POLITECHNIKA WROCŁAWSKA

Projektowanie Efektywnych Algorytmów

Sprawozdanie z Etapu 1

Implementacja algorytmów dokładnych dla problemu komiwojażera.

Wykonał:	**Maksymilian Guździoł, 233534**
Termin:	**WT 18.55-20.35**
Data:	**13.11.2018r**
Prowadzący:	**Dr inż. Jarosław Rudy**
Ocena:	

Uwagi prowadzącego:

Wstęp

Problem komiwojażera jest problemem optymalizacyjnym, polegającym na odnalezieniu minimalnego cyklu Hamiltona w grafie pełnym ważonym. Nazwa pochodzi od typowej ilustracji problemu: dane jest n miast, które muszą zostać odwiedzone przez komiwojażera oraz odległości między nimi. Zadaniem jest znalezienie najkrótszej drogi przechodzącej przez wszystkie miasta oraz wracającej do miasta początkowego. Problem komiwojażera jest klasy NP-trudny. W realizacji projektu wykorzystane zostały 3 metody rozwiązywania: przegląd zupełny, programowanie dynamiczne oraz metoda podziału i ograniczeń.

Programowanie dynamiczne

Programowanie dynamiczne polega na podziale problemu na mniejsze podproblemy oraz rozwiązaniu wszystkich podproblemów zaczynając od „najmniejszego, a kończąc na największym, zapisując optymalne wartości rozwiązań w tablicy. Skutkuje to rozwiązywaniem każdego podproblemu jedynie raz, w efekcie czego złożoność obliczeniowa maleje kosztem wzrostu złożoności pamięciowej algorytmu. Algorytm dynamiczny zastosowany dla problemu komiwojażera posiada złożoność obliczeniową O(n^2*2^n).
W przypadku problemu komiwojażera podproblemem jest przejście od wierzchołka 0 do jednego z pozostałych przez sekwencję określonych wierzchołków. Przykładowo dla grafu o pięciu wierzchołkach podproblemem może być przejście od wierzchołka 0 do wierzchołka 3 przez wierzchołki 2, 3 i 4. Rozwiązywanie całego problemu rozpoczyna się od obliczenia optymalnego rozwiązania dla najmniejszych podproblemów tj. optymalna ścieżka od wierzchołka 0 przez zbiory jednoelementowe, czyli przykładowo przejście od wierzchołka 0 do wierzchołka 1 przechodząc jedynie przez wierzchołek 1. Następnie wyliczyć trzeba optymalne rozwiązania dojścia do każdego wierzchołka przez zbiór dwóch wierzchołków (nie licząc wierzchołka zerowego). W tym celu wykorzystujemy rozwiązania z poprzedniego kroku.

Metoda podziału i ograniczeń

Metoda podziału i ograniczeń polega na analizie drzewa przestrzeni stanów. Drzewo to reprezentuje wszystkie możliwe ścieżki jakimi może pójść algorytm. Rozpoczynając od korzenia, w każdym węźle wykonywane jest obliczenie rozwiązywania częściowego, aż do momentu dojścia do liścia. Przechodzenie przez każdy węzeł okazuje się być kosztowne, dlatego też dla każdego węzła wyliczane jest dolne ograniczenie, które pozwala określić czy dany wierzchołek jest obiecujący i czy powinien być dalej rozwijany. Dolne ograniczenie, to najlepsza możliwa wartość jaka może być osiągnięta z danego wierzchołka. Ważnym elementem metody podziału i ograniczeń jest właśnie funkcja wyliczająca ograniczenie. Funkcja ograniczająca nie jest wykorzystywana bezpośrednio do sprawdzania, czy wierzchołek jest obiecujący. Jednak na podstawie dolnego ograniczenia wyliczany jest koszt wierzchołka drzewa (który to już jest wykorzystywany do sprawdzania, czy wierzchołek jest obiecujący). Funkcja obliczająca dolne ograniczenie polega na zredukowaniu macierzy redukcji danego węzła, by w każdej kolumnie i wierszu znajdowało się przynajmniej jedno zero.

