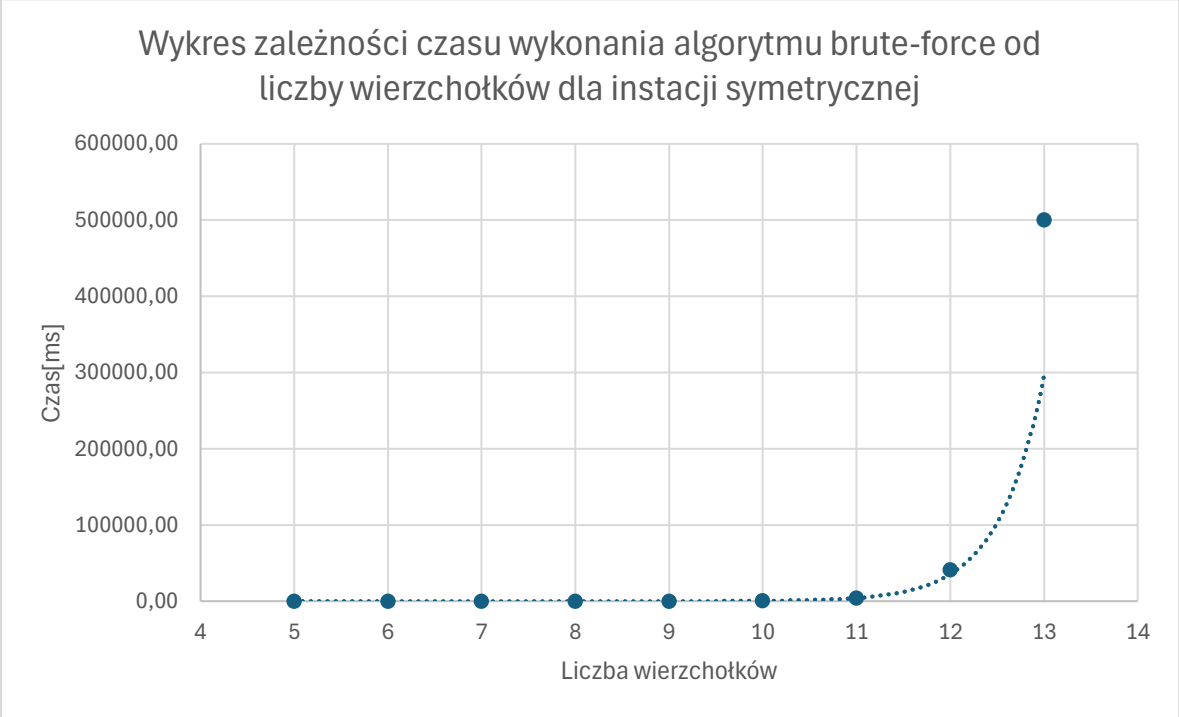
Ilość badanych instancji oraz liczba iteracji:

W moich badaniach, w których używałem 9 instancji grafów(symetrycznych) na każdy algorytm miałem ustawiony limit czasowy 30 minut oraz limit iteracji 100. Jeśli Po 30 minutach wszystkie iteracje się nie wykonały, to program zliczał średnią tylko z tylu iteracji ile się wykonało w zadanym czasie – 30 minut. Grafy generowane dla instancji symetrycznych miały takie same rozpiętości wag ($10 \leq Waga\ krawędzi \leq 40$), tak aby porównanie miało większy sens. Dla instancji asymetrycznych podanych w tabelce 1.2, które były znacząco większe od tych z tabelki 1.1 wykonałem jeszcze testy dla algorytmu nn. Dla algorytmów BnB większe rozpiętości wag krawędzi powodują szybsze działanie tych algorytmów, ponieważ algorytm może wcześniej „odciąć" mało obiecującą gałąź i w konsekwencji zakończyć algorytm szybciej z powodu zmniejszenia obliczeń.

