Projektowanie Efektywnych Algorytmów

Projekt

24.01.2024

263900 Aleksandra Chrustek

(4) Algorytm genetyczny

spis treści	*strona*

1 Sformułowanie zadania

Zadanie polega na opracowaniu, napisaniu i zbadaniu problemu komiwojażera w wersji optymalizacyjnej algorytmem genetyczny. Problem komiwojażera (eng. Travelling salesman problem, TSP) to zagadnienie polegające (w wersji optymalizacyjnej) na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

1. Graf pełny to zbiór wierzchołków, przy czym między każdymi dwoma wierchołkami istnieje krawędź je łącząca. [1]
2. Cykl Hamiltona to droga wiodąca przez wszystkie wierzchołki dokładnie raz, z wyjątkiem jednego wybranego, w którym cykl Hamiltona zaczyna się oraz kończy. [2]

Problem komiwojażera rozumiemy jako zadanie polegające na znalezienu najlepszej drogi dla podróżującego chcącego odwiedzić n miast i skończyć podróż w miejscu jej rozpoczęcia. Połączenie między każdym miastem ma swój koszt określający efektywnosć jej przebycia. Najlepsza droga to taka, której całkowity koszt (suma kosztów przebycia wszystkich połączeń między miastami na drodze) jest najmiejszy. Problem dzieli się na symetryczny i asymetryczny. Pierwszy polega na tym, że dla dowolnych miast A i B z danej instancji, koszt połączenia jest taki sam w przypadku przebycia połączenia z A do B, jak z B do A, czyli dane połączenie ma jeden koszt niezależnie od kierunku ruchu. W asymetrycznym problemie komiwojażera koszty te mogą być różne.

2 Metoda

Algorytm genetyczny polega na stworzeniu populacji, gdzie każda jednostka posiada pewne cechy. Następnie w kolejnych iteracjach algorytmu symulowane jest krzyżowanie genetyczne między najlpiej przystosowanymi osobnikami (w przypadku TSP są to najniższe koszty przejścia ścieżki). Rodziców wybiera się jedną z metod selekcji, w przypadku tego programu jest to ruletka. Następnie nowa jednostka może zostać zmutowana, w losowy sposób (w przypadku TSP losowa modyfikacja ścieżki, np. swap dwóch losowych wierzchołków). Kiedy wygenerowana zostanie nowa populacja, następuje połączenie jej z już istniejącą w wybrany przez nas sposób, np. usunięcie wszystkich starych jednostek.

1. Początkowa populacja: Tworzona jest populacja początkowa, która składa się z osobników. Każdy osobnik reprezentuje potencjalne rozwiązanie problemu. Populacja ta jest zazwyczaj generowana losowo.

2. Funkcja oceny (fitness): Dla każdego osobnika w populacji obliczana jest wartość funkcji oceny (fitness), która mierzy jakość danego rozwiązania. Funkcja ta jest zdefiniowana na podstawie celu optymalizacji.

3. Selekcja: Osobniki z populacji są wybierane do reprodukcji na podstawie ich wartości fitness. Osobniki o lepszym dopasowaniu mają większą szansę na zostanie wybranymi.

4. Krzyżowanie (crossover): Wybrane osobniki są łączone, aby utworzyć nowe potomstwo. Proces krzyżowania ma na celu połączenie cech rodziców, aby uzyskać potomstwo z potencjalnie lepszymi cechami.

5. Mutacja: Losowe zmiany są wprowadzane w genotypie potomstwa. Mutacja pomaga wprowadzić różnorodność genetyczną i przeciwdziała zbyt szybkiemu zbiegowi do lokalnych optimum.

6. Nowa populacja: Na podstawie wyników krzyżowania i mutacji tworzona jest nowa populacja.

7. Kryterium zakończenia: Sprawdzane jest kryterium zakończenia, na przykład liczba generacji, osiągnięcie określonej wartości fitness, czy inny warunek stopu. Jeżeli warunek jest spełniony, algorytm kończy działanie; w przeciwnym razie powraca do kroku 2.

Schemat działania algorytmu przedstawia się następująco:

1. Inicjalizacja: Losowo generuj początkową populację osobników. Każdy osobnik reprezentuje potencjalne rozwiązanie problemu.

2. Ocena (Fitness): Dla każdego osobnika w populacji oblicz wartość funkcji oceny (fitness), która mierzy jakość rozwiązania. Im wyższa wartość fitness, tym lepsze rozwiązanie.

3. Selekcja: Wybierz osobniki do reprodukcji na podstawie ich wartości fitness. Osobniki o lepszym dopasowaniu mają większą szansę na zostanie wybranymi. Różne strategie selekcji mogą być stosowane, takie jak ruletka, turniej czy ranking.

4. Krzyżowanie (Crossover): Wybrane pary osobników poddawane są procesowi krzyżowania. To proces, w którym geny dwóch rodziców są wymieniane, tworząc potomstwo. To działa na zasadzie symulowania reprodukcji.

5. Mutacja: Z pewnym prawdopodobieństwem wprowadź zmiany w genotypie potomstwa. Mutacja pomaga wprowadzić losowe zmiany w populacji, które mogą prowadzić do odkrycia nowych obszarów przestrzeni rozwiązań.

6. Kryterium zakończenia: Sprawdź warunek zakończenia, na przykład maksymalna liczba generacji, osiągnięcie określonej wartości fitness, czy inne kryterium. Jeśli warunek jest spełniony, algorytm kończy działanie; w przeciwnym razie wraca do kroku 2.

