Projektowanie efektywnych algorytmów

Projekt 3

ATSP – Algorytm Genetyczny

Mateusz Tesarewicz 272909

27.01.2025 Politechnika Wrocławska

Spis treści

1 Wstęp

Asymetryczny problem komiwojażera (ATSP, z ang. Asymmetric Traveling Salesman Problem) stanowi jedno z kluczowych wyzwań optymalizacyjnych w teorii grafów oraz badań operacyjnych. Zadanie to polega na wyznaczeniu najkrótszej możliwej trasy, która pozwala komiwojażerowi odwiedzić wszystkie zadane miasta dokładnie raz i powrócić do punktu początkowego, przy czym odległości między miastami mogą się różnić w zależności od kierunku przejścia (asymetria). ATSP znajduje szerokie zastosowanie w logistyce, planowaniu tras oraz optymalizacji produkcji, a także w systemach komunikacji miejskiej i transporcie.

Ze względu na swoją złożoność obliczeniową (problem należy do klasy NP-trudnych), rozwiązywanie ATSP wymaga zastosowania specjalistycznych algorytmów, takich jak algorytmy przeglądu zupełnego, podziału i ograniczeń lub programowania dynamicznego. Celem niniejszego badania jest ocena efektywności wybranych algorytmów stosowanych do rozwiązywania ATSP, poprzez analizę ich wyników w kontekście jakości uzyskanych rozwiązań.

2 Badane algorytmy

2.1 Algorytm Genetyczny

Algorytm genetyczny to metaheurystyczna metoda optymalizacji inspirowana procesami ewolucyjnymi, która polega na iteracyjnym poszukiwaniu najlepszego rozwiązania problemu poprzez symulację mechanizmów takich jak selekcja, krzyżowanie i mutacja. Proces rozpoczyna się od losowego wygenerowania populacji potencjalnych rozwiązań, z których każde oceniane jest za pomocą funkcji przystosowania określającej jego jakość. Następnie na podstawie wartości przystosowania wybierane są rozwiązania do kolejnego pokolenia (selekcja), przy czym lepsze osobniki mają większe szanse na

wybór. Wybrane rozwiązania są następnie modyfikowane – poprzez wymianę informacji genetycznej w procesie krzyżowania oraz losowe zmiany w procesie mutacji – co prowadzi do powstania nowej populacji. Algorytm iteruje ten proces, dążąc do znalezienia rozwiązań o jak najwyższej jakości. Działanie algorytmu kończy się po osiągnięciu określonego kryterium, np. ustalonej liczby iteracji, odpowiedniego poziomu jakości rozwiązania lub osiągnieciu limitu czasu.

3 Opis działania algorytmu

3.1 Pseudokod algorytmu genetycznego

```
geneticAlgorithm()
    population = generateRandomPopulation()
    fitness = initFitness()
    bestRoute = initBestRoute()
    while not timeExceed():
        //Obliczanie przystosowania
        fitness = calculateFitness(population)
        //Tworzenie nowej populacji
        newPopulation = []
        //Elityzm
        copyBestChromosomes(populatio, newPopulation)
        //Reszta populacji - krzyżowanie i mutacja
        while not populationCompleted(newPopulation):
            //Selekcja rodziców
            parent1 = []
            parent2 = []
            selection(population,fitness,parent1, parent2)
            //Krzyzowanie
            child1 = []
            child2 = []
            if (corssoverProbability()):
                crossover(parent1,parent2, child1)
                crossover(parent2,parent1, child2)
            else:
                child1 = parent1
                child2 = parent2
            //Mutacja
            if (mutationProbability()):
                mutate(child1)
            if (mutationProbability()):
                mutate(child2)
            addChildrenToPopulation(newPopulation,child1,child2)

            //Sukcesja
            population = newPopulation

        bestRoute = findBestRoute(population)
    return bestRoute
```

3.2 Metody krzyżowania

3.2.1 Order crossover

```
oderCrossover(parent1, parent2, child):
    //Losowanie dwóch różnych indeksów
    start, end = generateRandomGap()
    //Kopiowanie wylosowanego fragmentu z parent1 do child
    child[start:end] = parent1[start:end]
    //Kopiowanie reszty miast do child w kolejności parent2
    parent2Pos = end
    childPos = end
    while not childCompleted():
        parent2Pos = ++parent2Pos % parent2.size()
        if parent2[parent2Pos] not in child:
            childPos = ++childPos % parent2.size()
            child[childPos] = parent2[parent2Pos]
```

