# Zadanie projektowe 1

Sprawozdanie

Kurs: Projektowanie efektywnych algorytmów

Prowadzący: mgr inż. Antoni Sterna

Grupa: E01-60h (środa 13:15-15:00)

Autor: Mirosław Kuźniar (nr indeksu: 248870)

# 1 Wstęp teoretyczny

## 1.1 Problem optymalizacyjny

Rozważanym problemem optymalizacyjnym jest problem komiwojażera (ang. Travelling Salesman Problem). Zgodnie z definicją polega on na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. W praktyce często opisuje się go jako problem komiwojażera/wędrownego sprzedawcy który wyruszając z miasta początkowego musi odwiedzić każde z pozostałych miast tylko raz i wrócić do miasta początkowego. Ponadto trasa, którą przebędzie musi być minimalna pod względem przyjętego kosztu (np. odległości, czasu, kosztów ekonomicznych itp.). Warto także wspomnieć, że zgodnie z definicją grafu pełnego musi istnieć połączenie z danego miasta do wszystkich pozostałych.

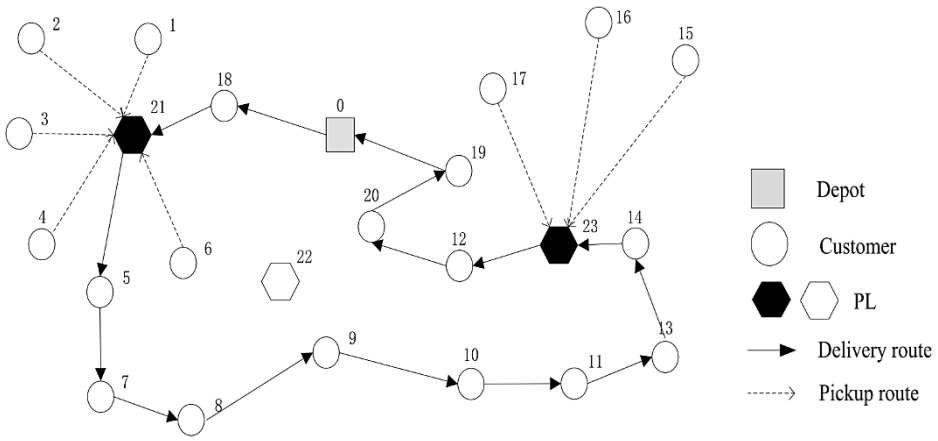
Problem komiwojażera należy do klasy problemów NP – trudnych. Problemy z tej klasy opisywane są jako takie dla których (najprawdopodobniej) nie można skonstruować algorytmów wielomianowych, czyli algorytmów efektywnych obliczeniowo.

Wyróżniamy dwa rodzaje problemu komiwojażera:

- Symetryczny problem komiwojażera (STSP), w którym waga krawędzi przyjmuje jedną wartość bez względu na kierunek poruszania się, tzn. odległość z miasta A do miasta B jest taka sama jak z miasta B do miasta A,
- Asymetryczny problem komiwojażera (ATSP), w którym waga krawędzi może przyjąć różne wartości w zależności od kierunku przemieszczania się, tzn. odległość z miasta A do miasta B może być różna niż z miasta B do miasta A.

Rozważanym dalej rodzajem będzie asymetryczny problem komiwojażera (ATSP).

Pomimo swojej złożoności problem komiwojażera nie jest tylko tematem dysput akademickich ale także znajduje zastosowanie w praktycznych aplikacjach. Dla przykładu, wydajne rozwiązania TSP są często wykorzystywane w tzw. dostawach ostatniej mili (ang. last mile delivery), gdzie towar przemieszczany jest z węzła transportowego, takiego jak skład lub magazyn do klienta. Dostawy te stanowią główne źródło różnic kosztów w całym łańcuchu dostaw. Dlatego wiele firm z sektora logistyki dąży do zminimalizowania kosztów dostawy ostatniej mili wykorzystując wydajne algorytmy bazujące na problemie komiwojażera.