Algorytm genetyczny jest iteracyjny i powtarza te kroki, stopniowo poprawiając jakość rozwiązania w każdej generacji. Dzięki kombinacji selekcji, krzyżowania i mutacji, algorytm stara się znaleźć najlepsze rozwiązanie w przestrzeni poszukiwań. Algorytm genetyczny jest skuteczny w rozwiązywaniu problemów optymalizacyjnych, zwłaszcza gdy przestrzeń poszukiwań jest duża i skomplikowana. Algorytm genetyczny jest podatny na różnorodne czynniki, które mogą wpływać na jego skuteczność i jakość uzyskiwanych rozwiązań.

1. Rozmiar populacji: Wielkość populacji odgrywa istotną rolę. Zbyt mała populacja może prowadzić do utraty różnorodności genetycznej, podczas gdy zbyt duża populacja może zwiększyć czas obliczeń.

2. Funkcja oceny (Fitness): Dokładność i adekwatność funkcji oceny mają kluczowe znaczenie. Wartość fitness powinna dokładnie odzwierciedlać jakość rozwiązania, aby algorytm skierował się w kierunku optymalizacji.

3. Prawdopodobieństwo krzyżowania i mutacji: Odpowiednie ustawienie prawdopodobieństwa krzyżowania i mutacji jest ważne. Zbyt niskie prawdopodobieństwo może sprawić, że algorytm będzie zbyt konserwatywny, podczas gdy zbyt wysokie może prowadzić do utraty stabilności.

4. Metoda selekcji: Wybór odpowiedniej metody selekcji, takiej jak ruletka, turniej czy ranking, ma wpływ na to, jakie osobniki zostaną wybrane do reprodukcji. Różne metody selekcji mogą prowadzić do różnych trajektorii ewolucyjnych.

5. Warunki zakończenia: Odpowiednio dobrany warunek zakończenia, na przykład maksymalna liczba generacji, osiągnięcie określonej wartości fitness lub brak poprawy przez pewien czas, ma wpływ na efektywność algorytmu.

6. Elitaryzm: Zachowanie pewnej liczby najlepszych osobników z każdej generacji (elitaryzm) może pomóc w utrzymaniu stabilności i uniknięciu utraty wartości najlepszych rozwiązań.

7. Reprezentacja rozwiązania: Sposób reprezentacji rozwiązania (genotyp) może wpłynąć na efektywność algorytmu. Odpowiednio dobrana reprezentacja powinna być dostosowana do struktury problemu.

8. Lokalne optimum: Algorytm genetyczny może mieć trudności w unikaniu lokalnych optimum. Wprowadzenie odpowiednich mechanizmów, takich jak mutacja, może pomóc w zbadaniu różnych obszarów przestrzeni rozwiązań.

9. Schemat krzyżowania: Wybór odpowiedniego schematu krzyżowania ma znaczenie. Różne schematy mogą być bardziej lub mniej odpowiednie dla konkretnego problemu.

W tej implementacji algorytm genetyczny obejmuje następujące operacje: sukcesję, krzyżowanie (crossover), mutację oraz selekcję.

1. Sukcesja: W kodzie sukcesja jest zrealizowana poprzez wybór najlepszych osobników i skopiowanie ich do nowej populacji. Sukcesja odbywa się w głównej pętli algorytmu genetycznego. Po zakończeniu jednej generacji, populacja jest sortowana według długości tras (od najlepszych do najgorszych). Następnie, część najlepszych osobników (określona jako 20% populacji) jest kopiowana do nowej populacji.

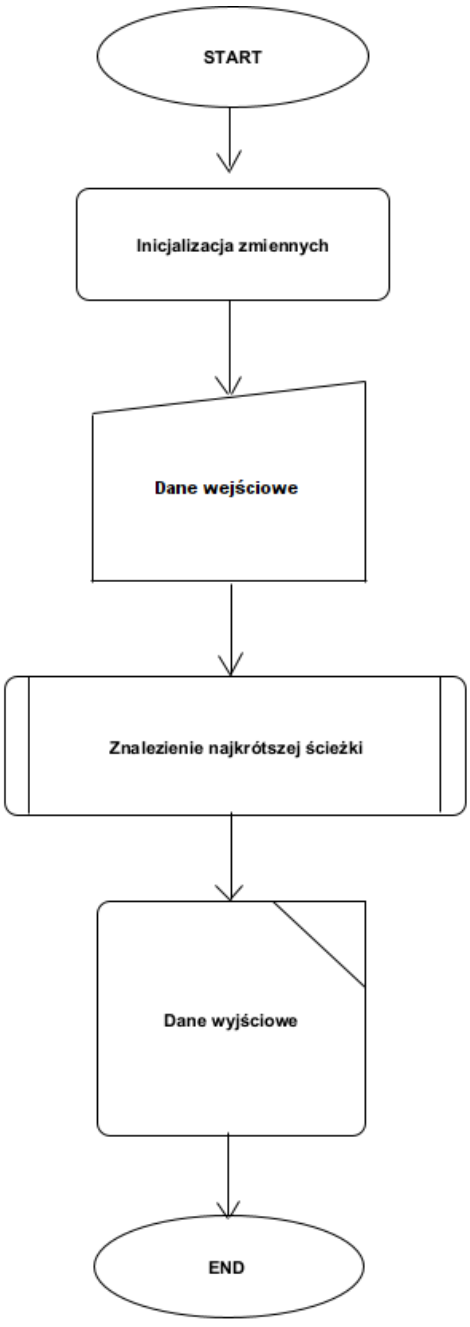
2. Krzyżowanie (Crossover): W kodzie krzyżowanie jest zrealizowane za pomocą operacji OX (Order Crossover). OXCrossing to funkcja, która przeprowadza krzyżowanie między dwoma rodzicami, tworząc dwójkę potomków. Funkcja OXCrossing losuje dwa punkty w trasie rodziców. Następnie kopiowany jest fragment trasy pomiędzy tymi dwoma punktami z jednego rodzica do jednego dziecka, a odwrotnie dla drugiego dziecka. Reszta trasy dziecka jest uzupełniana zgodnie z regułami, unikając powtórzeń. Krzyżowanie jest stosowane z określonym prawdopodobieństwem, które jest kontrolowane przez parametr reproductionProbability.