Punkt 4 – Wyniki badań

TABELA 2 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU BRUTE-FORCE DLA INSTANCJI SYMETRYCZNYCH

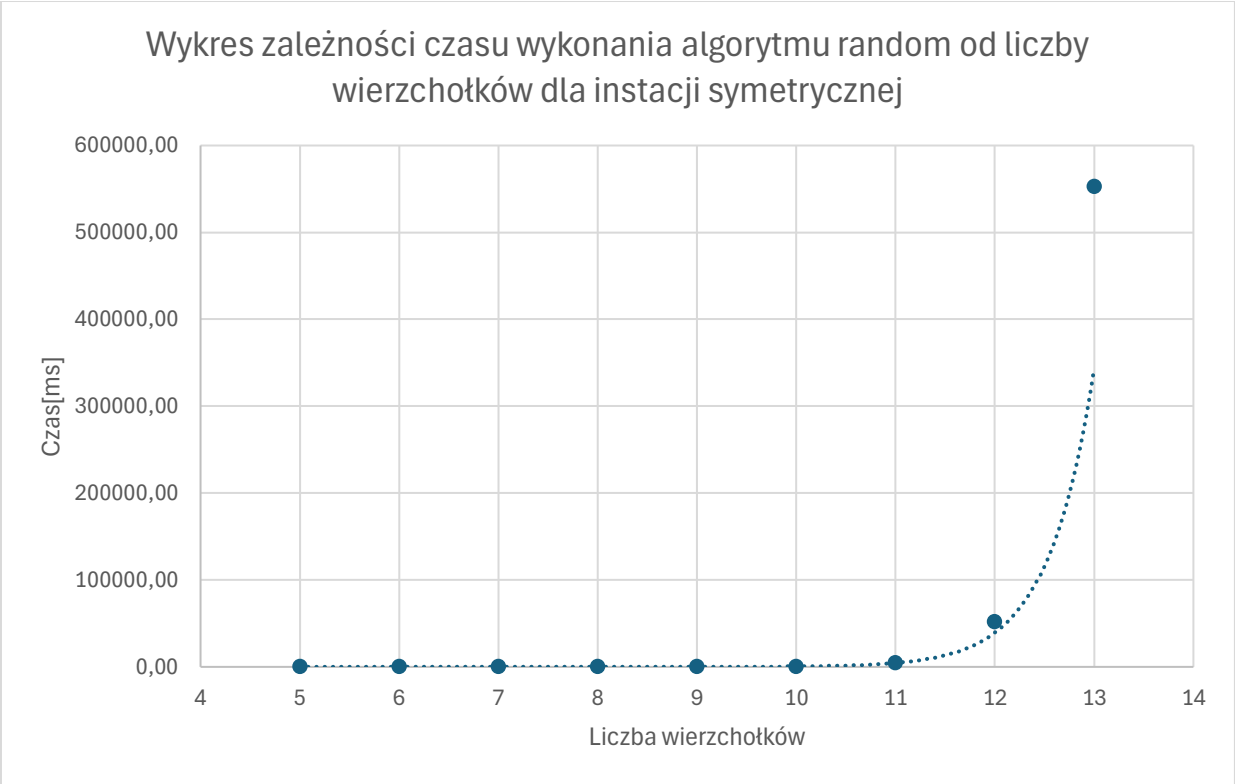
liczba wierzchołków	czas[ms]
5	0,03
6	0,11
7	0,60
8	5,01
9	39,81
10	386,77
11	3745,94
12	40909,10
13	500137



WYKRES 1 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU BRUTE-FORCE OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 3 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU RANDOM DLA INSTANCJI SYMETRYCZNYCH

