Politechnika Wrocławska

Sprawozdanie z Projektu 3

Przedmiotu Projektowanie Efektywnych Algorytmów

Temat: Rozwiązanie asymetrycznego problemu komiwojażera z wykorzystaniem Algorytmu Genetycznego

GitHub Project Link: https://github.com/ksemk/ATSP

Kyrylo Semenchenko 273004
Informatyka Techniczna I stopnia
Wydział Informatyki i Telekomunikacji

18 stycznia 2025

Spis treści

1 Wstęp Teoretyczny

1.1 Opis Rozpatrywanego Problemu

Asymetryczny problem komiwojażera (ang. Asymmetric Traveling Salesman Problem, ATSP) jest klasycznym problemem optymalizacyjnym w teorii grafów i optymalizacji kombinatorycznej. W problemie tym mamy do czynienia z grafem skierowanym, w którym miasta są reprezentowane jako wierzchołki, a koszt przejścia między miastami jest opisany za pomocą wag przypisanych do krawędzi. Kluczową różnicą w stosunku do klasycznego problemu komiwojażera (TSP) jest asymetria kosztów: koszt podróży z miasta A do miasta B może różnić się od kosztu powrotu z miasta B do miasta A.

Celem ATSP jest znalezienie najkrótszej możliwej ścieżki, która przechodzi przez każde miasto dokładnie jeden raz i wraca do punktu początkowego, minimalizując całkowity koszt podróży. Problem ATSP znajduje zastosowanie w logistyce, planowaniu tras, a także w zarządzaniu transportem i dystrybucją zasobów, gdzie optymalizacja kosztów jest kluczowa. Ponieważ problem ATSP jest NP-trudny, jego rozwiązanie wymaga stosowania zaawansowanych technik optymalizacyjnych.

Definicja problemu: Mając dany zestaw n miast i macierz odległości D, gdzie $D[i][j]$ reprezentuje odległość z miasta i do miasta j, celem ATSP jest znalezienie permutacji π miast, która minimalizuje całkowity koszt podróży:

$$\text{Minimize} \sum_{i=1}^{n} D[\pi(i)][\pi(i+1)] + D[\pi(n)][\pi(1)]$$

gdzie $\pi(n+1) = \pi(1)$, aby zapewnić zamknięcie trasy.

Zastosowania: ATSP ma liczne praktyczne zastosowania, w tym:

- *Logistyka i transport:* Optymalizacja tras dostaw, gdzie koszty podróży nie są symetryczne.
- *Produkcja:* Sekwencjonowanie operacji na maszynach, gdzie czasy ustawiania między zadaniami są asymetryczne.
- *Telekomunikacja:* Trasowanie pakietów danych przez sieci z asymetrycznymi kosztami transmisji.

Złożoność: ATSP jest problemem NP-trudnym, co oznacza, że nie jest znany żaden algorytm działający w czasie wielomianowym, który rozwiązuje wszystkie przypadki problemu optymalnie. Ta złożoność wynika z konieczności oceny wszystkich możliwych permutacji miast w celu znalezienia optymalnej trasy, co rośnie wykładniczo wraz z liczbą miast.

1.2 Opis Algorytmu

Algorytm genetyczny jest inspirowany procesem ewolucji, opisanym przez Charlesa Darwina. W ramach tego algorytmu stosuje się specjalne oznaczenia dla rozwiązywanego problemu:

- Jeden zestaw miast, tworzący cykl Hamiltona, nazywany jest **chromosomem**.
- Pojedyncze miasto wchodzące w skład chromosomu określa się mianem **genu**.

Na początku generowany jest podstawowy zestaw chromosomów o stałym, określonym rozmiarze, zwany **populacją**. Następnie realizowany jest proces generacji potomków, który obejmuje następujące mechanizmy:

1. **Mutacja**: kolejność genów w wybranym segmencie chromosomu zostaje zmieniona. W mojej wersji algorytmu wykorzystuję dwie metody mutacji:
    - **Shuffle**: Kolejność genów w wybranym segmencie chromosomu zostaje losowo zmieniona.
    - **Inversion**: Kolejność genów w wybranym segmencie chromosomu zostaje odwrócona.
2. **Krzyżowanie**: Dwa chromosomy rodzicielskie wymieniają część swoich genów, tworząc dwójkę potomków.
3. **Zastąpienie**: Niektóre chromosomy są zastępowane przez losowo wygenerowane chromosomy.