Przykład praktyczny

Dla zaprezentowania działania algorytmów rozwiązujących problem komiwojażera wykorzystamy graf reprezentowany przez daną macierz:

$$G = \begin{bmatrix} 0 & 4 & 1 & 9 \\ 3 & 0 & 6 & 11 \\ 4 & 1 & 0 & 2 \\ 6 & 5 & 4 & 0 \end{bmatrix}$$

Gdzie G[m][n] jest wagą krawędzi idącej od wierzchołka m do n.
Numeracja wierzchołków zaczyna się od 0 i od wierzchołka zerowego zawsze zaczynamy cykl.

Przegląd zupełny

Do zapamiętania najlepszego kosztu cyklu oraz samego cyklu posłużą nam zmienne best_cost i best_path. Zmienna best_cost na początku równa się nieskończoności, a best_path nie zawiera w sobie żadnej ścieżki. Zmienne path i cost reprezentują aktualnie badaną permutację

Krok 1:
best_cost = INF, best_path = null
path = 0-2-3-1-0 cost = 11 => zmiana najlepszej ścieżki i kosztu
Krok 2:
best_cost = 11, best_path = 0-2-3-1-0
path = 0-3-2-1-0 cost = 17 => nie zmieniamy najlepszej ścieżki i kosztu
…
Krok 6:
best_cost = 11, best_path = 0-2-3-1-0
path = 0-1-2-3-0 cost = 18 => nie zmieniamy najlepszej ścieżki i kosztu
Ostatecznie algorytm przeglądu zupełnego zwraca nam ścieżkę 0-2-3-1-0 o koszcie 11.

Programowanie dynamiczne

Niech D(V, l) będzie oznaczało optymalną długość ścieżki wychodzącą z wierzchołka 0 do wierzchołka l przechodzącego przez wierzchołki V. Przykładowo D({1, 2}, 2) będzie oznaczało przejście od wierzchołka 0 do wierzchołka 2 przez wierzchołki 1, 2.
Dodatkowo p(V, l) będzie oznaczało wierzchołek ze zbioru V poprzedzający wierzchołek l.

Krok 1:
Wyznaczenie najlepszej ścieżki dla zbiorów jednoelementowych:
D({1}, 1) = K[0][1] = 4, p({1}, 1) = 0
D({2}, 2) = K[0][2] = 1, p({2}, 2) = 0
D({3}, 3) = K[0][3] = 9, p({2}, 2) = 0

Krok 2:
Wyznaczenie najlepszej ścieżki dla zbiorów dwuelementowych:
D({1, 2}, 1) = min(D({2}, 2) + K[2][1]) = min(1 + 1) = min(2) = 2, p({1, 2}, 1) = 2
D({1, 2}, 2) = min(D({1}, 1) + K[1][2]) = min(4 + 6) = min(2) = 6, p({1, 2}, 2) = 1

D({2, 3}, 2) = min(D({3}, 3) + K[3][2]) = min(9 + 4) = min(2) = 13, p({2, 3}, 2) = 3
D({2, 3}, 3) = min(D({2}, 2) + K[2][3]) = min(1 + 2) = min(2) = 3, p({2, 3}, 3) = 2
D({1, 3}, 1) = min(D({3}, 3) + K[3][1]) = min(9 + 5) = min(2) = 16, p({1, 3}, 1) = 3
D({1, 3}, 3) = min(D({1}, 1) + K[1][3]) = min(4 + 11) = min(2) = 15, p({1, 3}, 3) = 1

Krok 3:
Wyznaczenie najlepszej ścieżki dla zbiorów trzyelementowych:

D({1, 2, 3}, 1) = min(D({2, 3}, 2) + K[2][1], D({2, 3}, 3) + K[3][1]) = min(13 + 1, 3 + 5) = min(14, 8) = 8, p({1, 2, 3}, 1) = 3