## 1.2 Algorytmy dokładne

### 1.2.1 Przegląd zupełny

W przypadku problemu komiwojażera przegląd zupełny polega na wygenerowaniu wszystkich możliwych ścieżek Hamiltona poprzez systematyczne permutowanie wierzchołków reprezentujących miasta, a następnie sprawdzeniu długości każdej ścieżki i wybranie jako rozwiązania tej o najmniejszym możliwym koszcie.

Biorąc pod uwagę założenia projektowe dopuszczalne są tylko te permutacje, które zaczynają się od określonego miasta początkowego. Zmniejsza to ilość możliwych rozwiązań z $n!$ do $(n-1)!$

Zaletą przeglądu zupełnego jest jego intuicyjność oraz gwarancja znalezienia rozwiązania optymalnego. Z drugiej strony redukcja przestrzeni rozwiązań jest niewielka, co przekłada się na długi czas wykonywania – złożoność czasowa $O(n!)$ i efektywną używalność tylko przy niewielkich rozmiarach problemu.

### 1.2.2 Programowanie dynamiczne

Programowanie dynamiczne to koncepcja projektowania algorytmów polegająca na podziale złożonego problemu na zależne od siebie podproblemy o mniejszej złożoności. Rozwiązania mniejszych podproblemów są wykorzystywane do rozwiązań większych podproblemów. Każdy podproblem rozważany jest jedynie raz.

Algorytmem opartym na koncepcji programowania dynamicznego, używanym do rozwiązywania problemu komiwojażera jest algorytm Helda-Karpa (czasami określany jako algorytm Bellmana-Helda-Karpa).

Działanie algorytmu można opisać równaniem:

$$g(i,S) = \begin{cases} min_{k \in S}\{c_{i \to k} + g(k, S - \{k\})\}, \text{ gdy } S \notin \{\emptyset\} \\ c_{i \to p}, \text{ gdy } S \in \{\emptyset\} \end{cases}$$

Optymalna długość ścieżki z aktualnego miasta $i$ jest oznaczona jako $g(i,S)$. Zbiór $S$ zawiera w sobie wszystkie nieodwiedzone do tej pory miasta. Dopóki zbiór $S$ nie jest zbiorem pustym wartość funkcji obliczana jest jako minimum z kosztu $c$ przejścia z aktualnego miasta $i$ do nieodwiedzonego miasta $k$ powiększona o wartość funkcji dla miasta $k$. W momence gdy zbiór $S$ jest zbiorem pustym (wszystkie miasta zostały odwiedzone) wartość funkcji wynosi tyle co koszt przejścia z aktualnego miasta $i$ do miasta początkowego $p$.

Algorytm Helda-Karpa charakteryzuje się złożonością czasową $O(n^2 2^n)$.

### 1.2.3 Metoda podziału i ograniczeń

Działanie metody podziału i ograniczeń polega na obliczaniu w każdym węźle drzewa przestrzeni stanów ograniczenia, które pozwala określić go jako obiecujący bądź nie. W dalszej fazie algorytm przegląda tylko potomków węzłów obiecujących. Pozwala to, razem z dobraniem odpowiedniej strategii liczenia granicy, zmniejszyć ilość odwiedzonych wierzchołków i szybciej znaleźć rozwiązanie problemu.

Jako, że metoda B&B nie została zaimplementowana w projekcie nie będzie dalej omawiana.

# 2 Przykład praktyczny

Poniżej przedstawiony zostanie opis działania zaimplementowanych algorytmów dla przykładowego problemu o rozmiarze $N = 3$, opisanego poniższą macierzą sąsiedztwa.