Taka dywersyfikacja metod generacji potomstwa pozwala na zwiększenie różnorodności nowej generacji osobników. Następnie, na podstawie procesu selekcji, wybierana jest pula najlepszych osobników spośród nowej oraz starej generacji.

Aby zapobiec utracie różnorodności populacji oraz zmniejszyć prawdopodobieństwo utknięcia w lokalnym minimum, pozostałe miejsca w populacji uzupełniane są losowo wybranymi osobnikami. Dzięki temu populacja zachowuje odpowiedni poziom zmienności i zwiększa szanse na znalezienie optymalnego rozwiązania.

1.3 Opis fragmentów kodu

W poniższej sekcji przedstawiono kluczowe komponenty zaimplementowanego algorytmu genetycznego. Każdy z fragmentów kodu pełni istotną rolę w procesie optymalizacji problemu ATSP (Asymmetric Traveling Salesman Problem).

1.3.1 Inicjalizacja algorytmu

Pierwszym krokiem algorytmu jest wczytanie parametrów konfiguracyjnych z pliku JSON. Parametry te obejmują m.in.:

- `populationSize` - rozmiar populacji,
- `mutationRate` - prawdopodobieństwo mutacji,
- `crossoverRate` - prawdopodobieństwo krzyżowania,
- `iterationNum` - liczba iteracji algorytmu.

Następnie generowana jest populacja początkowa za pomocą funkcji `initializePopulation`, która tworzy losowe permutacje wierzchołków problemu.

1.3.2 Ocena rozwiązań

Funkcja celu, zaimplementowana jako `calculatePathCost`, oblicza koszt danego chromosomu poprzez sumowanie wartości z macierzy odległości. Wartość ta jest używana jako kryterium optymalizacyjne. Dodatkowo funkcja `calculateDiversity` mierzy różnorodność populacji, obliczając stopień podobieństwa między chromosomami.

1.3.3 Krzyżowanie

Operator krzyżowania, zaimplementowany w funkcji `performCrossing`, wykorzystuje metodę `Order Crossover (OX)`. Proces ten polega na wymianie segmentów genów między dwoma chromosomami-rodzicami w celu wygenerowania dwóch chromosomów potomnych, które zachowują poprawność permutacji.

1.3.4 Mutacja

Mutacja realizowana jest w funkcji `performMutation`, która wprowadza losowe zmiany w chromosomie. Wykorzystywany jest operator zamiany (`swap mutation`), który losowo wybiera dwa geny w chromosomie i zamienia ich pozycje. Mechanizm ten pozwala uniknąć przedwczesnej zbieżności do lokalnych minimów.

1.3.5 Selekcja

Selekcja przeprowadzana jest za pomocą funkcji `selection`, która wybiera najlepsze rozwiązania na podstawie wartości funkcji celu. Algorytm łączy populację rodzicielską i potomną, a następnie wybiera `populationSize` najlepszych osobników, zapewniając propagację najlepszych rozwiązań.

1.3.6 Główna pętla algorytmu

Cały proces optymalizacji zarządzany jest przez funkcję `runGeneticAlgorithm`, która iteracyjnie wykonuje następujące kroki:

1. Ocena populacji (`calculatePathCost`).
2. Selekcja rodziców (`selection`).
3. Krzyżowanie wybranych rodziców (`performCrossing`).
4. Mutacja potomków (`performMutation`).
5. Aktualizacja populacji na podstawie najlepszych rozwiązań.

Pętla powtarzana jest przez `iterationNum` iteracji lub do spełnienia kryterium stopu (np. brak poprawy w kolejnych iteracjach).

1.3.7 Wyniki końcowe

Po zakończeniu iteracji algorytm zwraca najlepsze znalezione rozwiązanie (chromosom) oraz jego koszt. Wyniki mogą być również zapisane do pliku w celu późniejszej analizy lub wizualizacji.

1.4 Oszacowanie złożoności obliczeniowej oraz czasowej

Złożoność obliczeniowa algorytmu genetycznego zależy od kilku kluczowych parametrów, takich jak rozmiar populacji (P), liczba generacji (G) oraz złożoność operacji wykonywanych na chromosomach, takich jak mutacja, krzyżowanie i selekcja.