3.2.2 PMX crossover

```
pmxCrossover(parent1, parent2, child):
    //Losowanie dwóch róznych indeksów
    start, end = generateRandomGap()
    //Kopiowanie wylosowanego fragmentu z parent1 do child
    fragment = parent[start:end]
    child[start:end] = fragment
    //Kopiowanie elementów z parent2, które są w fragmencie do child
    for  i in range(start, end):
        //Patrzymy na analogiczny fragment w P2 i szukamy elementów
        //które nie zostały skopiowane z p1
        if not parent2[i] in fragment:
            //Szukamy pozycji elementu w p2, który znajduje się
            //Na pozycji i w parent1
            elPos = findEl(parent1[i],parent2)  //(target, array)
            //Rekurencyjne przeszukiwanie wolnej pozycji w child
            while child[pos] not free:
                //sprawdzamy jaka liczba jest na pozycji elPos
                //I sprawdzamy, gdzie znajduje się ta liczba w parent2
                elPos = findEl(parent1[elPos], parent2)
            child[elPos] = parent2[i]

    //Kopiowanie reszty elementów z parent2
    for i in range(parent2.size()):
        if child[i] is free:
            child[i] = parent2[i]
```

3.3 Metody mutacji

3.3.1 Inverse mutation

```
mutation(child):
    //Losowanie dwóch różnych indeksów
    start, end = generateRandomGap()
    reverse(child[start:end])
```

4 Plan przeprowadzenia badania

Poprawnie zaplanowanie przebiegu badania jest bardzo kluczowe, aby uniknąć ewentualnych komplikacji lub co gorsza, niepoprawnych wyników przeprowadzonego badania. Badaniu poddamy algorytm genetyczny dla jednej macierzy kosztów i różnych metod krzyżowania oraz różnych rozmiarów populacji. Odległości między miastami zostaną przedstawione w postaci macierzy kwadratowej. Odległości będą dodatnimi liczbami całkowitymi.

4.1 Środowisko i specyfikacja sprzętu

Do przeprowadzenia badania został napisany obiektowy program w języku C++ kompilowany w środowisku CLion. Badania zostaną przeprowadzone na komputerze o podanej specyfikacji: Procesor :AMD Ryzen 7 5700X 8-Core Processor 3.40 GHz Pamięć RAM: 16 GB 3200 MHz System operacyjny: Windows 10 Home Typ systemu: 64-bitowy system operacyjny, procesor x64

4.2 Przedstawienie odległości między miastami

Odległości między miastami będą zapisane w kwadratowej macierzy, gdzie numer wiersza będzie oznaczał miasto początkowe, a numer kolumny miasto końcowe. Wartością w macierzy będzie odległość między miastem początkowym a końcowym. Drogi między miastami będą zazwyczaj asymetryczne. Po przekątnej macierzy wpisane zostaną wartości -1 lub 0, żeby wykluczyć drogi z miast do siebie nawzajem. W innych komórkach będą mogły się znajdować tylko liczby naturalne dodatnie. Aby wczytać macierz, należy przygotować ją w pliku o rozszerzeniu .atsp.

4.3 Badane parametry

Badaniu poddajemy dwie metody krzyżowania: OX i PMX oraz 3 rozmiary populacji: 100, 200 i 500 osobników. Populacja o rozmiarze 50 została pokazana w ramach ciekawostki. Sprawdzimy wpływ tych parametrów na

otrzymywane wyniki w stałym czasie i stałej macierzy kosztów. Dla najlepszego rozwiązania pokażemy zmianę średniego błędu względnego w jednostce czasu, czyli jak średni błąd ewoluuje wraz z przebiegiem algorytmu.

4.4 Pozostałe parametry Algorytmu Genetycznego

Algorytm genetyczny posiada więcej parametrów niż te, które badamy, dlatego należy pamiętać też o nich.

Prawdopodobieństwo mutacji = 0.01

Prawdopodobieństwo krzyżowania = 0.8

Tournament size = 3

Współczynnik elityzmu = 0.2

Czas działania algorytmu = 240 sekund

Macierz kosztów = plik ftv170.atsp zawierający 171 miast z asymetrycznymi drogami.