$$\begin{bmatrix} 0 & 23 & 4 \\ 1 & 0 & 9 \\ 12 & 4 & 0 \end{bmatrix}$$

## 2.1 Przegląd zupełny

**Krok 1**: Wygenerowanie permutacji bazowej

W omawianym przykładzie permutacja bazowa przyjmie postać: `0,1,2`

**Krok 2**: Obliczenie kosztu permutacji i wpisanie go do tablicy kosztów

| i | przejście | koszt += macierz_sasiedztwa[…] |
|---|---|---|
| 0 | 0 -> 1 | 0 + 23 = 23 |
| 1 | 1 -> 2 | 23 + 9 = 32 |
| | 2 -> 0 | 32 + 12 = **44** |

| indeks: | 0 |
|---|---|
| tablica_kosztow: | 44 |

**Krok 3:** Wprowadzenie permutacji do tablicy permutacji

| indeks: | 0 | 1 | 2 |
|---|---|---|---|
| tablica_permutacji: | 0 | 1 | 2 |

**Krok 4:** Wygenerowanie pozostałych permutacji i wykonanie dla nich Kroków 2 i 3

Do generowania permutacji została używa funkcja `std::next_permutation()`

Po wykonaniu Kroku 4 tablica kosztów i permutacji prezentuje się następująco:

| indeks: | 0 | 1 |
|---|---|---|
| tablica_kosztow: | 44 | 9 |

| indeks: | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| tablica_permutacji: | 0 | 1 | 2 | 0 | 2 | 1 |

**Krok 5:** Wyznaczenie minimum z tablicy kosztów

```
koszt_calkowity = min(tablica_kosztow) = 9

indeks_min = 1
```

**Krok 6:** Wyznaczenie ścieżki o minimalnym koszcie z tablicy permutacji

| Od: | tablica_permutacji[N*indeks_min+0] |
|---|---|
| Do: | tablica_permutacji[N*indeks_min+(N-1)] |

Ścieżka: **0->2->1(->0)**

## 2.2 Programowanie dynamiczne

**Krok 1:** Wyznaczenie wartości funkcji $g(i,S)$ dla zbiorów pustych:

$$g(1,\{\emptyset\}) = c_{1\to 0} = 1$$

$$g(2,\{\emptyset\}) = c_{2\to 0} = 12$$

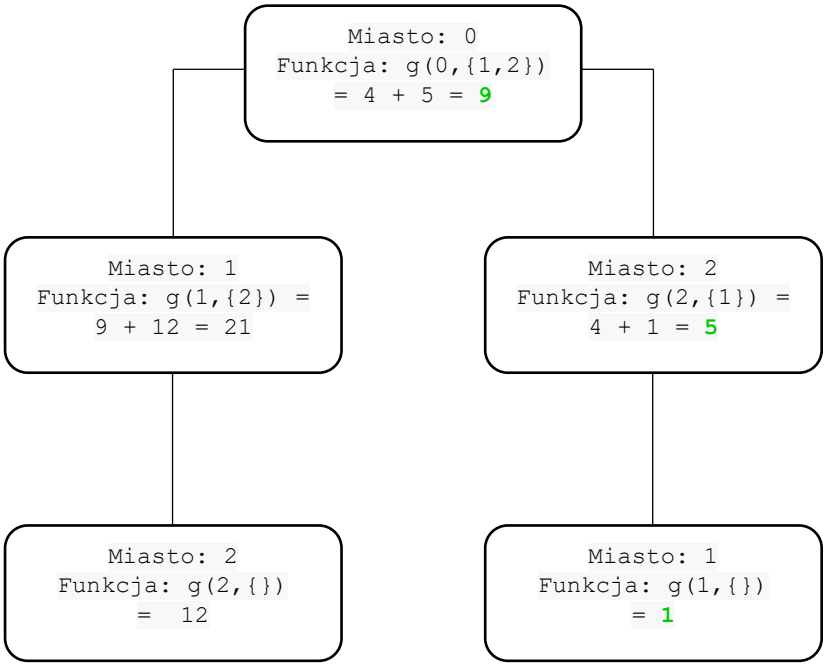
**Krok 2:** Wyznaczenie wartości funkcji $g(i,S)$ dla zbiorów jednoelementowych:

$$g(2,\{1\}) = \min\left(c_{2\to 1} + g(1,\{\emptyset\}) = \min(4+1) = 5\right.$$

$$g(1,\{2\}) = \min\left(c_{1\to 2} + g(2,\{\emptyset\}) = \min(9+12) = 21\right.$$