- **Mutacja**:
  - W przypadku mutacji *shuffle* złożoność wynosi $O(L)$, gdzie L to długość chromosomu, ponieważ każdy gen w wybranym segmencie jest losowo przestawiany.
  - W przypadku mutacji *inversion* złożoność również wynosi $O(L)$, ponieważ wymaga odwrócenia kolejności genów w segmencie chromosomu.
- **Krzyżowanie**: Operacja krzyżowania dwóch chromosomów ma złożoność $O(L)$, gdyż wymaga iteracji po genach w celu wymiany segmentów między rodzicami.
- **Selekcja**: Wybór najlepszych osobników z populacji wymaga sortowania, co daje złożoność $O(P \log P)$, gdzie P to rozmiar populacji.

Całkowita złożoność obliczeniowa algorytmu genetycznego dla G generacji można oszacować jako:

$$O(G \cdot P \cdot (O(L) + O(P \log P))) = O(G \cdot P \cdot (L + P \log P))$$

przy założeniu, że mutacje i krzyżowanie są wykonywane na każdym chromosomie w każdej generacji.

1.4.1 Złożoność czasowa

W praktyce czas wykonania algorytmu genetycznego zależy od:

- liczby generacji (G),
- liczby operacji na populacji w każdej generacji,
- czasu wykonania pojedynczej operacji na chromosomie, który zależy od długości chromosomu (L).

Czas wykonywania może być również zależny od implementacji oraz sprzętu, np. równoległe wykonywanie operacji na chromosomach może znacząco skrócić czas wykonania algorytmu. W szczególności algorytm genetyczny jest dobrze skalowalny w środowisku wielowątkowym, co umożliwia równoczesne wykonywanie mutacji, krzyżowania i oceny przystosowania dla różnych osobników.

Podsumowując, złożoność algorytmu genetycznego jest ściśle zależna od parametrów problemu oraz implementacji, jednak dzięki swojej modularności i możliwości równoległego przetwarzania może być efektywnie wykorzystywany w praktycznych zastosowaniach.

2 Implementacja Algorytmu

3 Przykład Praktyczny

3.1 Algorytm Genetyczny

Rozważmy problem ATSP (Asymmetric Traveling Salesman Problem) dla poniższej macierzy kosztów:

$$D = \begin{bmatrix} -1 & 10 & 15 & 20 \\ 5 & -1 & 9 & 10 \\ 6 & 13 & -1 & 12 \\ 8 & 8 & 9 & -1 \end{bmatrix}$$

3.1.1 Legenda

- $D[i][j]$: koszt przejścia z miasta i do miasta j.
- -1: brak możliwości przejścia (miasta nie odwiedzają się same).

3.1.2 Parametry Algorytmu

Przyjmijmy następujące wartości parametrów:

- `populationSize`: 6 (liczba chromosomów w populacji).
- `mutationRate`: 0.2 (prawdopodobieństwo mutacji chromosomu).
- `randomRate`: 0.1 (odsetek losowo generowanych chromosomów w każdej generacji).
- `crossoverRate`: 0.8 (prawdopodobieństwo krzyżowania chromosomów).
- `iterationNum`: 10 (liczba generacji).
- `crossingSegmentSizeRate`: 0.5 (odsetek genów wymienianych podczas krzyżowania).
- `mutationSegmentSizeRate`: 0.3 (odsetek genów podlegających mutacji w chromosomie).
- `randomRateNewGen`: 0.2 (odsetek losowych chromosomów w nowej generacji).

3.1.3 Typ Mutacji

W tym przykładzie zastosowano mutację typu **inversion**, w której kolejność genów w losowo wybranym segmencie chromosomu zostaje odwrócona.

3.1.4 Kroki Algorytmu

1. Inicjalizacja Populacji Na początku algorytmu generujemy 6 losowych chromosomów, które reprezentują różne permutacje miast. Przykładowo:

$$\text{Chromosom } 1 : [1, 2, 3, 4] \quad \text{Koszt: } 10 + 9 + 12 + 8 = 39$$

$$\text{Chromosom } 2 : [1, 3, 4, 2] \quad \text{Koszt: } 15 + 12 + 8 + 5 = 40$$

2. Ocena Przystosowania (Fitness) Dla każdego chromosomu obliczamy koszt całkowity trasy. Celem algorytmu jest minimalizacja tego kosztu.

3. Krzyżowanie Zgodnie z parametrem `crossoverRate` (0.8), wybieramy 80% chromosomów do krzyżowania. Dla pary rodziców losowo wybieramy segment wymieniany między chromosomami na podstawie parametru `crossingSegmentSizeRate` (0.5).