D({1, 2, 3}, 2) = min(D({1, 3}, 1) + K[1][2], D({1, 3}, 3) + K[3][2]) = min(16 + 6, 15 + 4) = min(22, 19) = 19, p({1, 2, 3}, 2) = 3

D({1, 2, 3}, 3) = min(D({1, 2}, 1) + K[1][3], D({1, 2}, 2) + K[2][3]) = min(2 + 11, 6 + 2) = min(13, 8) = 8, p({1, 2, 3}, 3) = 2

Krok 4:
Wyznaczenie optymalnego kosztu cyklu Hamiltona:

min(D({1, 2, 3}, 1) + K[1][0], D({1, 2, 3}, 2) + K[2][0], D({1, 2, 3}, 2) + K[2][0]) = min(8 +3, 19 + 4, 8 + 6) = min(11, 23, 14) = 11

Krok 5:
Wyznaczenie cyklu o optymalnym koszcie:
Dla optymalnej ścieżki wierzchołkiem poprzedzającym wierzchołek 0 był wierzchołek o indeksie 1, zatem:
p({1, ,2, 3}, 1) = 3,
p({2, 3,}, 3) = 2
p({2}, 2) = 0
Ostatecznie algorytm programowania dynamicznego zwraca nam ścieżkę 0-2-3-1-0 o koszcie 11.

Metoda podziału i ograniczeń

Niech Ni(c, P) będzie i-tym węzłem drzewa stanu o koszcie c i zawierającym listę wierzchołków grafu P. Należy też ustawić wartość best_cost na nieskończoność.
c_Ni oznaczać będzie koszt węzła Ni.
Krok 1:
best_cost = INF
best_path = null
Przygotowanie macierzy grafu do redukcji:

$$R0 = \begin{bmatrix} INF & 4 & 1 & 9 \\ 3 & INF & 6 & 11 \\ 4 & 1 & INF & 2 \\ 6 & 5 & 4 & INF \end{bmatrix}$$

Krok 2:
Obliczenie kosztu dla korzenia drzewa stanu poprzez zredukowanie macierzy:

- Odjęcie kosztu ze wszystkich kolumn

$$R0 = \begin{bmatrix} INF & 3 & 0 & 7 \\ 0 & INF & 5 & 9 \\ 1 & 0 & INF & 0 \\ 3 & 4 & 3 & INF \end{bmatrix}$$

Suma odjętych wartości wynosi 3 + 1 + 1 + 2 = 7

- Odjęcie kosztu z wszystkich wierszy

$$R0 = \begin{bmatrix} INF & 3 & 0 & 7 \\ 0 & INF & 5 & 9 \\ 1 & 0 & INF & 0 \\ 0 & 1 & 0 & INF \end{bmatrix}$$

Suma odjętych wartości wynosi 0 + 0 + 0 + 3 = 3

Koszt węzła wynosi 7 + 3 = 10
Zatem otrzymujemy węzeł N0(10, 0) który wstawiamy do kolejki:
priority_queue: N0(10, 0)
best_path = null
best_cost = INF

Krok 3:
Pobieramy węzeł N0 z kolejki priorytetowej i obliczamy koszt dla każdego jego potomka.

- N1(?, 0-1)

  „Blokujemy" przejścia dla wierzchołka 0 oraz 1

$$R1 = \begin{bmatrix} INF & INF & INF & INF \\ INF & INF & 5 & 9 \\ 1 & INF & INF & 0 \\ 0 & 1 & 0 & INF \end{bmatrix}$$

  Redukujemy macierz

$$R1 = \begin{bmatrix} INF & INF & INF & INF \\ INF & INF & 0 & 4 \\ 1 & INF & INF & 0 \\ 0 & 0 & 0 & INF \end{bmatrix}$$

  Zredukowana wartość r = 1 + 5 = 6

  Koszt przejścia c_N0 + r + R0[0][1] = 10 + 6 + 3 = 19

  19 < best_cost zatem węzeł N1(19, 0-1) wstawiamy do kolejki priorytetowej

  priority_queue: N1(19, 0-1)
  best_path = null
  best_cost = INF

- N2(?, 0-2)