**Krok 3:** Wyznaczenie wartości funkcji $g(i,S)$ dla zbiorów dwuelementowych:

$$g(0,\{1,2\}) = \min\left(c_{0\to 1} + g(1,\{2\}), c_{0\to 2} + g(2,\{1\}) = \min(23+21, 4+5) = 9\right.$$




`koszt_calkowity = g(0,{1,2}) =` **9**

Ścieżka: **0->2->1->0**

# 3 Implementacja algorytmów

Wybranym językiem programowania jest C++. Algorytmy operują na zmiennych dziesiętnych `int`. Do przechowywania danych wykorzystano strukturę danych z biblioteki STL - `vector`.

# 4 Plan eksperymentu

## 4.1 Rozmiar problemu

W przypadku algorytmu przeglądu zupełnego wykonano pomiary dla 10 różnych rozmiarów problemu: 3,4,5,6,7,8,9,10,11,12. Dla problemów o większych rozmiarach czas wykonania algorytmu znacząco utrudniał przeprowadzenie pomiarów.

W przypadku algorytmu Helda-Karpa wykonano pomiary dla 13 rożnych rozmiarów problemu: 3,4,5,6,7,8,9,10,11,12,13,14,15. Dla problemów o większych rozmiarach czas wykonania algorytmu znacząco utrudniał przeprowadzenie pomiarów.

## 4.2 Sposób generowania danych

Do generowania danych posłużył zewnętrzny program. Działanie programu polega na zapisywaniu macierzy o zadanych przez użytkownika rozmiarach do pliku tekstowego. Macierz na swojej przekątnej wypełniona jest zerami, a w pozostałych miejscach losowymi liczbami całkowitymi.