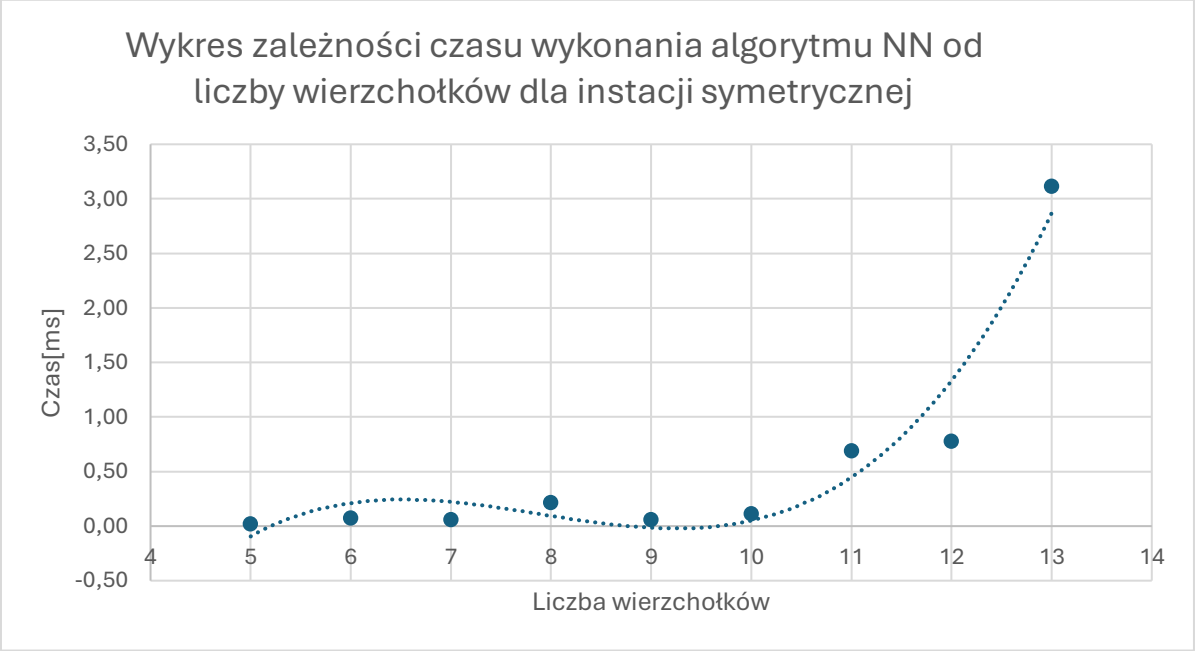
liczba wierzchołków	czas[ms]
5	0,04
6	0,07
7	0,29
8	2,77
9	67,18
10	383,62
11	4342,47
12	51428,60
13	552333,30



WYKRES 2 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU RANDOM OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 4 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU NN DLA INSTANCJI SYMETRYCZNYCH

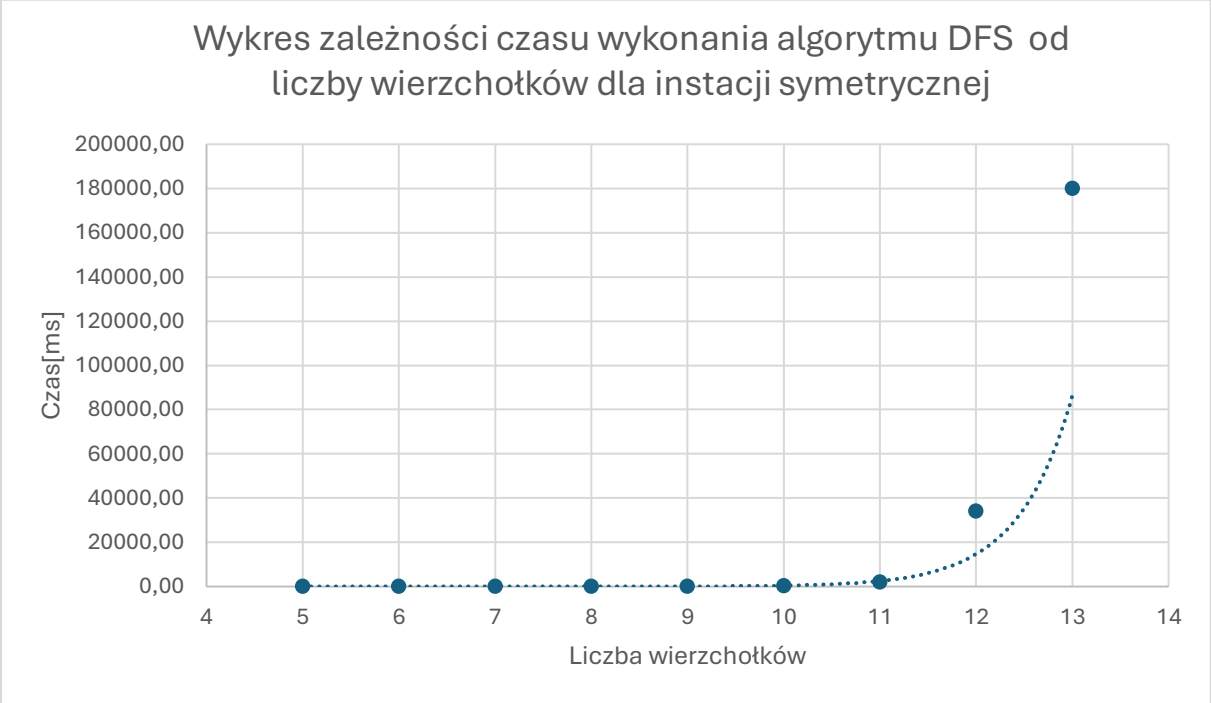
liczba wierzchołków	czas[ms]
5	0,02
6	0,07
7	0,06
8	0,21
9	0,06
10	0,11
11	0,69
12	0,78
13	3,11