  „Blokujemy" przejścia dla wierzchołka 0 oraz 2

$$R2 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & 9 \\ INF & 0 & INF & 0 \\ 0 & 1 & INF & INF \end{bmatrix}$$

  Redukujemy macierz

$$R2 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & 9 \\ INF & 0 & INF & 0 \\ 0 & 1 & INF & INF \end{bmatrix}$$

  Zredukowana wartość r = 0

Koszt przejścia c_N0 + r + R0[0][2] = 10 + 0 + 0 = 10

10 < best_cost zatem węzeł N2(10, 0-2) wstawiamy do kolejki priorytetowej

priority_queue: N2(10, 0-2), N1(19, 0-1)
best_path = null
best_cost = INF

- N3(?, 0-3)

  „Blokujemy" przejścia dla wierzchołka 0 oraz 2

$$R3 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & 5 & INF \\ 1 & 0 & INF & INF \\ INF & 1 & 0 & INF \end{bmatrix}$$

  Redukujemy macierz

$$R3 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & 5 & INF \\ 1 & 0 & INF & INF \\ INF & 1 & 0 & INF \end{bmatrix}$$

  Zredukowana wartość r = 0

  Koszt przejścia c_N0 + r + R0[0][3] = 10 + 0 + 7 = 17

  17 < best_cost zatem węzeł N3(10, 0-3) wstawiamy do kolejki priorytetowej

  priority_queue: N2(10, 0-2), N3(17, 0-3), N1(19, 0-1)
  best_path = null
  best_cost = INF

Krok 4:

Pobieramy węzeł N2 z kolejki priorytetowej i obliczamy koszt dla każdego jego potomka.

- N4(?, 0-2-1)

  „Blokujemy" przejścia dla wierzchołka 2 oraz 1

$$R4 = \begin{bmatrix} INF & INF & INF & INF \\ INF & INF & INF & 9 \\ INF & INF & INF & INF \\ 0 & INF & INF & INF \end{bmatrix}$$

  Redukujemy macierz

$$R4 = \begin{bmatrix} INF & INF & INF & INF \\ INF & INF & INF & 0 \\ INF & INF & INF & INF \\ 0 & INF & INF & INF \end{bmatrix}$$

  Zredukowana wartość r = 0

  Koszt przejścia c_N2 + r + R2[2][1] = 10 + 9+ 0 = 19

  19 < best_cost zatem węzeł N4(19, 0-2-1) wstawiamy do kolejki priorytetowej

  priority_queue: N3(17, 0-3), N1(19, 0-1), N4(19, 0-2-1)

  best_path = null

  best_cost = INF

- N5(?, 0-2-3)

  „Blokujemy" przejścia dla wierzchołka 2 oraz 3

$$R5 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & INF \\ INF & INF & INF & INF \\ INF & 1 & INF & INF \end{bmatrix}$$

Redukujemy macierz

$$R5 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & INF \\ INF & INF & INF & INF \\ INF & 0 & INF & INF \end{bmatrix}$$

Zredukowana wartość r = 1

Koszt przejścia c_N2 + r + R2[2][3] = 10 + 1 + 0 = 11

11 < best_cost zatem węzeł N5(11, 0-2-3) wstawiamy do kolejki priorytetowej

priority_queue: N5(11, 0-2-3), N3(17, 0-3), N1(19, 0-1), N4(19, 0-2-1)

best_path = null

best_cost = INF

Krok 5:

Pobieramy węzeł N5 z kolejki priorytetowej i obliczamy koszt dla każdego jego potomka.