3. Mutacja: W kodzie mutacja jest realizowana poprzez operację inwersji (invertowania) losowego fragmentu trasy. Funkcja Mutation losuje dwa różne wierzchołki w trasie i inwertuje kolejność między nimi. Mutacja jest stosowana z określonym prawdopodobieństwem, które jest kontrolowane przez parametr mutationProbability.

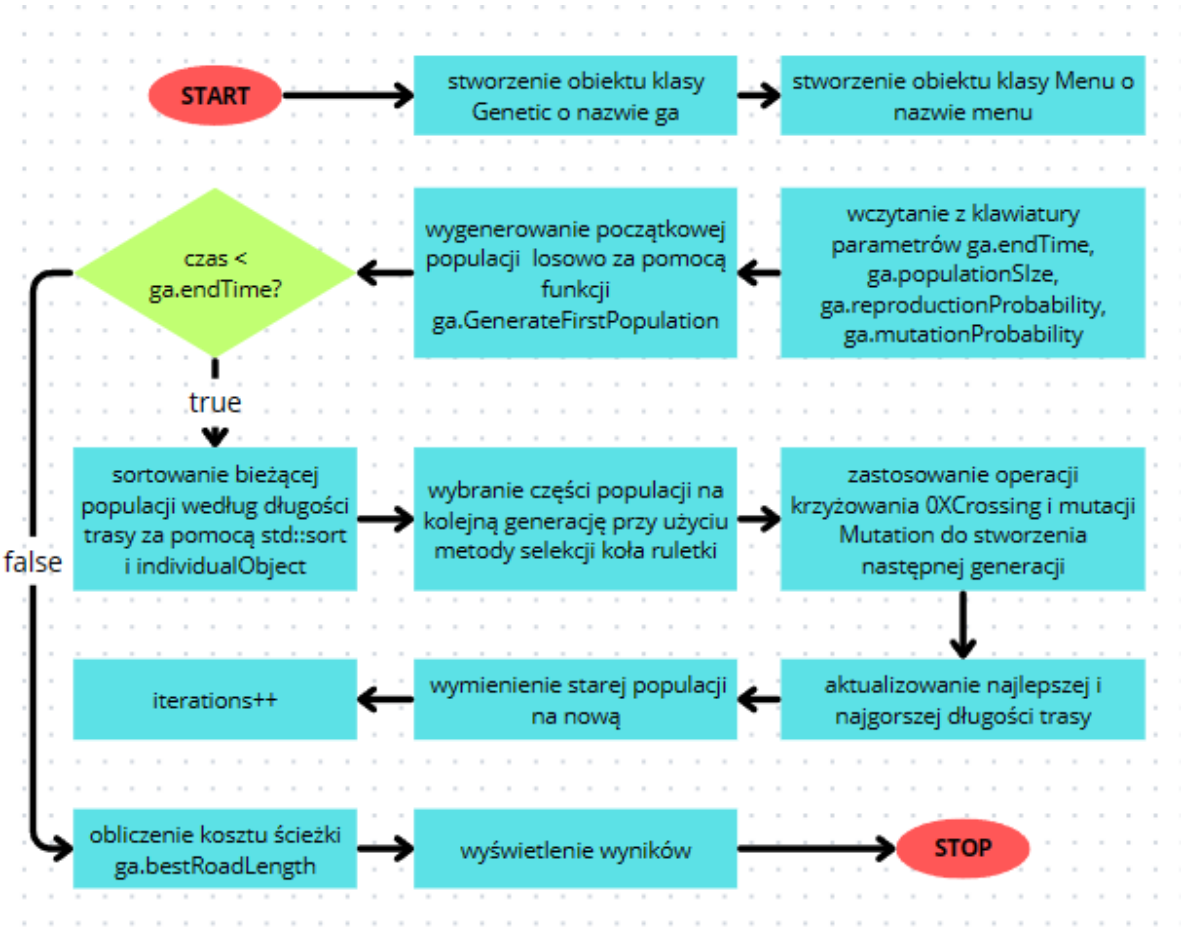
4. Selekcja: W kodzie selekcja jest realizowana za pomocą metody koła ruletki. Suma dostosowań (prawdopodobieństw) wszystkich osobników jest używana do losowego wyboru osobników do reprodukcji. Wartości dostosowania są używane do przypisania prawdopodobieństw reprodukcji, a następnie losuje się osobniki na podstawie tych prawdopodobieństw.

3 Algorytm

Rozwiązanie zaimplementowane w programie obrazują następujące schematy.