Przykład krzyżowania dwóch chromosomów:

$$\text{Rodzic 1: } [1, 2, 3, 4], \quad \text{Rodzic 2: } [4, 3, 2, 1]$$

Segment wymieniany: $2, 3$. Wynik:

$$\text{Potomek 1: } [1, 3, 2, 4], \quad \text{Potomek 2: } [4, 2, 3, 1]$$

4. Mutacja Dla 20% chromosomów (zgodnie z `mutationRate`) stosujemy mutację typu **inversion**. Przykład:

$$\text{Chromosom przed mutacją: } [1, 2, 3, 4]$$

Losowy segment do odwrócenia: $2, 3$. Wynik:

$$\text{Chromosom po mutacji: } [1, 3, 2, 4]$$

5. Generowanie Nowej Populacji Na podstawie parametru `randomRateNewGen` (0.2), 20% nowej populacji stanowią losowo generowane chromosomy. Pozostałe miejsca wypełniamy najlepszymi osobnikami oraz ich potomstwem.

6. Powtarzanie Algorytm wykonuje powyższe kroki przez 10 iteracji (`iterationNum`).

3.1.5 Wynik Końcowy

Po 10 iteracjach otrzymujemy chromosom o najniższym koszcie, np.:

$$\text{Najlepszy chromosom: } [1, 3, 2, 4] \quad \text{Koszt: } 15 + 13 + 5 + 8 = 41$$

Wynik ten reprezentuje najkrótszą trasę znalezioną przez algorytm dla podanej macierzy kosztów.

4 Wyniki Eksperymentów

4.1 Specyfikacja sprzętowa

Wszystkie pomiary algorytmów wykonane w języku C++ na laptopie z następującymi parametrami:

- Procesor: Intel Core i5-12450 (bazowa częstotliwość 2.00 GHz)
- Pamięć RAM: 16 GB
- System operacyjny: Windows 10

- Środowisko programistyczne: Visual Studio Code
- Budowanie: cmake 3.29.2
- Kompilator: g++ 13.2.0
- Typ budowania: Release

Obliczenia, wykresy oraz analizę wyników przeprowadzono w języku Python 3.12.4 z wykorzystaniem bibliotek: `pandas, numpy, seaborn.`

5 Eksperymenty i Analiza Wyników

W ramach pracy przeprowadzono następujące eksperymenty:

- Eksperyment dla różnych rozmiarów populacji: $200, 500, 1000, 1500, 2000, 3000, 4000$ przy ustalonej liczbie iteracji (2000).
- Eksperyment dla stałej wartości współczynnika mutacji (*mutation rate*) oraz badanie wpływu zwiększenia procentu krzyżowania (*crossover rate*).
- Eksperyment dla stałej wartości krzyżowania oraz zmiennej wartości współczynnika mutacji.
- Eksperyment dla różnych typów mutacji (*mutation type*) oraz różnych rozmiarów segmentów mutacji.

5.1 Metodyka Eksperymentu

Do przeprowadzenia eksperymentów wykorzystano macierze sąsiedztwa dostępne na stronie *TspLib - Universität Heidelberg*.

Wszystkie pomiary wykonano dla ponad 20 instancji. Liczba instancji była dostosowywana w zależności od czasu trwania pomiarów. Parametry wstępne eksperymentu zostały ustalone na podstawie empirycznego doboru, w celu uzyskania jak najlepszych wyników.