WYKRES 3 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU NN OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 5 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU DFS DLA INSTANCJI SYMETRYCZNYCH

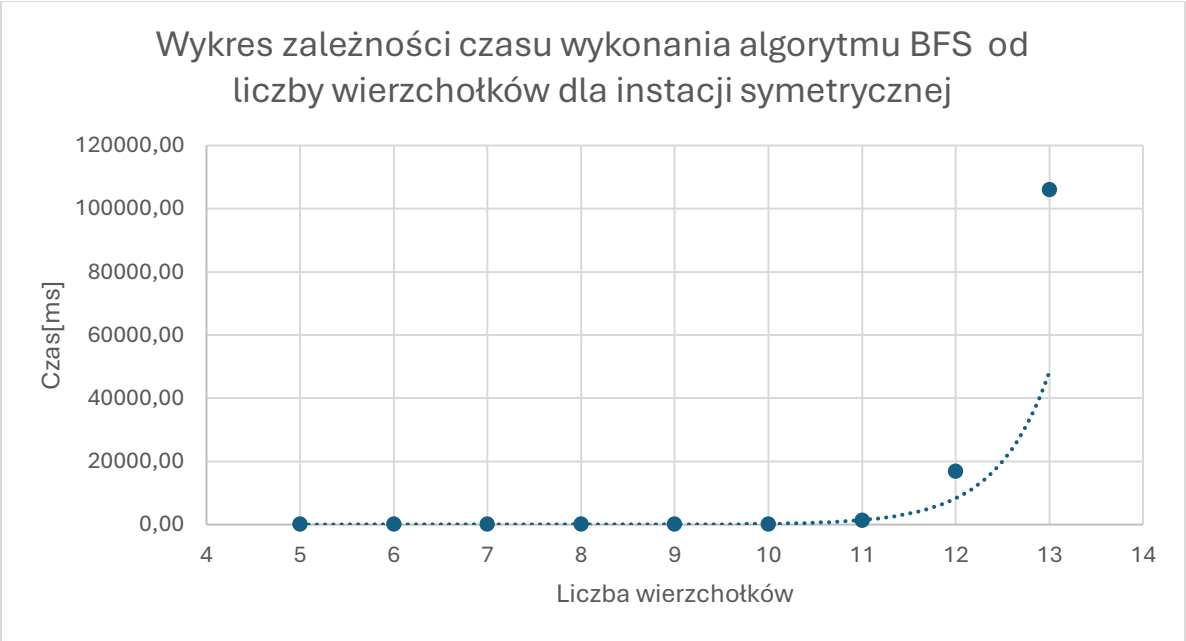
liczba wierzchołków	czas[ms]
5	0,11
6	0,83
7	1,58
8	8,26
9	21,37
10	156,43
11	2068,75
12	33962,30
13	180000



WYKRES 4 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU DFS OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 6 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU BFS DLA INSTANCJI SYMETRYCZNYCH