Rysunek 1: Schemat blokowy programu



Rysunek 2: Schemat blokowy algorytmu opartego na metodzie algorytmu genetycznego

Opis algorytmu:

1. Inicjalizacja: Tworzenie obiektu klasy Genetic o nazwie ga. Tworzenie obiektu klasy Menu o nazwie menu. Pobranie od użytkownika parametrów algorytmu genetycznego, takich jak czas wykonania (ga.endTime), rozmiar populacji (ga.populationSize), prawdopodobieństwo reprodukcji (ga.reproductionProbability) i prawdopodobieństwo mutacji (ga.mutationProbability).
2. Rozpoczęcie pętli wykonawczej przez określony czas (ga.endTime):
3. Generowanie początkowej populacji losowo za pomocą funkcji ga.GenerateFirstPopulation.: Iteracja przez liczbę osobników w populacji (ga.populationSize). Dla każdego osobnika, wywołanie funkcji ga.GenerateRandomRoad (Inicjalizuje nowego osobnika i tablicę informującą, które wierzchołki są już w trasie. Dopóki trasa nie zawiera wszystkich wierzchołków: Losuje wierzchołek, który jeszcze nie jest w trasie. Dodaje ten wierzchołek do trasy i aktualizuje tablicę informującą.) w celu wygenerowania losowej trasy.
4. Wejście do głównej pętli, w której są tworzone nowe generacje do momentu osiągnięcia określonego czasu wykonania.
   (a) Sortowanie bieżącej populacji według długości trasy za pomocą funkcji std::sort i individualObject.
   (b) Wybieranie części populacji na kolejną generację przy użyciu metody selekcji koła ruletki.
   (c) Stosowanie operacji krzyżowania OXCrossing(Losuje dwa punkty w trasie rodziców. Kopiuje fragment trasy pomiędzy tymi punktami z jednego rodzica do jednego dziecka i odwrotnie dla drugiego dziecka. Uzupełnia resztę trasy dziecka, unikając powtórzeń. Aktualizuje długość trasy dla obu dzieci.) i mutacji Mutation(Losuje dwa różne wierzchołki w trasie. Inwertuje kolejność wierzchołków między tymi dwoma punktami. Aktualizuje długość trasy po mutacji.), aby stworzyć następną generację.
   (d) Aktualizacja najlepszej i najgorszej długość trasy.
   (e) Zamiana starej populacji z nową.
   (f) Sprawdzanie warunku czasu - warunek zakończenia pętli.
5. Obliczenie najlepszej długości trasy ga.bestRoadLength.
6. Wyświetlenie wyników.

4 Dane testowe

Do sprawdzenia poprawności działania algorytmu i wykonania badań wybrano następujący zestaw instancji:
Dla problemu symetrycznego:
burma14.tsp 3323
gr21.tsp 2707
gr24.tsp 1272
att48.tsp 10628
gr96.tsp 55209
pr107.tsp 44303
gr202.tsp 40160
rbg323.tsp 1326
Dla problemu asymetrycznego:
p43.atsp 5620
ftv44.atsp 1613
ft70.atsp 38673
ftv170.atsp 2755
rbg358.atsp 1163
rbg443.atsp 2720
Instancje zostały pobrane ze zbioru TSPLIB[3].

5 Procedura badawcza

Zbadany został wpływ parametrów, takich jak prawdopodobieństwo mutacji, prawdopodobieństwo krzyżowania, wielkość populacji początkowej na jakość rozwiązania problemu. Obliczony został błąd względny rozwiązania według wzoru: $(|x - y|/x) * 100\%$, gdzie x to koszt optymalny, a y to koszt obliczony. Procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym *conf.ini*
(format pliku: *nazwa_instancji liczba_wykonań nazwa_pliku_wyjściowego*).
Każda z instancji rozwiązywana była 10 razy dla wyznaczonych najlepszych parametrów. Do pliku wyjściowego csv zapisywane były informacje o instancji: jej nazwa, liczba wykonań algorytmu, kryterium stopu w sekundach, rozmiar populacji początkowej, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji. Następnie zapisywane były czasy wykonań algorytmu oraz obliczony koszt dla każdego testu tej instancji. Plik wyjściowy zapisywany był w formacie csv. Poniżej przedstawiono zawartość przykładowego pliku wyjściowego dla instancji burma14.tsp przy 10 wywołaniach algorytmu.

burma14.tsp 10 20 7 0.3 0.1
20002;3323
20001;3323
20001;3323
20002;3323
20001;3323
20002;3323
20001;3323
20001;3323
20002;3323
20001;3323
testburma14.csv

Pomiary zostały wykonane na platformie sprzętowej:
procesor: Intel® Core™ i5-01400F CPU @ 2.90GHz × 6
pamięć operacyjna: 16 GiB
system operacyjny: Windows 11 64-bit

W programie czas jest mierzony za pomocą funkcji *read_QPC()*, która wykorzystuje API systemu Windows do odczytywania licznika wydajności procesora (QPC - QueryPerformanceCounter).

```
long long int read_QPC()
{
    LARGE_INTEGER count;
    QueryPerformanceCounter(&count);
    return((long long int)count.QuadPart);
}
```

Opis działania:

1. *QueryPerformanceCounter(&count)* pobiera aktualną wartość licznika wydajności procesora i zapisuje ją w strukturze *count*.
2. *count.QuadPart* zawiera odczytaną wartość, która jest zwracana jako long long int.
3. W funkcji *choose_ option()* jest wykorzystywana funkcja *QueryPerformanceFrequency*, która zwraca liczbę tików licznika wydajności na sekundę. Jest to wykorzystywane do przekształcenia różnicy czasu na milisekundy:

```
long long int frequency, start, elapsed;
QueryPerformanceFrequency((LARGE_INTEGER *)&frequency);
```