Podczas przeprowadzania testów wszystkie te wartości pozostawały bez zmian.

4.5 Przedstawienie wyników

Wyniki zostaną przedstawione w postaci tabeli, w której będą zawarte informacje o najlepszym średnim błędzie względnym i najlepszym wyniku dla każdej próby algorytmu i każdej kombinacji badanych parametrów. Łącznie będziemy mieć 6 testów po 10 prób.

Dodatkowo zbierzemy najlepsze wyniki otrzymane z każdego testu i przedstawimy na wykresie zmianę uśrednionego współczynnika błędu względnego w funkcji czasu działania algorytmu. Pozwoli to nam zobaczyć jak czas działania algorytmu wpływa na znajdowanie lepszych rozwiązań.

4.6 Czas działania algorytmu

Obecne badanie jest powiązanie z badaniem poprzednim, więc zachowujemy czasy ustalone w poprzednim badaniu dla różnych rozmiarów macierzy. W obecnym badaniu badamy wpływy parametrów dla jednego pliku: ftv170.atsp, który wcześniej był badany w czasie 240 sekund i tej wartości trzymamy się również tym razem.

4.7 Przebieg eksperymentu

Zaimplementowany program jest rozszerzeniem programu z poprzedniego projektu o Algorytm Genetyczny wraz z dwiema metodami krzyżowania i jedną metodą mutacji. Dane są wczytywane z pliku o rozszerzeniu .atsp. Tym razem badanie przeprowadzamy tylko na jednym pliku: ftv170.atsp z ustawionym czasem na 240 sekund. Plik ftv170.atsp zawiera macierz kosztów dla 171 miast, trasy są asymetryczne. Badaniu musimy poddać obie metody krzyżowania i 3 rozmiary populacji, jest to łącznie 6 testów. Dla każdej kombinacji zostanie włączony test, który poda 10 wyników (prób). W wynikach zawiera się najkrótsza ścieżka, czas znalezienia najkrótszej ścieżki, średnie błędy względne liczone w interwale co 10 sekund.

5 Wyniki badań

5.1 Główna tabela z wynikami badania

Tabela 1 Główne wyniki

Order Crossover								
	populacja 50		populacja 100		populacja 200		populacja 500	
Próba	średnia	najlepszy	średnia	najlepszy	średnia	najlepszy	średnia	najlepszy
1	39,31%	3831	37,28%	3760	28,64%	3541	29,07%	3534
2	35,39%	3730	37,21%	3769	51,47%	4155	28,24%	3519
3	29,66%	3568	29,33%	3552	39,64%	3840	38,58%	3803
4	40,94%	3883	49,80%	4114	29,15%	3539	22,03%	3354
5	32,63%	3654	43,09%	3921	32,23%	3643	37,21%	3775
6	55,10%	4241	34,33%	3701	34,05%	3690	28,01%	3516
7	42,11%	3915	34,08%	3666	35,06%	3713	28,42%	3519
8	35,21%	3723	62,58%	4468	26,53%	3479	26,84%	3479
9	38,69%	3821	45,76%	4015	42,61%	3926	34,45%	3688
10	41,67%	3903	59,71%	4367	40,80%	3846	39,02%	3827
PMX Crossover								
	populacja 50		populacja 100		populacja 200		populacja 500	
Próba	średnia	najlepszy	średnia	najlepszy	średnia	najlepszy	średnia	najlepszy
1	172,48%	7477	170,60%	7455	202,29%	8320	189,11%	7965
2	205,51%	8417	194,77%	8121	187,99%	7925	197,46%	8188
3	185,12%	7855	173,61%	7538	193,32%	8065	189,07%	7963
4	198,88%	8234	184,21%	7830	193,87%	8096	192,34%	8045
5	165,60%	7317	174,84%	7553	217,24%	8732	159,02%	7123
6	187,91%	7932	178,48%	7654	196,12%	8121	186,86%	7890
7	205,95%	8429	163,05%	7247	168,75%	7396	155,75%	7024
8	167,22%	7362	180,18%	7719	181,38%	7734	169,73%	7429
9	201,34%	8302	179,09%	7689	184,14%	7813	157,58%	7089
10	181,77%	7763	183,19%	7782	199,46%	8236	183,67%	7789

Początkowo przeprowadziłem badania dla populacji 50, 100 i 200, ale nie zauważyłem większych różnic, dlatego postanowiłem zrobić dodatkowe testy dla populacji 500 osobników i stwiedziłem, że głównymi parametrami populacji zostaną 100, 200 i 500. Mimo tego wklejam testy dla 50 jako ciekawostka i dodatkowe informacje.