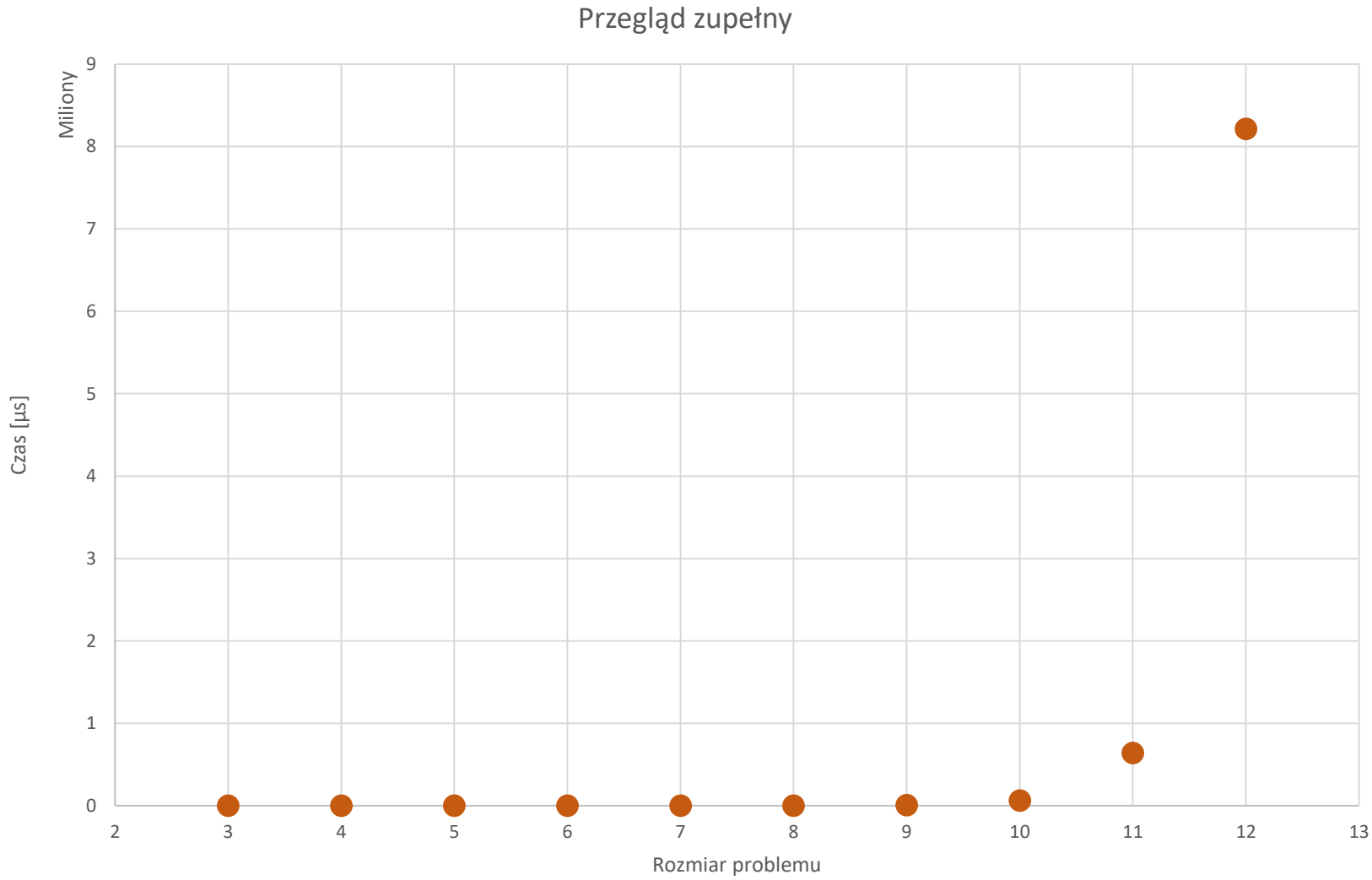
## 4.3 Pomiar czasu

Do pomiaru czasu użyta została rozbudowana wersja funkcji `read_QPC()`, zawierająca funkcję `QueryPerformanceFrequency()`. Zastosowanie jej umożliwia pomiar czasu w rozdzielczości mikrosekundowej.