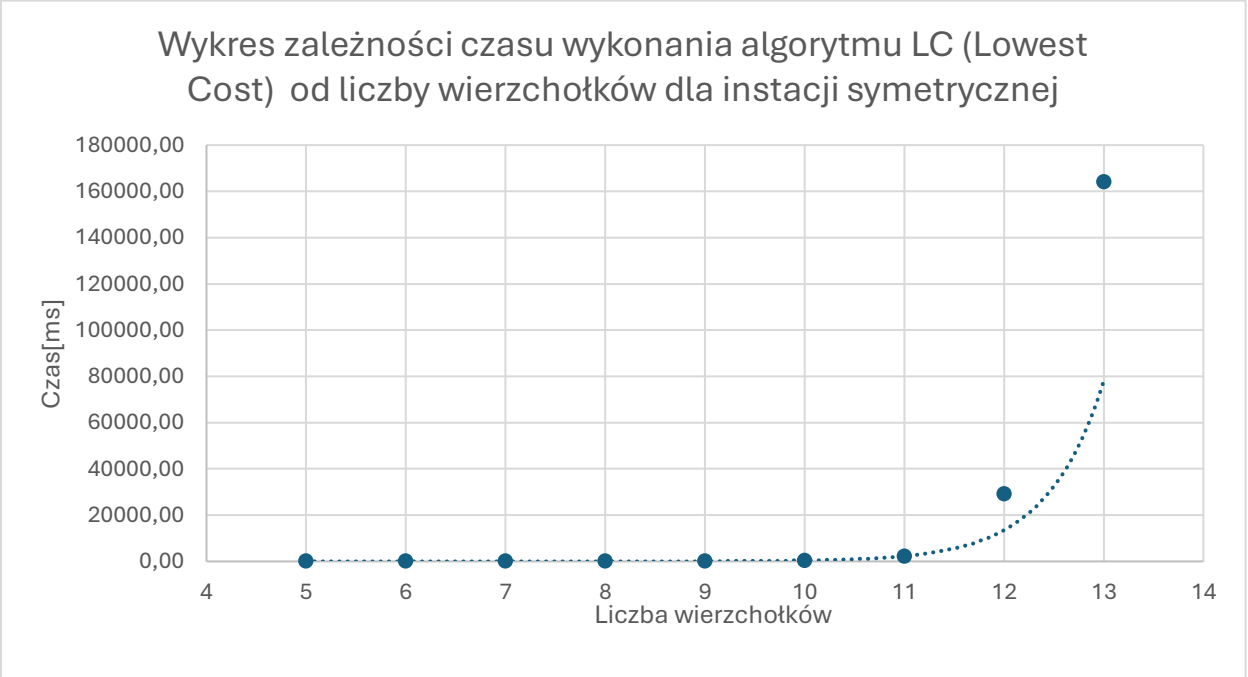
liczba wierzchołków	czas[ms]
5	0,07
6	0,44
7	1,09
8	5,69
9	11,84
10	94,14
11	1274,80
12	16710,10
13	105993



WYKRES 5 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU BFS OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 7 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU LC DLA INSTANCJI SYMETRYCZNYCH

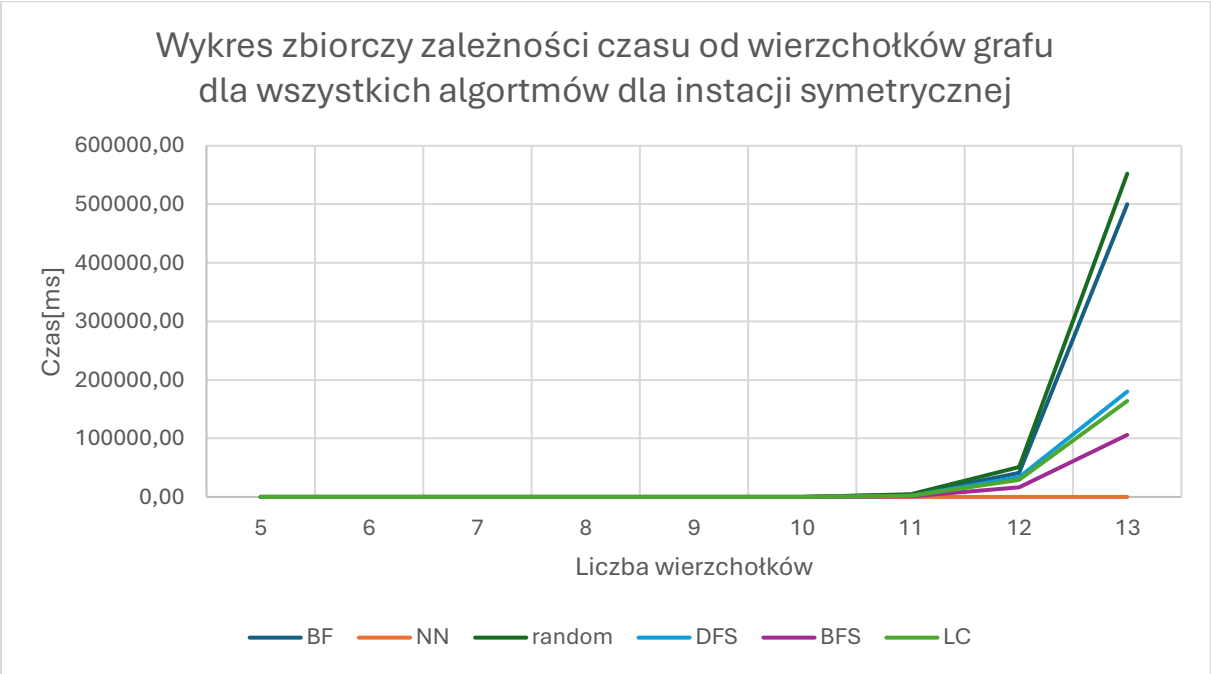
liczba wierzchołków	czas[ms]
5	0,11
6	0,71
7	1,67
8	8,70
9	18,27
10	155,23
11	2023,09
12	29036
13	163988