5.1.1 Opisy kolumn

Średnia – jest to najlepszy średni błąd względny dla danego testu, który wystąpił podczas przebiegu algorytmu. Średni błąd względny jest liczony według danego schematu:

1. W każdej iteracji algorytmu genetycznego po wygenerowania populacji otrzymujemy zestaw wyników $R_1, R_2 R_3, \dots, R_N$, gdzie N to rozmiar populacji
2. Liczymy średnią wartość funkcji celu dla wszystkich osobników w populacji:

$$Średni\ wynik\ populacji = \frac{\sum_{i=1}^{N} R_i}{N}$$

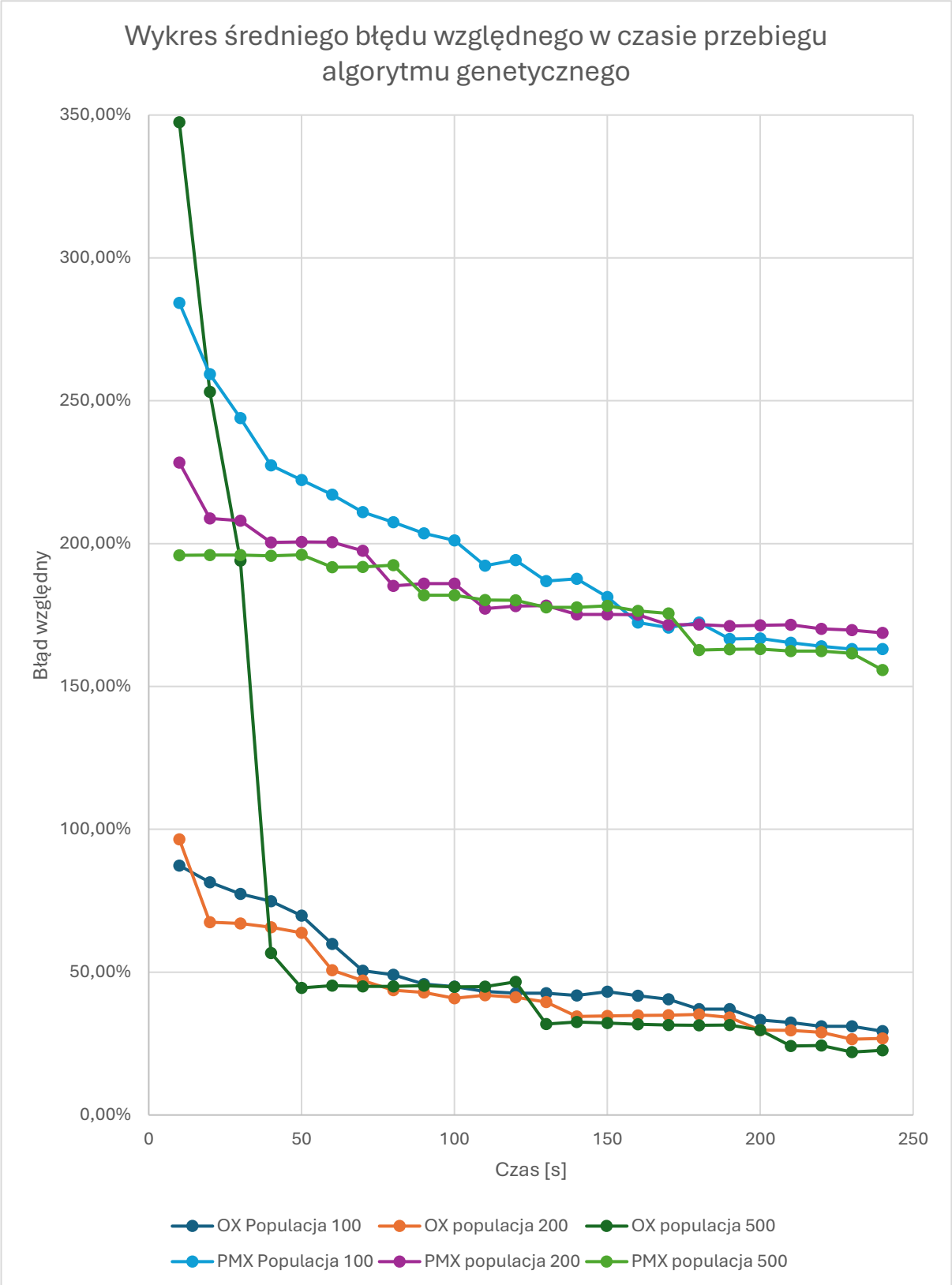
3. Porównujemy średni wynik z wynikiem optymalnym, aby obliczyć średni błąd względny:

$$Średni\ błąd\ względny = \frac{|średni\ wnik\ populacji - Opt|}{Opt} \times 100\%$$

W moim przypadku średni błąd względny był liczony co 10 sekund działania algorytmu i zapisywani do listy. Po wykonaniu całego testu wybierałem najlepszą wartość, która znajduje się w tabeli.

Najlepszy – W tej kolumnie znajduję się najlepsze wyniki funkcji celu otrzymane w każdej próbie.

5.2 Wykres porównujący przebiegi średnich błędów względnych



Rysunek 1 Główny wykres zależności średniego błędu względnego od czasu działania algorytmu

5.3 Wnioski

Metoda krzyżowania OX (Order crossover) radzi sobie o wiele lepiej i dzięki niej algorytm może konkurować z Algorytmem Tabu Search lub Symulowanego wyżarzania. Metoda krzyżowania PMX znacznie odbiega od optymalnego rozwiązania. Kolejną rzeczą, którą możemy zauważyć to wpływ rozmiaru implementacji na średni błąd względny, który się zmniejsza wraz z rozmiarem populacji. Oznacza to, że większe rozmiary populacji sprzyjają algorytmowi. Czas również sprzyja algorytmowi, ponieważ widać, że z przebiegiem czasu zmniejsza się średni błąd względny.

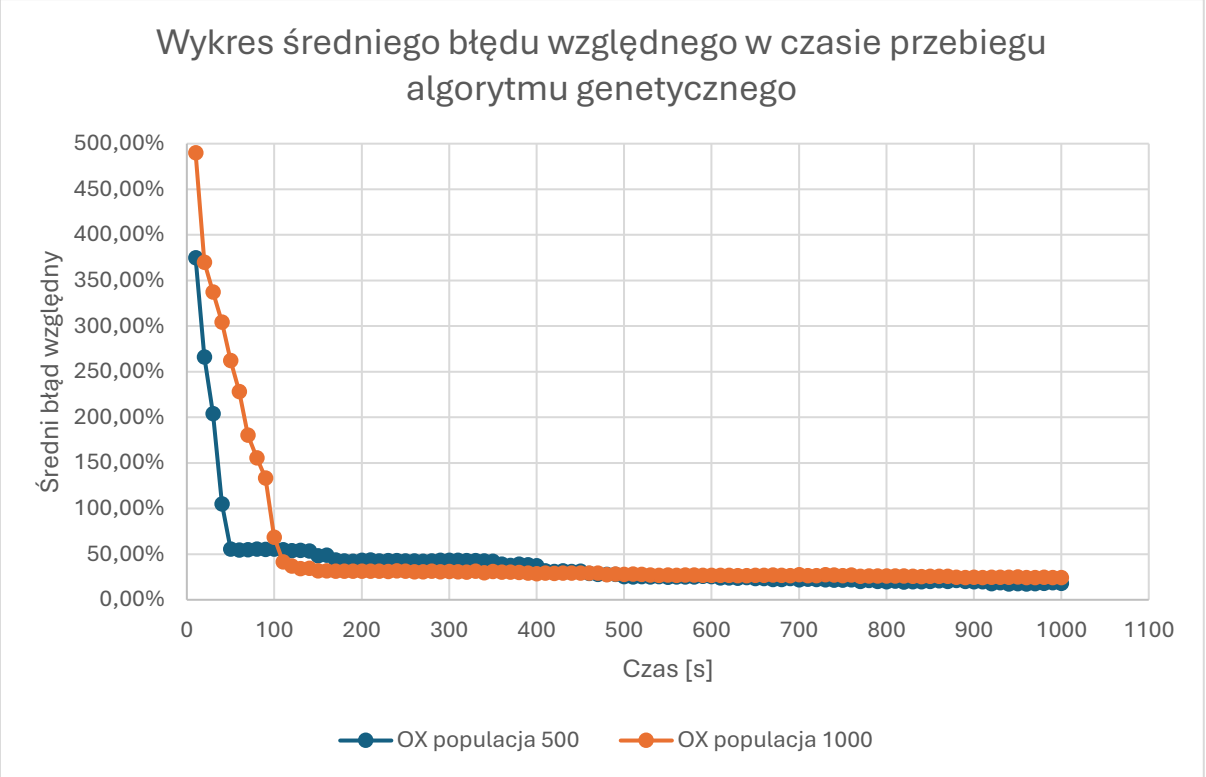
5.4 Porównanie wyników z algorytmami Tabu Search i Symulowanego wyżarzania