- N6(?, 0-2-3-1)

  „Blokujemy" przejścia dla wierzchołka 3 oraz 1

$$R6 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & INF \\ INF & INF & INF & INF \\ INF & INF & INF & INF \end{bmatrix}$$

  Redukujemy macierz

$$R6 = \begin{bmatrix} INF & INF & INF & INF \\ 0 & INF & INF & INF \\ INF & INF & INF & INF \\ INF & INF & INF & INF \end{bmatrix}$$

  Zredukowana wartość r = 0

  Koszt przejścia c_N5 + r + R5[3][1] = 11 + 0 + 0 = 11

  N6 jest liściem oraz 11 < best_cost, zatem N6 jest aktualnym najlepszym rozwiązaniem

  priority_queue: N3(17, 0-3), N1(19, 0-1), N4(19, 0-2-1)

  best_path = 0-2-3-1

  best_cost = 11

Krok 7:

Pozostałe węzły mają wyższy koszt niż aktualne najlepsze rozwiązania, dlatego też zostaną odrzucone.

Ostatecznie algorytm metodą podziału i ograniczeń zwraca nam ścieżkę 0-2-3-1-0 o koszcie 11.

Plan eksperymentu

Do pomiarów czasu wykorzystano bibliotekę <chrono>. Wagi krawędzi grafu generowane były w sposób pseudolosowy z wykorzystaniem generatora liczb pseudolosowych rand(). Wygenerowane liczby były z zakresu od 1 do 1000. W przypadku przeglądu zupełnego grafy posiadały następującą liczbę wierzchołków 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. Dla programowania dynamicznego oraz metody podziału ograniczeń grafy były rzędu 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22. Dla każdej wielkości dokonano 100 pomiarów czasu, za każdym razem generując nowy zestaw wartości pseudolosowych.

Tabela 1 Wyniki pomiaru czasu dla przeglądu zupełnego

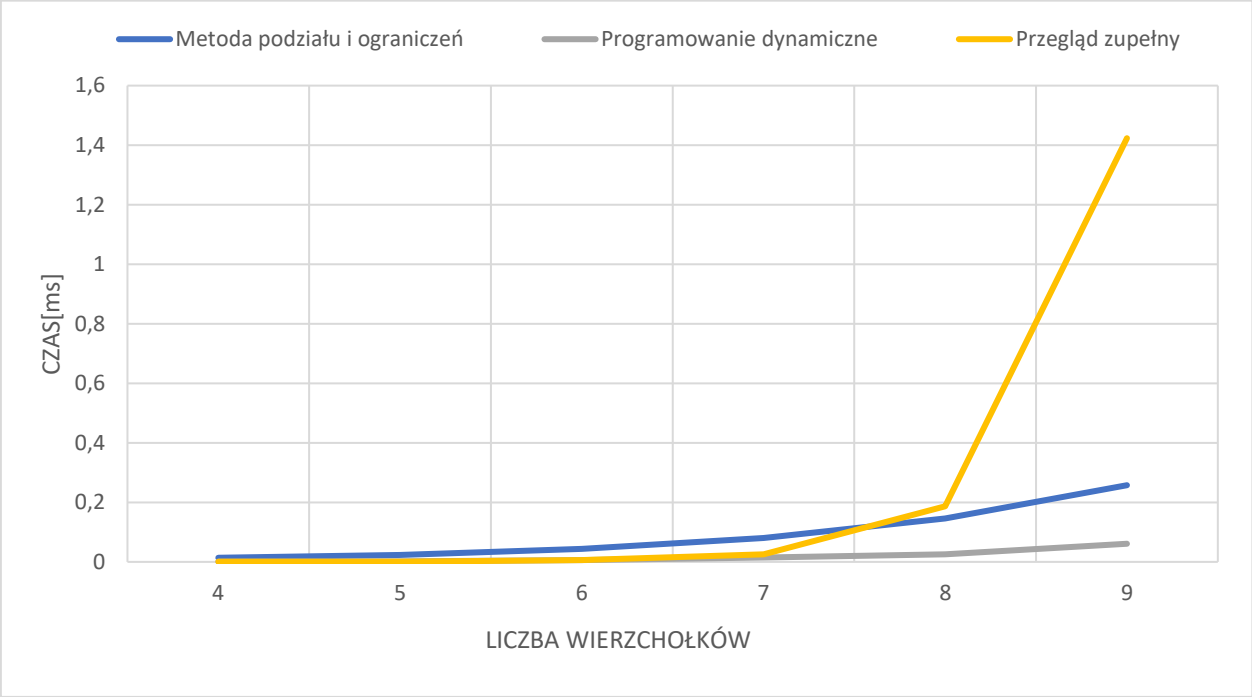
Przegląd zupełny										
L. w.	4	5	6	7	8	9	10	11	12	13
Czas[ms]	0.0007	0.0011	0.0058	0.0256	0.1866	1.4233	11.8613	119.7743	1329.4981	16244.3100