WYKRES 6 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU LC OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 8 PRZEDSTAWIA CZAS WYKONANIA WSZYSTKICH ALGORYTMÓW DLA INSTANCJI SYMETRYCZNYCH

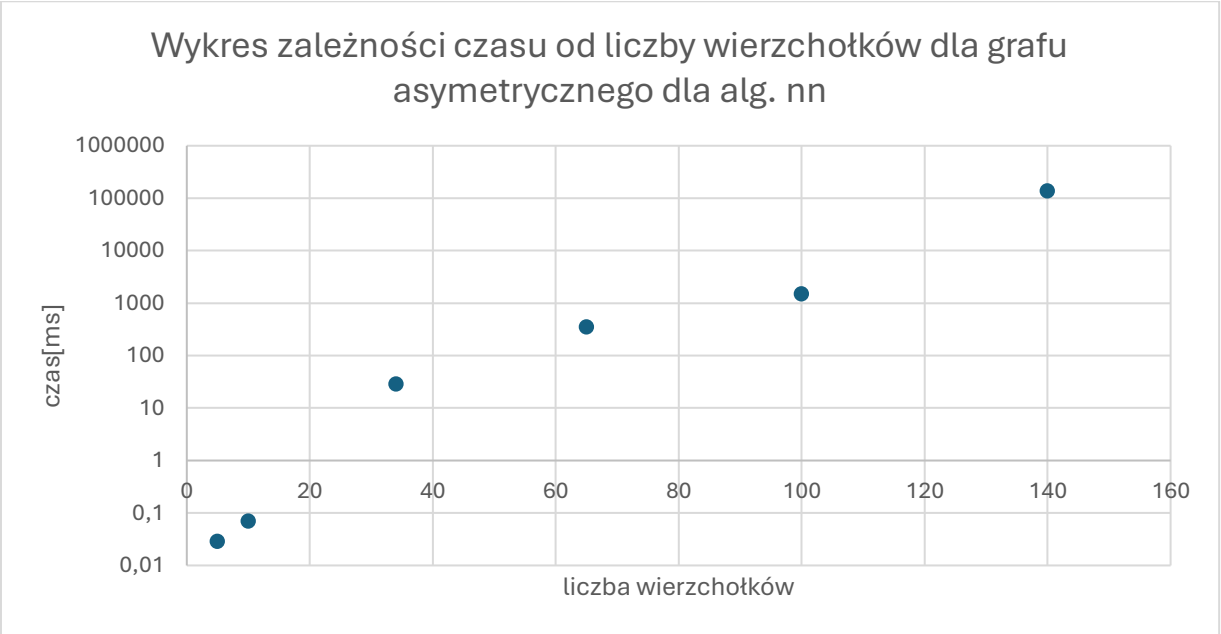
Liczba wierzchołków	Czas [ms]					
	Brute-force	Nearest-neighbour	random	DFS	BFS	LC
5	0,03	0,02	0,04	0,11	0,07	0,11
6	0,11	0,07	0,07	0,83	0,44	0,71
7	0,60	0,06	0,29	1,58	1,09	1,67
8	5,01	0,21	2,77	8,26	5,69	8,70
9	39,81	0,06	67,18	21,37	11,84	18,27
10	386,77	0,11	383,62	156,43	94,14	155,23
11	3745,94	0,69	4342,47	2068,75	1274,80	2023,09
12	40909,10	0,78	51428,60	33962,30	16710,10	29036
13	500137	3,11	552333,3	180000	105993	163988



WYKRES 7 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA WSZYSTKICH ALGORYTMÓW OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI SYMETRYCZNEJ