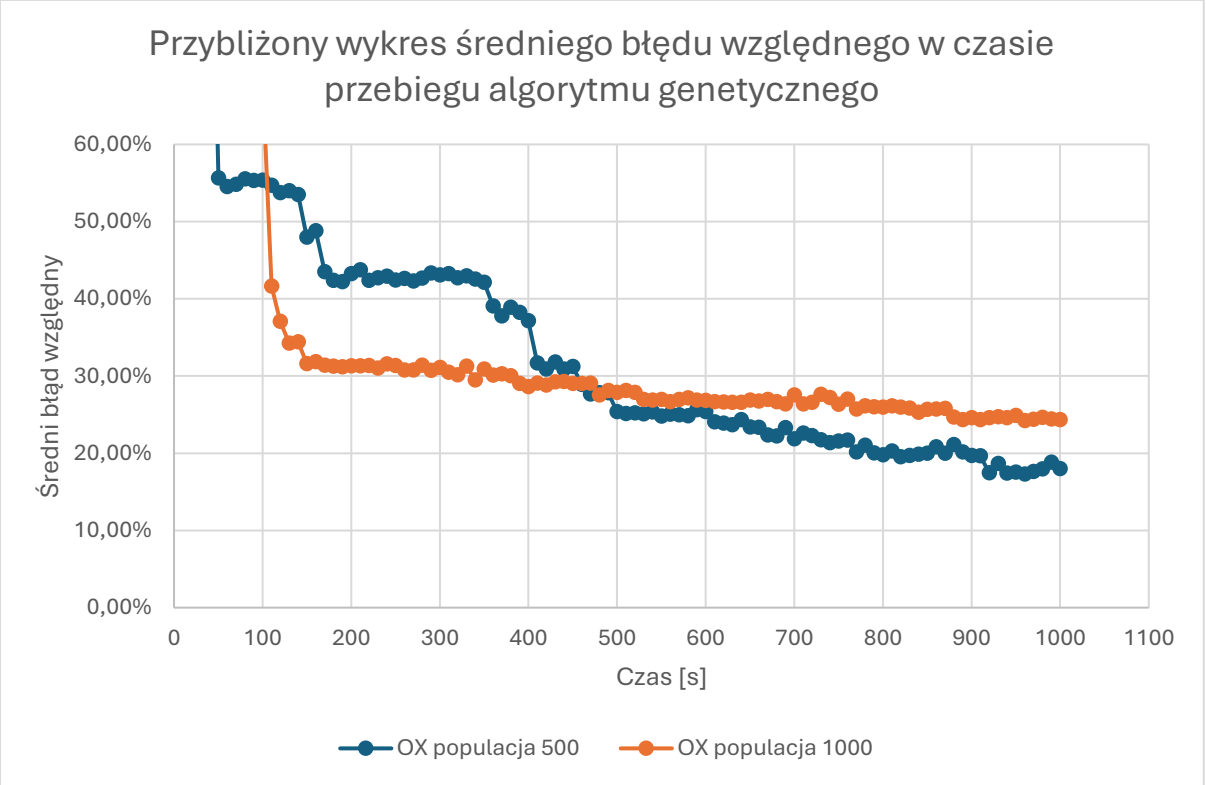
Tabela 2 Porównanie wyników z TS i SA

Algorytm → / Informacje ↓	Genetic Algorithm	Tabu Search	Simulated Annealing
najlepsze rozwiązanie	3354	3285	2818
Czas znalezienia	222,813	86,446	1,27
Błąd względny	21,74%	19,24%	2,29%

Porównując Algorytm Genetyczny z algorytmem Tabu Search i Symulowanym wyżarzaniem możemy stwierdzić, że radzi on sobie najgorzej, lecz wynik nie odbiega znacznie od wyniku uzyskanego za pomocą Tabu Search. Różnicą jest czas otrzymania wyniku, Algorytm Genetyczny działa znacznie wolniej i potrzebuje więcej czasu na znalezienie lepszego rozwiązania.