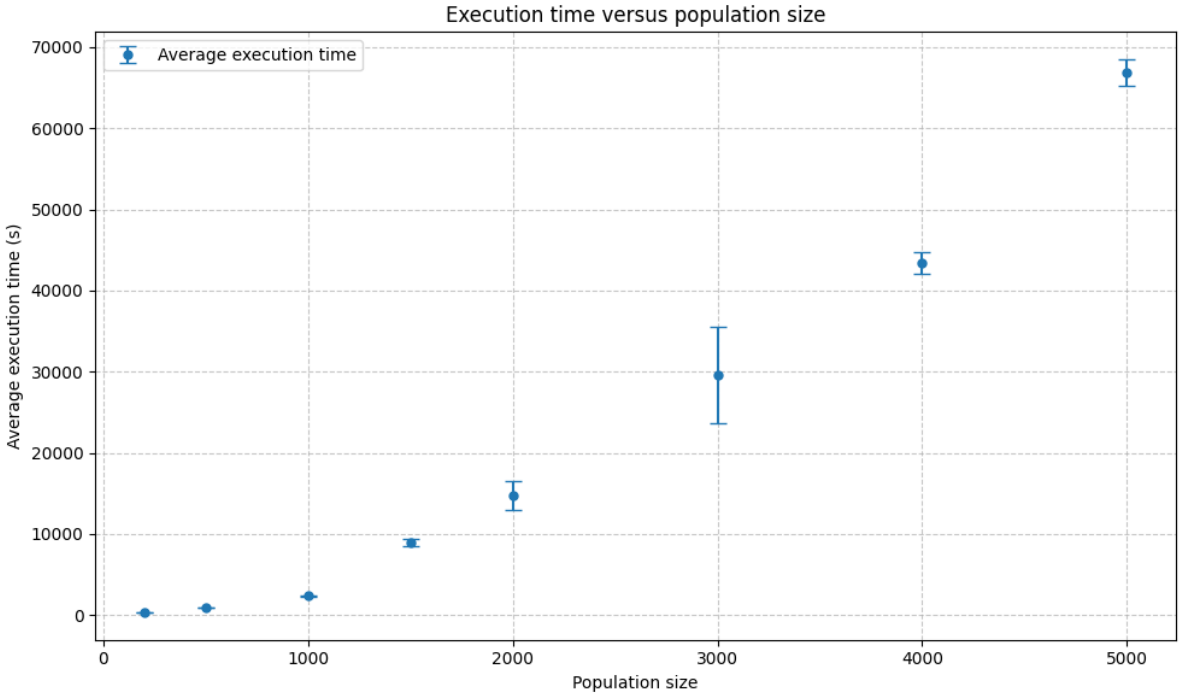
Przykładowy plik konfiguracyjny używany w eksperymentach:

```
{
    "geneticAlgorithmConfiguration": {
        "crossingSegmentSizeRate": 40,
        "mutationSegmentSizeRate": 40,
        "mutationType": "s",
        "populationSize": 2000,
        "iterationNum": 2000,
        "mutationRate": 20,
        "randomRate": 30,
        "randomRateNewGen": 20
    }
}
```

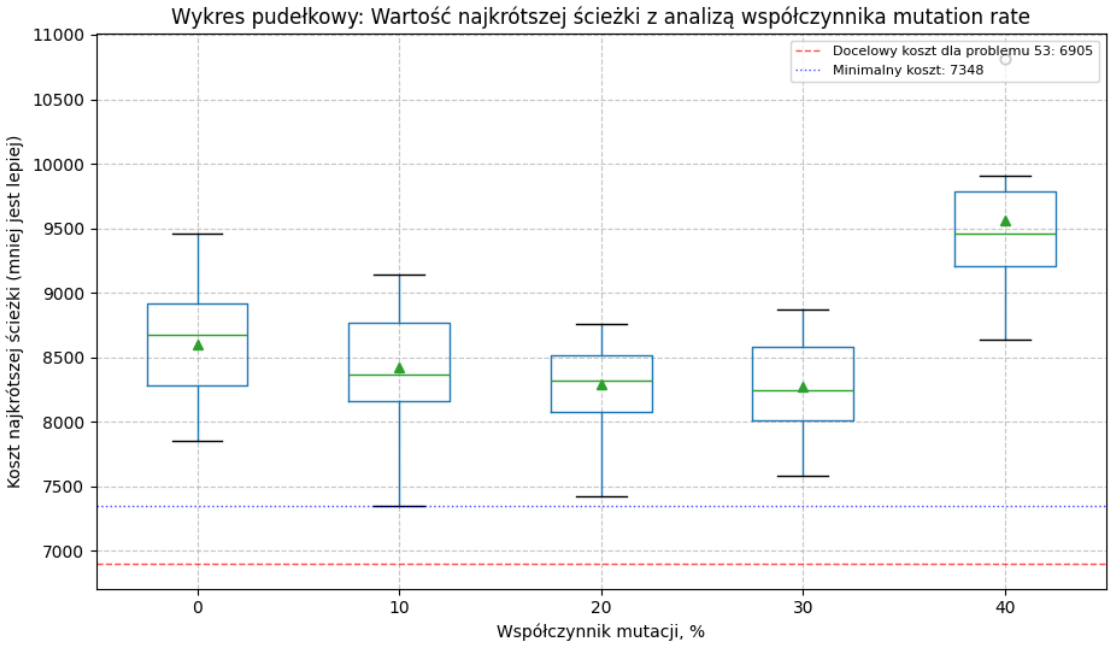
5.2 Wykresy Wyników

Do generowania wykresów wykorzystano skrypty napisane w Pythonie oraz biblioteki takie jak *matplotlib*, *numpy* i *seaborn*. Aby zilustrować różnorodność wyników oraz zakres błędu, część danych została zaprezentowana za pomocą wykresów pudełkowych.