Cały proces mierzenia czasu wygląda więc następująco:

1. Pobierana jest częstotliwość licznika wydajności za pomocą *QueryPerformanceFrequency* i zapisywana w zmiennej *frequency*.
2. Przed wykonaniem operacji, której czas chcemy zbadać, wykonujemy *start = read_ QPC()* aby zapisać aktualną wartość licznika wydajności.
3. Po wykonaniu operacji, ponownie wywołujemy *read_ QPC()* i odejmujemy *start* od odczytanej wartości, aby uzyskać czas, jaki upłynął.
4. Następnie przekształcamy różnicę czasu na milisekundy - używając (1000.0 * elapsed) / frequency).

Taki sposób mierzenia czasu opiera się na tym, że licznik wydajności procesora działa z bardzo wysoką precyzją, co pozwala na dokładne mierzenie czasu wykonania operacji.

Po każdym powtórzeniu wykonania algorytmu dla danej instancji, program informuje o znalezionej ścieżce oraz jej koszcie, jak również czasie działania programu. Wyniki zostały opracowane w programie Microsoft Excel.

6 Wyniki

6.1 Dla problemu symetrycznego:

Tabele przedstawiają generowany błąd w zależności od prawdopodobieństwa krzyżowania dla różnych wielkości populacji początkowych n, n/2, n/4 przy dobranym prawdopodobieństwie mutacji.

Rysunek 3: Tabela wyników dla n=14

Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/2	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/4	błąd (%)	0	0	0	0	0	0	0	0	0	0

Rysunek 4: Tabela wyników dla n=21

Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/2	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/4	błąd (%)	0	0	0	0	0	0	0	0	0	0

Rysunek 5: Tabela wyników dla n=48

Prawdopodobieństwo mutacji: 0,8											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	2	2	3	4	4	3	4	3	1	4
n/2	błąd (%)	4	6	3	2	2	2	4	4	1	1
n/4	błąd (%)	2	3	1	2	4	0	2	1	2	5

Rysunek 6: Tabela wyników dla n=96

Prawdopodobieństwo mutacji: 0,2											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	25	22	21	22	24	14	27	17	26	28
n/2	błąd (%)	23	29	13	22	17	13	17	10	18	14
n/4	błąd (%)	9	13	11	13	15	8	12	9	15	14

Rysunek 7: Tabela wyników dla n=107

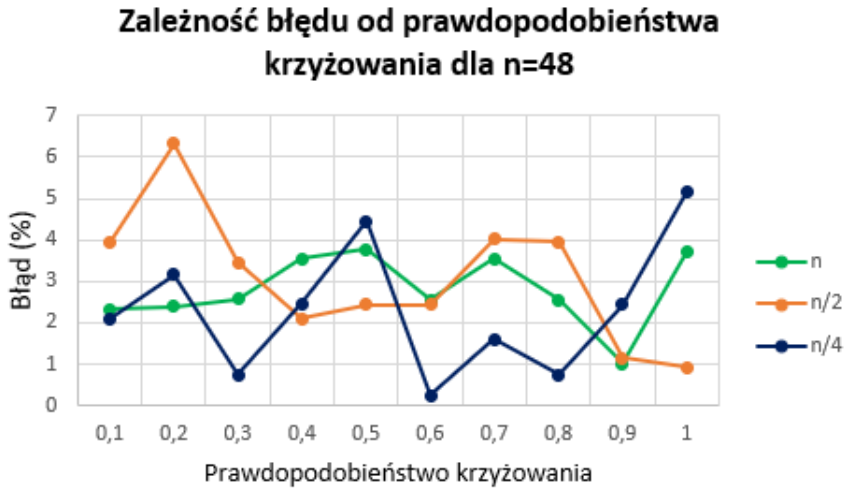
Prawdopodobieństwo mutacji: 0,2											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	9	16	12	20	12	12	10	19	16	13
n/2	błąd (%)	3	5	19	7	14	11	1	7	6	13
n/4	błąd (%)	10	10	7	11	4	11	7	10	11	8

Rysunek 8: Tabela wyników dla n=202

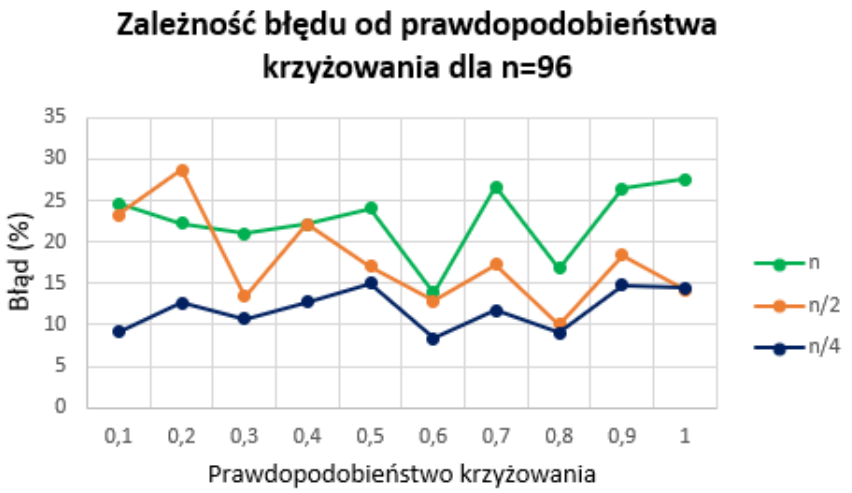
Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	88	76	89	79	87	80	84	88	73	88
n/2	błąd (%)	60	68	66	67	69	70	55	69	65	54
n/4	błąd (%)	48	41	38	33	44	47	45	39	43	42