Tabela 2 Wyniki pomiaru czasu dla programowania dynamicznego

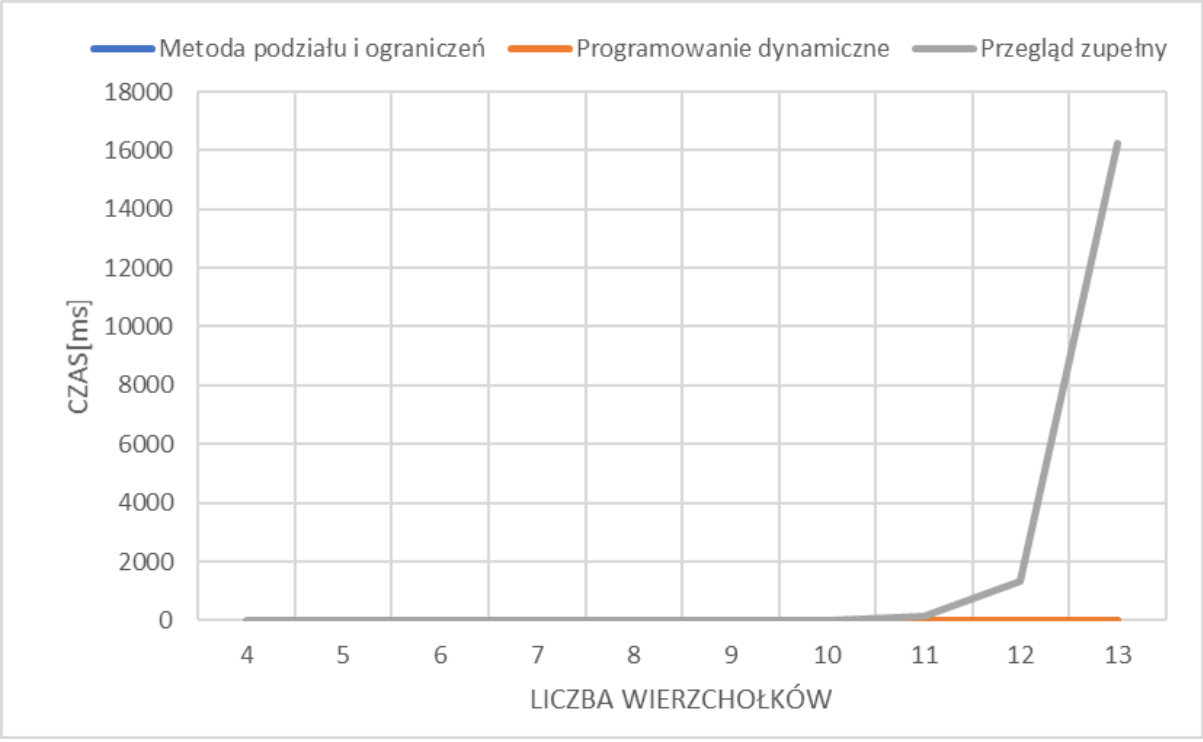
Programowanie dynamiczne														
L. w.	4	5	6	7	8	9	10	11	12	14	16	18	20	22
Czas[ms]	0.0009	0.0021	0.0058	0.0148	0.0255	0.0610	0.15	0.37	0.84	4.64	26.68	204.16	1038.23	5375.39