6 Dodatek – przebieg algorytmu w dłuższym czasie i dla większej populacji





Rysunek 2 Wykres zależności średniego błędu względnego od czasu dla dłuższego przebiegu algorytmu

Jako dodatek przetestowałem działanie algorytmu w czasie **1000** sekund dla populacji **500** oraz **1000**. Chciałem sprawdzić zależność czasu od generowania lepszych wyników. Po tym czasie otrzymałem najlepszy wynik równy **3286** dla rozmiaru populacji 500 oraz wynik **3410** dla populacji o rozmiarze 1000. Oznacza to, że znaczne wydłużanie czasu nieznacznie polepsza wynik algorytmu. Kolejną rzeczą jest rozmiar populacji, który również posiada swoje ograniczenia i nie może być nieskończenie duży. Możemy zauważyć, że algorytm dla populacji o rozmiarze 1000 poradził sobie gorzej od rozmiaru 500. Po wykresie widać, że większa populacja przez pewien czas radziła sobie lepiej, ale finalnie mniejsza populacja ją wyprzedziła. Może to wynikać z tego, że większa populacja posiada więcej różnych rozwiązań, więcej rozwiązań jest przenoszonych za sprawą elityzmu lub braku krzyżowania, może to wpływać na ustabilizowanie się algorytmu i słabszą eksplorację. Nie możemy traktować tych parametrów jako najlepsze dla ogółu, ponieważ moim zdaniem rozmiar problemu ma znaczny wpływ na działanie algorytmu z określonymi parametrami. Wcześniej mogliśmy zauważyć gorsze działania algorytmu dla mniejszych populacji, ale tutaj znaleźliśmy górną granicę, która mówi nam, że więcej nie oznacza lepiej. Dla dobrego działania przy określonych rozmiarach problemów trzeba znaleźć złoty środek rozmiaru populacji oraz poeksperymentować z dodatkowymi parametrami, które znajdują się w algorytmie.

7 Podsumowanie

Algorytm genetyczny jest najsłabszym algorytmem spośród trzech algorytmów, które były porównywane wyżej. Mimo tego wyniki nie odbiegają od reszty i posiadają około dwudziestoprocentowy błąd względny, który moim zdaniem jest akceptowalny dla problemów takich rozmiarów. Niestety przy użyciu metody krzyżowania PMX wyniki są o wiele gorsze i ciężko stwierdzić co jest przyczyną. Sama metoda została zaimplementowana zgodnie z regułą i

nie powinny występować w niej żadne błędy implementacji. Problemem może być układ miast i dróg między nimi w pliku, na którym były przeprowadzane badania. Zakres badania, który ogranicza się do jednego pliku nie pozwala nam zweryfikować, czy problem jest związany właśnie z tym. Algorytm cechuje się swoją prostotą w implementacji i nawiązaniem do natury. Kolejną zaletą jest wpływ czasu na poprawę wyniku, gdzie u pozostałych algorytmów nie zawsze było to widoczne. Algorytm posiada naprawdę wiele parametrów, które można testować i na pewno można je dopasowywać do indywidualnych plików, aby algorytm działał lepiej, dodatkowo istnieje wiele metod krzyżowania i mutacji oraz wyboru rodziców, które również mogłyby usprawnić algorytm i uczynić go znacznie lepszym. Trudnością, którą sprawia nam algorytm Genetyczny jest brak uniwersalnego wzoru dobierania parametrów, ale z drugiej strony otwiera nam drogę na własne eksperymentowanie z parametrami i dopasowanie ich do indywidualnych plików. Algorytm genetyczny na pewno zapadnie mi w pamięci ze względu na swoją prostotę implementacji i logikę działania, która jest związania z naturą świata.

8 Źródła

www.stackoverflow.com

https://www.aragorn.wi.pb.edu.pl/~wkwedlo/EA5.pdf

Wykłady dr. Inż. Tomasza Kapłona