Rysunek 9: Tabela wyników dla n=323

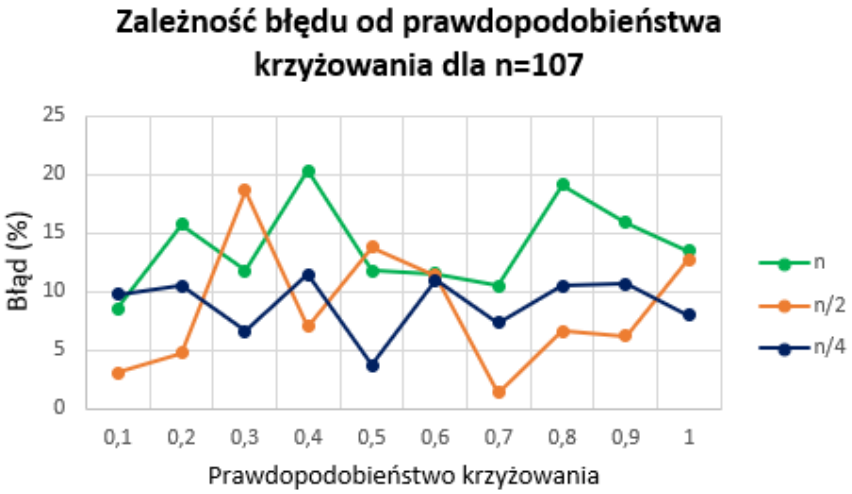
Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	108	107	113	103	127	108	114	107	122	113
n/2	błąd (%)	102	99	101	100	94	102	97	106	97	104
n/4	błąd (%)	94	98	89	94	90	86	85	88	83	94



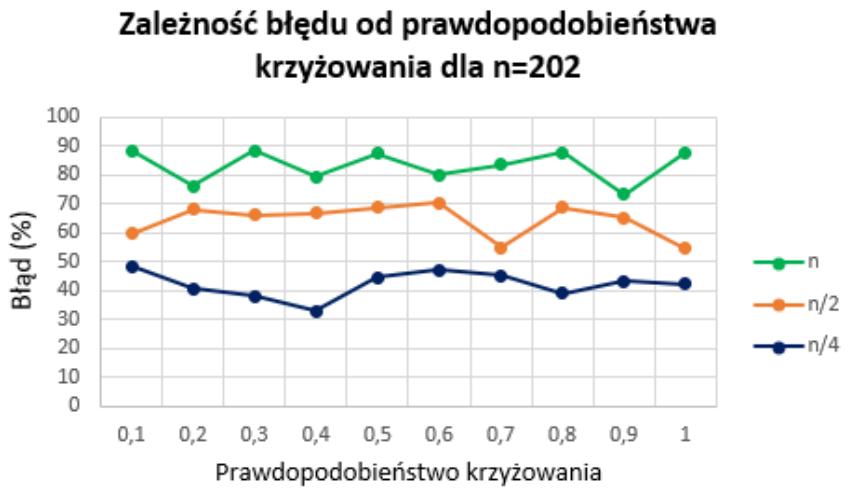
Rysunek 10: Zależność błędu od prawdopodobieństwa krzyżowania dla n=48



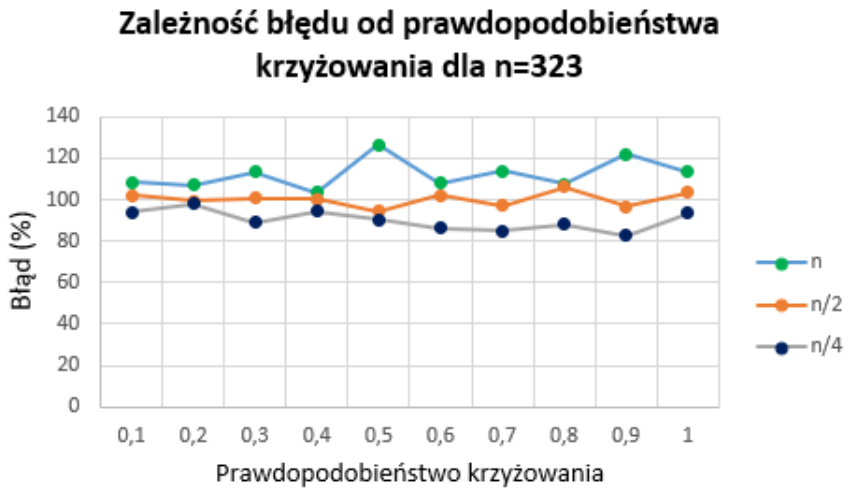
Rysunek 11: Zależność błędu od prawdopodobieństwa krzyżowania dla n=96