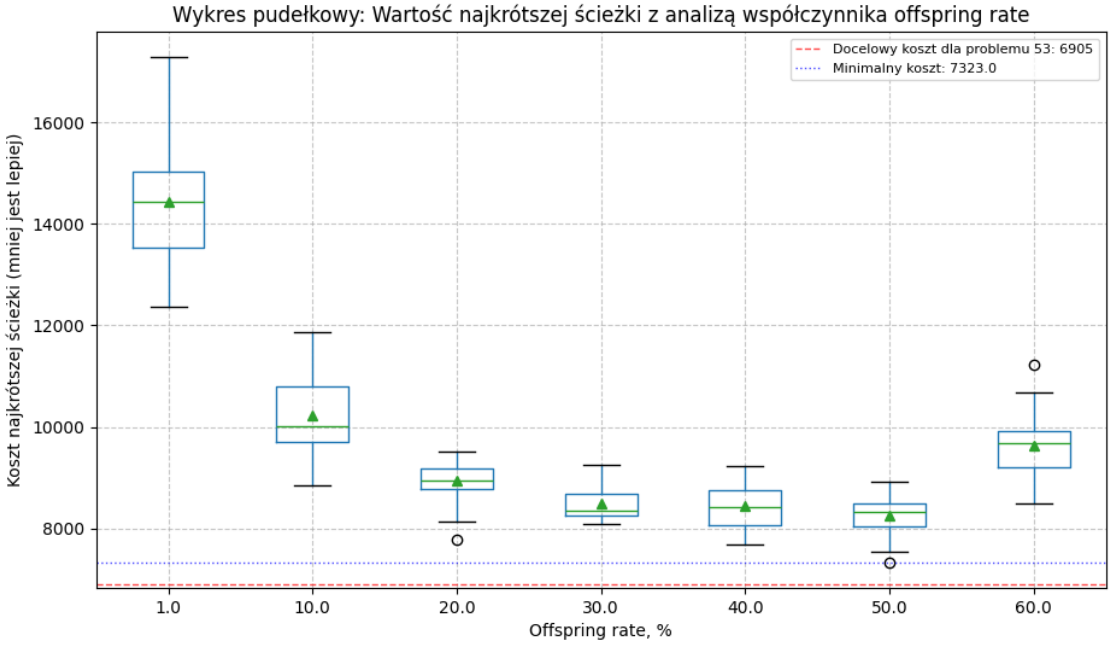
Rysunek 1: Wartość najkrótszej ścieżki w zależności od rozmiaru populacji (wykres pudełkowy).



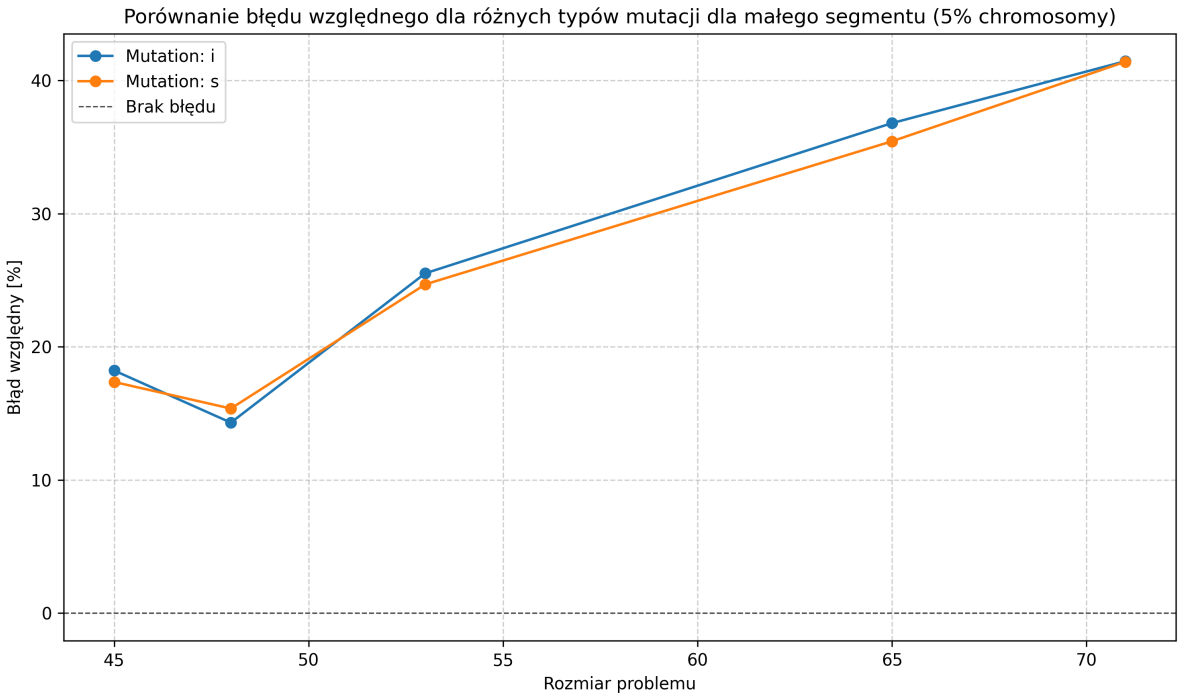
Rysunek 2: Czas wykonania algorytmu w zależności od rozmiaru populacji.