Tabela 3 Wyniki pomiaru czasu dla metody podziału i ograniczeń

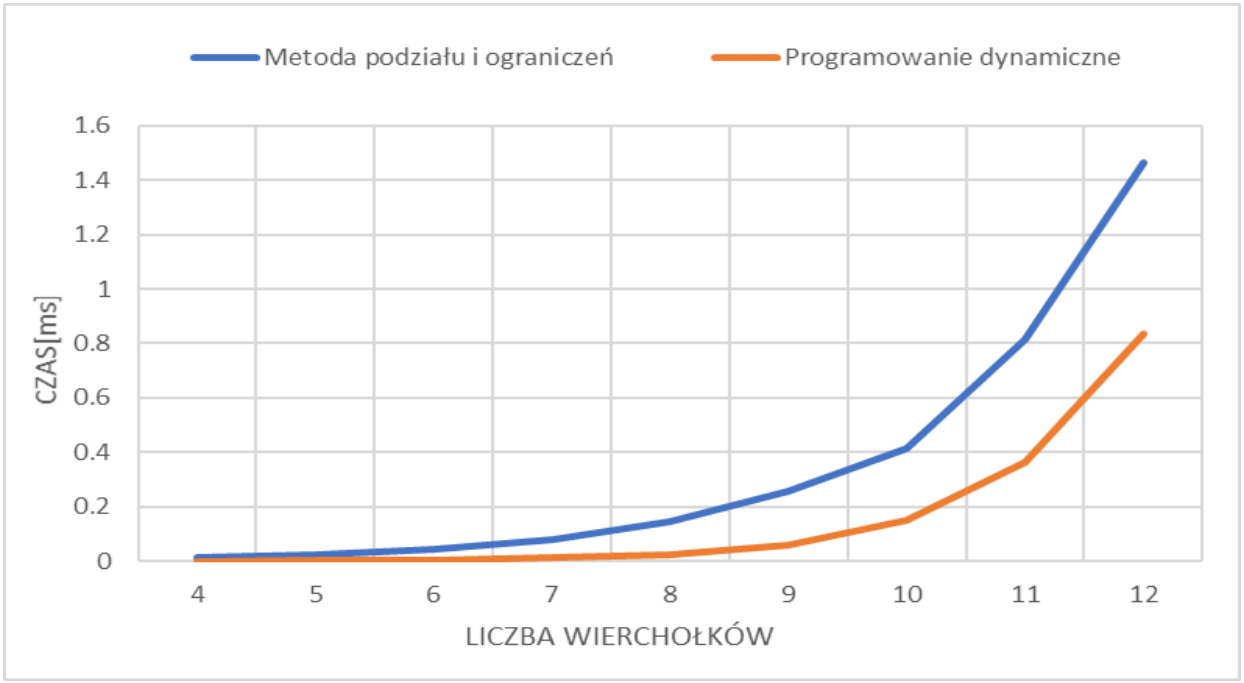
Metoda podziału i ograniczeń														
L. w.	4	5	6	7	8	9	10	11	12	14	16	18	20	22
Czas[ms]	0.014	0.023	0.044	0.081	0.146	0.258	0.414	0.82	1.46	4.03	8.70	30.90	60.40	213.99