TABELA 9 PRZEDSTAWIA CZAS WYKONANIA ALGORYTMU NN DLA INSTANCJI ASYMETRYCZNYCH W ZALEŻNOŚCI OD LICZBY WIERZCHOŁKÓW

liczba wierzchołków	czas[ms]
5	0,02833
10	0,069513
34	28,5336
65	346,866
100	1487,27
140	135251



WYKRES 8 PRZEDSTAWIA ZALEŻNOŚĆ CZASU WYKONANIA ALGORYTMU NN OD LICZBY WIERZCHOŁKÓW DLA INSTANCJI ASYMETRYCZNEJ (SKALA LOGARYTMICZNA)

Punkt 5 - Analiza wyników

Na podstawie wyników przedstawionych w sprawozdaniu, w szczególności w tabelach i wykresach, można dokonać szczegółowej analizy efektywności algorytmów i ich zastosowań w zależności od problemu i liczby wierzchołków. Analiza ta odnosi się bezpośrednio do hipotez badawczych, a na koniec przedstawione zostaną wnioski.

W tabeli 2 przedstawiono czasy działania algorytmu brute-force dla różnych liczności wierzchołków. Analiza wyników pokazuje, że dla grafów o liczbie wierzchołków powyżej 10 czas działania gwałtownie rośnie, co jest zgodne z oczekiwaniami wynikającymi ze złożoności obliczeniowej tego algorytmu O(n!). Przykładowo, w tabeli 2, dla 6 wierzchołków czas wynosi 0,11ms, natomiast dla 10 wierzchołków już prawie 0,4s. To potwierdza hipotezę 1, że brute-force wykazuje znaczący wzrost czasu działania dla większych instancji, jeszcze lepiej ten wzrost pokazuje wykres nr 1.

Algorytm nearest-neighbour (tabela 4, wykres 3) prezentuje znacznie bardziej zrównoważone wyniki. Czas działania, niezależnie od liczby wierzchołków, pozostaje w przedziale do kilku milisekund. Przykładowo, dla 10 wierzchołków czas wynosi jedynie 0.11ms. Jest to wynik znacznie lepszy w porównaniu z brute-force, co potwierdza hipotezę 3 dotyczącą efektywności algorytmów heurystycznych. Dodatkowo w tabeli 9 i na wykresie nr 8 zostały przedstawione wyniki działania algorytmu NN (dla instancji asymetrycznych) dla znacznie większych grafów (największy 140 wierzchołków) i dopiero tam widać większy wzrost czasu wykonania algorytmu dla większych instancji, co potwierdza hipotezę nr 3, że metoda nn przydaje się do testowania większych grafów, lecz kosztem tego, że nie otrzymamy rozwiązania optymalnego.

Z kolei algorytm BFS (tabela 6, wykres 5) ma czasy działania, które wzrastają wykładniczo, jednak wolniej niż brute-force. Zastosowanie BFS jest bardziej praktyczne dla nieco większych instacji, co potwierdza jego użyteczność w stosunku do brute-force. Podobnie jest z pozostałymi algorytmami BnB.

Algorytmy BnB pokazują w tabeli nr 8 i na wykresie 7, że są one nieco lepsze niż algorytm brute-force, co potwierdza hipotezę badawczą nr 4.

Algorytm random wykazuje, zgodnie z danymi z tabeli 3 i wykresu 2, na nieco gorsze czasy niż algorytm brute-force, co czyni go najmniej efektywnym algorytmem z wszystkich tu testowanych. Natomiast najlepszym pod względem czasowym okazał się algorytm BFS dla każdej z testowanych instancji symetrycznych.

Wyniki prezentowane w tabelach i wykresach pozwalają wyciągnąć następujące wnioski:

Wszystkie algorytmy poza nn nie nadają się do testowania grafów większych niż 13 wierzchołków w rozsądnym czasie (30minut), co potwierdza hipotezę nr 2. Nawet algorytmy BnB, które są „ulepszeniem" brute-forca mają problem aby wykonać się w rozsądnym czasie dla nieco większych instancji niż dla brute-force. NN natomiast może być stosowany aby znaleźć ścieżkę (niekoniecznie optymalną) dla dużych instacji np. powyżej 100 wierzchołków.

Punkt 6 – Bibliografia

1. http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/
2. dr inż. Tomasz Kapłon „Projektowanie efektywnych algorytmów” Wykład, Politechnika Wrocławska, semestr zimowy 2024/2025
3. https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/
4. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
5. https://en.wikipedia.org/wiki/Travelling_salesman_problem