# 5 Wyniki pomiarów

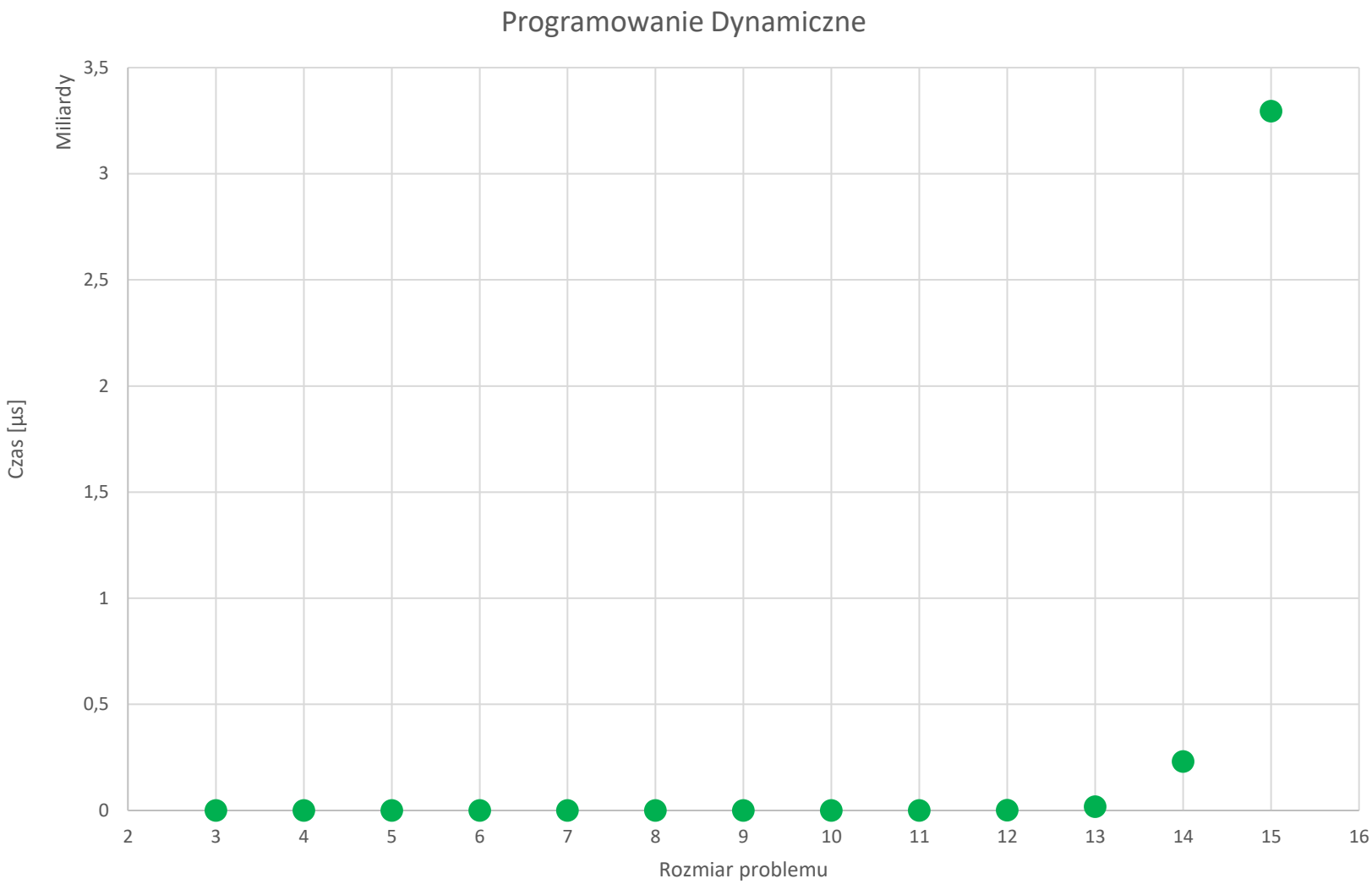
## 5.1 Przegląd zupełny

| N | Czas [µs] |
|---|---|
| 3 | 11 |
| 4 | 12 |
| 5 | 18 |
| 6 | 33 |
| 7 | 102 |
| 8 | 699 |
| 9 | 6129 |
| 10 | 63044 |
| 11 | 639596 |
| 12 | 8215196 |