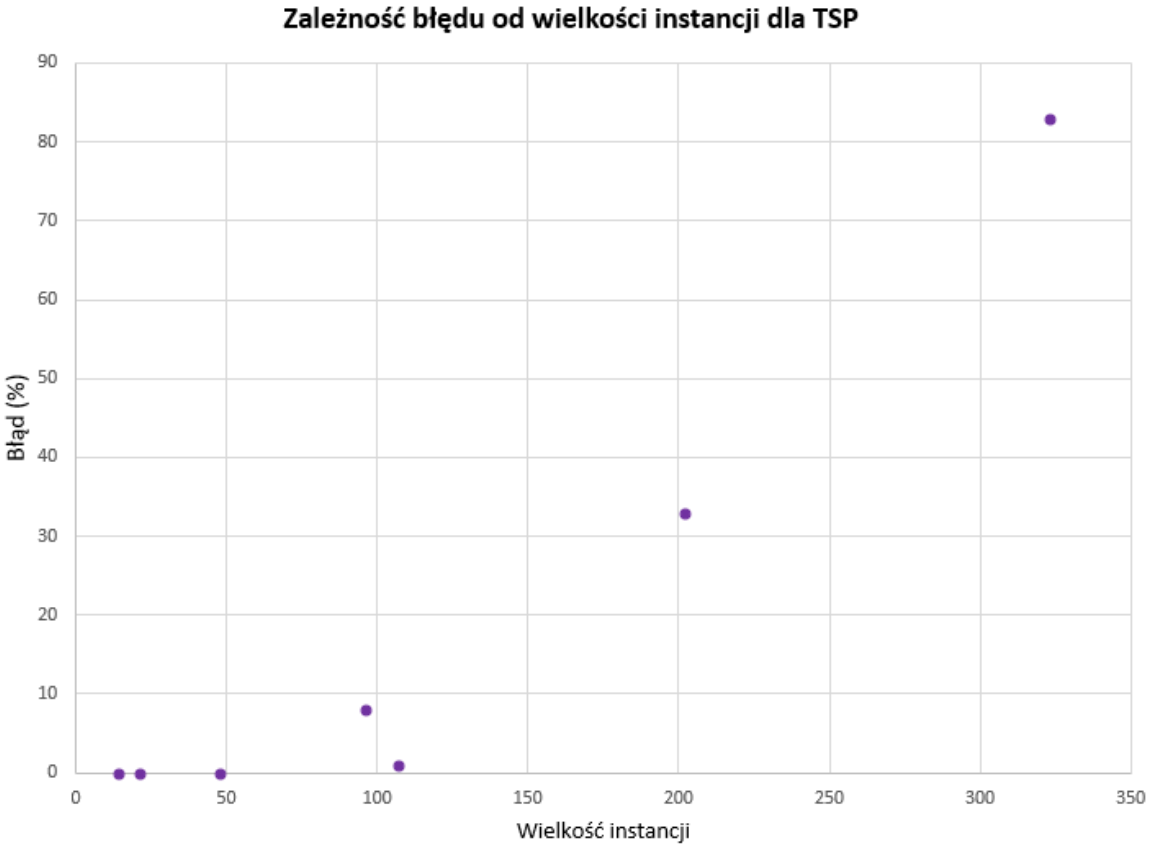
Rysunek 12: Zależność błędu od prawdopodobieństwa krzyżowania dla n=107



Rysunek 13: Zależność błędu od prawdopodobieństwa krzyżowania dla n=202



Rysunek 14: Zależność błędu od prawdopodobieństwa krzyżowania dla n=323



Rysunek 15: Zależność błędu od wielkości instancji

6.2 Dla problemu asymetrycznego:

Tabele przedstawiają generowany błąd w zależności od prawdopodobieństwa krzyżowania dla różnych wielkości populacji początkowych n, n/2, n/4 przy dobranym prawdopodobieństwie mutacji.

Rysunek 16: Tabela wyników dla n=43

Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/2	błąd (%)	0	0	0	0	0	0	0	0	0	0
n/4	błąd (%)	0	0	0	0	0	0	0	0	0	0

Rysunek 17: Tabela wyników dla n=45

Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	1	1	1	1	1	1	1	1	1	1
n/2	błąd (%)	0	1	1	1	1	1	1	1	1	1
n/4	błąd (%)	1	0	1	0	1	1	1	1	1	1

Rysunek 18: Tabela wyników dla n=70

Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	14	14	18	15	13	15	18	15	18	16
n/2	błąd (%)	10	9	7	6	6	8	8	7	8	8
n/4	błąd (%)	5	2	5	4	6	5	6	6	9	4

Rysunek 19: Tabela wyników dla n=171

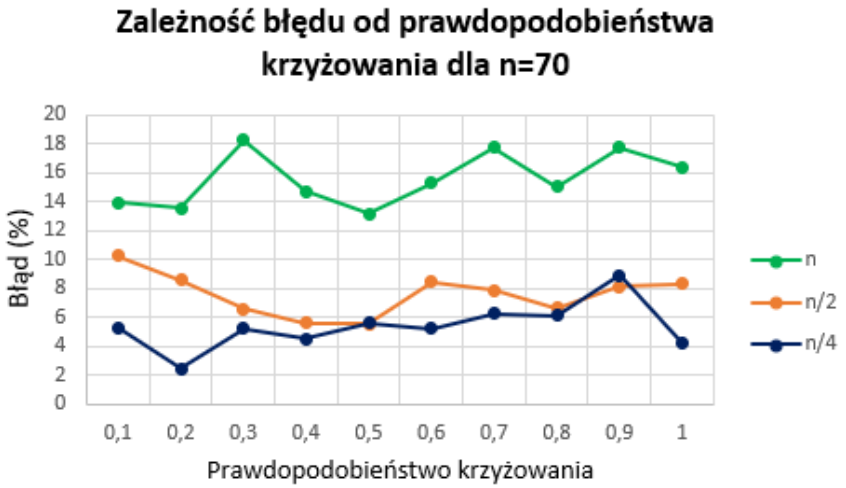
Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	193	178	185	180	167	173	165	170	168	161
n/2	błąd (%)	191	218	217	225	231	203	233	235	239	228
n/4	błąd (%)	114	115	126	116	136	122	110	106	144	99