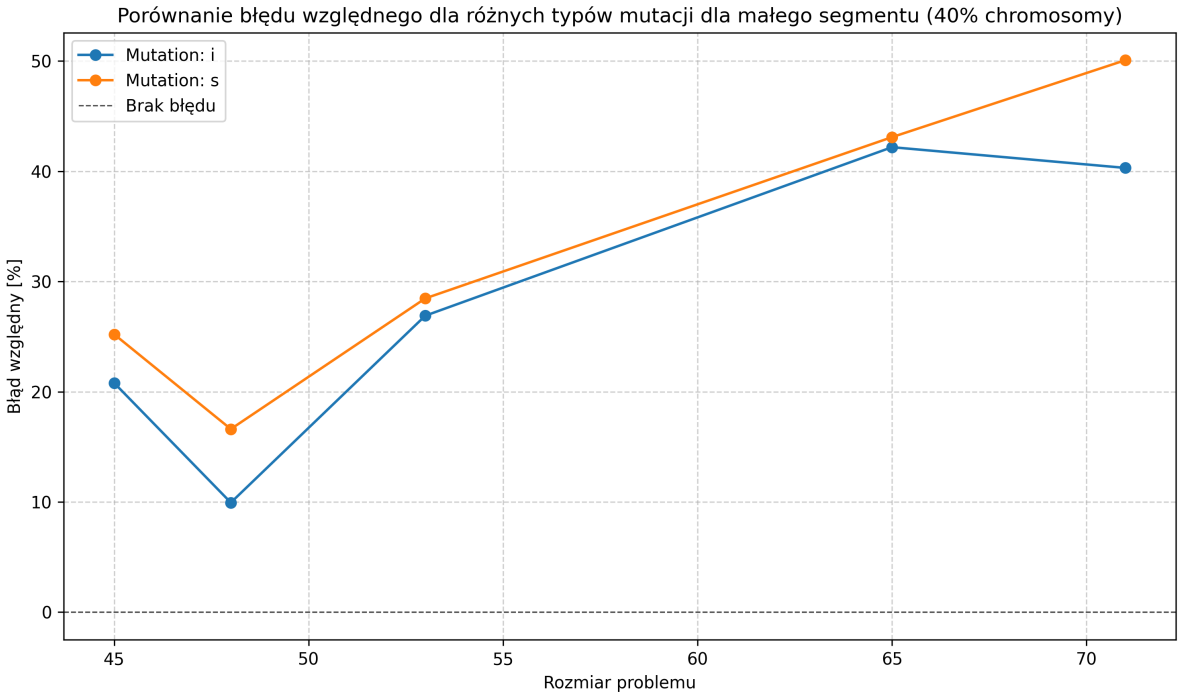
Rysunek 3: Zależność wyników od współczynników mutacji.



Rysunek 4: Zależność wyników od współczynników krzyżowania.



Rysunek 5: Rozmiar błędu względnego dla problemów różnego rodzaju dla dwóch typów mutacji: *shuffle* i *inverse* (rozmiar segmentu 5% długości chromosomu).



Rysunek 6: Rozmiar błędu względnego dla problemów różnego rodzaju dla dwóch typów mutacji: *shuffle* i *inverse* (rozmiar segmentu 40% długości chromosomu).