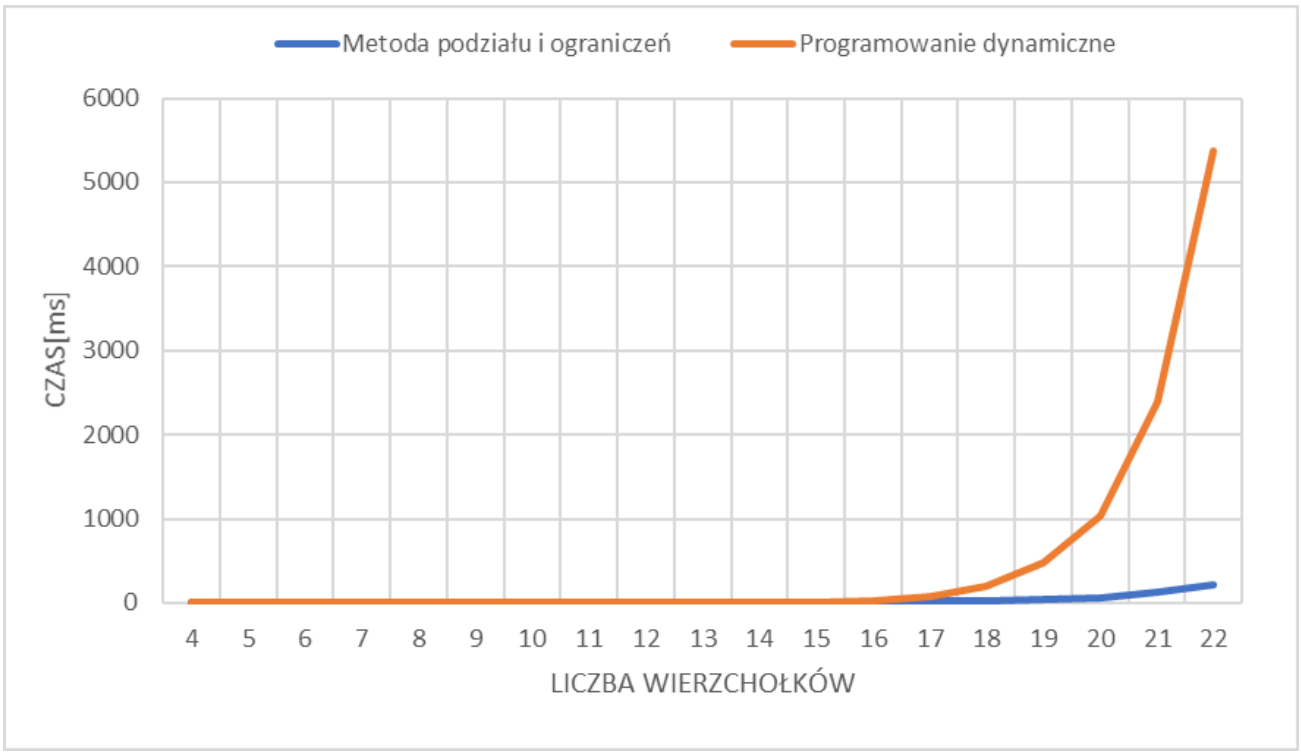
Wykres 1 Wykres pomiaru czasu w zależności od liczby wierzchołków dla zakresu wierzchołków od 4 do 9 dla wszystkich 3 algorytmów



Wykres 2 Wykres pomiaru czasu w zależności od liczby wierzchołków dla zakresu wierzchołków od 4 do 13 dla wszystkich 3 algorytmów



Wykres 4 Wykres pomiaru czasu w zależności od liczby wierzchołków dla zakresu wierzchołków od 4 do 12 dla wszystkich programowania dynamicznego i metody podziału i ograniczeń



Wykres 3 Wykres pomiaru czasu w zależności od liczby wierzchołków dla zakresu wierzchołków od 4 do 22 dla wszystkich programowania dynamicznego i metody podziału i ograniczeń

Wnioski

Różnice między działaniem przeglądu zupełnego, a dwóch pozostałych algorytmów są zauważalne. Obliczenia wykonywane dla metody podziału i ograniczeń oraz programowania dynamicznego są dużo szybsze dla większych grafów. Warto jednak zauważyć, iż metoda przeglądu zupełnego wykonuje się szybciej dla grafów o mniejszej liczbie wierzchołków (programowanie dynamiczne jest szybsze od 6 wierzchołków, natomiast metoda podziału i ograniczeń od 8). W przypadku porównania metody podziału i ograniczeń oraz programowania dynamicznego, można zauważyć, że z początku to programowanie dynamiczne jest szybsze. Sytuacja zmienia się jednak dla grafów stopnia 14 i wyższych. Złożoności obliczeniowe podane w literaturze pokrywają się z tymi wynikającymi z eksperymentu.