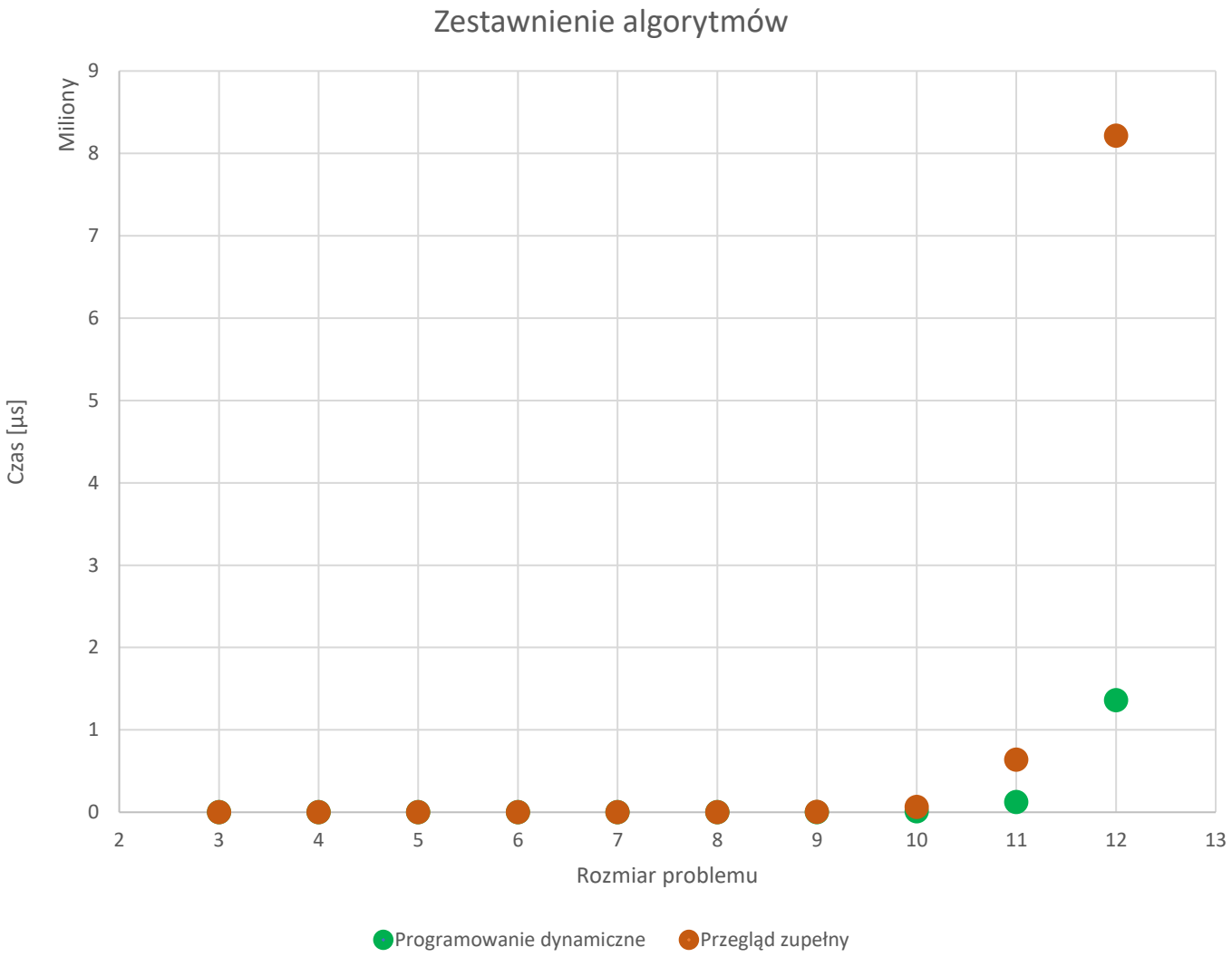
## 5.2 Programowanie dynamiczne

| N | Czas [µs] |
|---|---|
| 3 | 1 |
| 4 | 2 |
| 5 | 3 |
| 6 | 6 |
| 7 | 23 |
| 8 | 143 |
| 9 | 1223 |
| 10 | 12052 |
| 11 | 123522 |
| 12 | 1361424 |
| 13 | 16968813 |
| 14 | 230537769 |
| 15 | 3294330374 |



## 5.3 Porównanie algorytmów

| N | Czas [µs] | |
|---|---|---|
| | **Programowanie dynamiczne** | **Przegląd zupełny** |
| 3 | 1 | 11 |
| 4 | 2 | 12 |
| 5 | 3 | 18 |
| 6 | 6 | 33 |
| 7 | 23 | 102 |
| 8 | 143 | 699 |
| 9 | 1223 | 6129 |
| 10 | 12052 | 63044 |
| 11 | 123522 | 639596 |
| 12 | 1361424 | 8215196 |



# 6 Wnioski

Zadaniem obydwu omówionych algorytmów jest znalezienie ścieżki o minimalnym koszcie w problemie komiwojażera. Zarówno przegląd zupełny jak i programowanie dynamiczne należą do rodziny algorytmów dokładnych tzn. takich, których wynikiem jest rozwiązanie optymalne. Jednak jak można zauważyć na powyższych wykresach programowanie dynamiczne cechuje się krótszym czasem wykonania, co szczególnie staje się widoczne przy większych rozmiarach problemu. Algorytm przeglądu zupełnego, będący najbardziej intuicyjnym, wykazuje dłuższe czasy wykonania nawet przy małych rozmiarach problemu ale wraz ze wzrostem rozmiaru problemu czasy ulegają jeszcze bardziej zauważalnemu wydłużeniu w porównaniu z programowaniem dynamicznym. W praktyce czyni go to algorytmem całkowicie niewydajnym. Biorąc pod uwagę tylko omówione algorytmy, gdy priorytetem jest rozwiązanie optymalne zaleca się użycie programowania dynamicznego.