5.3 Analiza Wyników

Na podstawie przeprowadzonych eksperymentów oraz analizy wyników przedstawionych na wykresach można wyciągnąć następujące wnioski:

1. **Rozmiar populacji**: Zbyt mały rozmiar populacji negatywnie wpływa na różnorodność osobników, prowadząc do szybkiej konwergencji do lokalnego minimum, co skutkuje gorszymi wynikami (rysunek 1). Z kolei zbyt duże rozmiary populacji powodują wykładniczy wzrost czasu generacji potomstwa (rysunek 2). Optymalizacja tego parametru powinna uwzględniać cele algorytmu oraz priorytety, takie jak dokładność i czas wykonania.

2. **Parametry mutacji i krzyżowania**: Parametry takie jak współczynnik mutacji oraz krzyżowania odgrywają kluczową rolę w efektywności algorytmu. Jak przedstawiono na rysunkach 3 i 4, wyniki wykazują charakterystykę paraboliczną z asymetrią względem zakresu współczynników. Odpowiednie ich dobranie zwiększa różnorodność populacji, a generacja losowych chromosomów dodatkowo wzmacnia eksplorację przestrzeni rozwiązań.

3. **Typ mutacji**: Typ mutacji nie ma istotnego wpływu na jakość rozwiązania przy małych rozmiarach segmentów mutujących (rysunek 5). Jednak przy większych segmentach (40% długości chromosomu) różnice stają się wyraźne, a mutacja typu *inverse* wykazuje lepsze rezultaty (rysunek 6).

6 Wnioski

Przeprowadzone eksperymenty potwierdzają kluczową rolę parametrów takich jak rozmiar populacji, współczynniki mutacji oraz krzyżowania w optymalizacji działania algorytmu genetycznego.

1. Wybór rozmiaru populacji powinien uwzględniać kompromis między różnorodnością a czasem wykonania.

2. Optymalizacja współczynników mutacji i krzyżowania umożliwia osiągnięcie lepszej równowagi między eksploracją a eksploatacją przestrzeni rozwiązań.

3. Typ mutacji staje się istotny przy większych rozmiarach segmentów mutujących, co sugeruje konieczność dalszych badań nad efektywnością różnych podejść w zależności od charakterystyki problemu.

4. Algorytm genetyczny (GA) wyróżnia się na tle innych metod heurystycznych i algorytmów deterministycznych.

    - W porównaniu do **Brute Force**, GA jest zdecydowanie bardziej efektywny czasowo, ponieważ unika pełnego przeszukiwania przestrzeni rozwiązań, co dla większych problemów staje się praktycznie niewykonalne.
    - W porównaniu z **Branch and Bound**, GA jest bardziej elastyczny i skuteczny w problemach o dużych rozmiarach, gdzie dokładność rozwiązania może być poświęcona na rzecz krótszego czasu obliczeń.

- W porównaniu z **Tabu Search**, GA oferuje większą różnorodność eksploracji dzięki swojej strukturze opartej na populacji. Tabu Search, choć szybki i skuteczny w lokalnym przeszukiwaniu, może utknąć w lokalnych minimach, jeśli przestrzeń rozwiązań nie zostanie odpowiednio eksplorowana.

Algorytm genetyczny najlepiej sprawdza się w problemach wymagających zrównoważenia eksploracji (badania różnych rozwiązań) i eksploatacji (ulepszania najlepszych znalezionych rozwiązań). Jego skuteczność jest szczególnie widoczna w zadaniach o dużej złożoności i nieznanej strukturze funkcji celu.

5. **Podsumowanie efektywności**: Algorytm genetyczny, mimo większej liczby parametrów do optymalizacji, oferuje skalowalność i elastyczność, co czyni go atrakcyjnym wyborem w szerokim zakresie problemów optymalizacyjnych.

Powyższe obserwacje i porównania sugerują, że algorytm genetyczny jest wszechstronnym narzędziem, które można dostosować do różnorodnych zastosowań, zwłaszcza tam, gdzie tradycyjne metody deterministyczne lub heurystyczne mają swoje ograniczenia.

7 Literatura

1 Neapolitan R., Naimipour K.: *Podstawy algorytmów z przykładami w C++* (rozdział 6.), 2004

2 *TspLib - Universität Heidelberg*

3 *Introduction to Genetic Algorithm n application on Traveling Sales Man Problem (TSP)*