Rysunek 20: Tabela wyników dla n=358

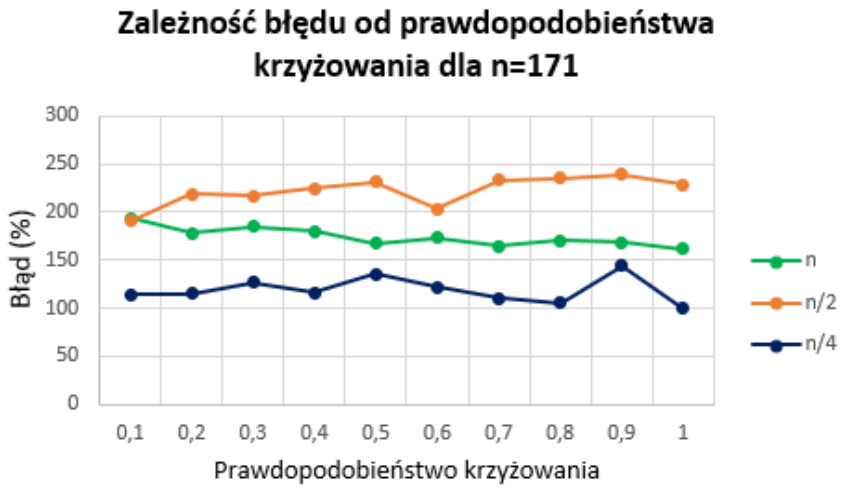
Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	176	168	173	175	178	184	176	175	173	175
n/2	błąd (%)	160	161	158	147	151	156	161	160	150	147
n/4	błąd (%)	133	122	138	138	139	140	142	147	139	136

Rysunek 21: Tabela wyników dla n=443

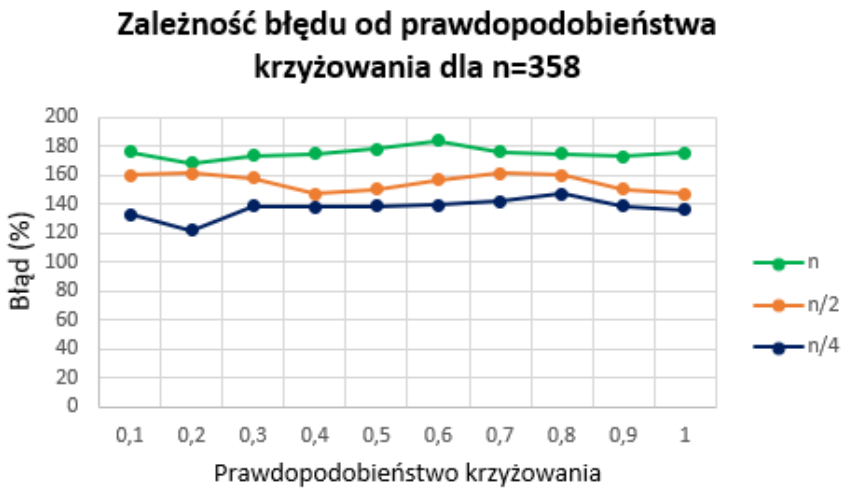
Prawdopodobieństwo mutacji: 0,1											
	pk	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
n	błąd (%)	77	79	82	83	83	82	83	81	82	81
n/2	błąd (%)	69	69	68	71	70	72	72	71	69	67
n/4	błąd (%)	62	62	70	56	68	67	66	68	67	67



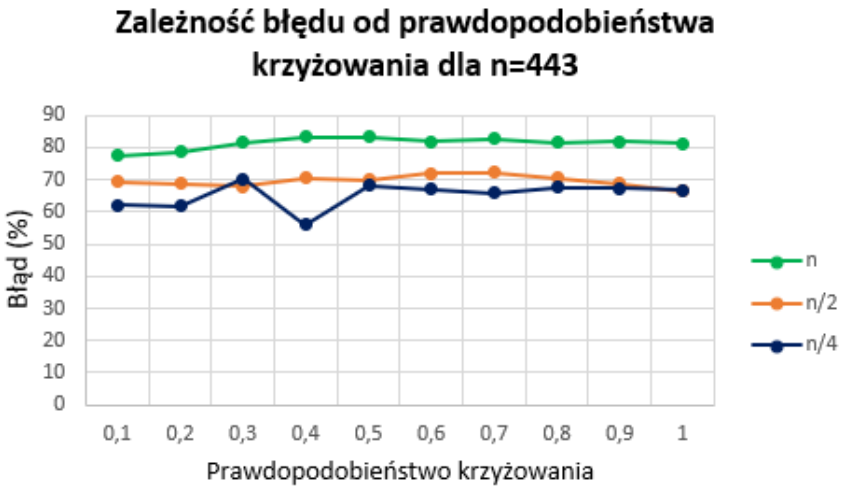
Rysunek 22: Zależność błędu od prawdopodobieństwa krzyżowania dla n=70



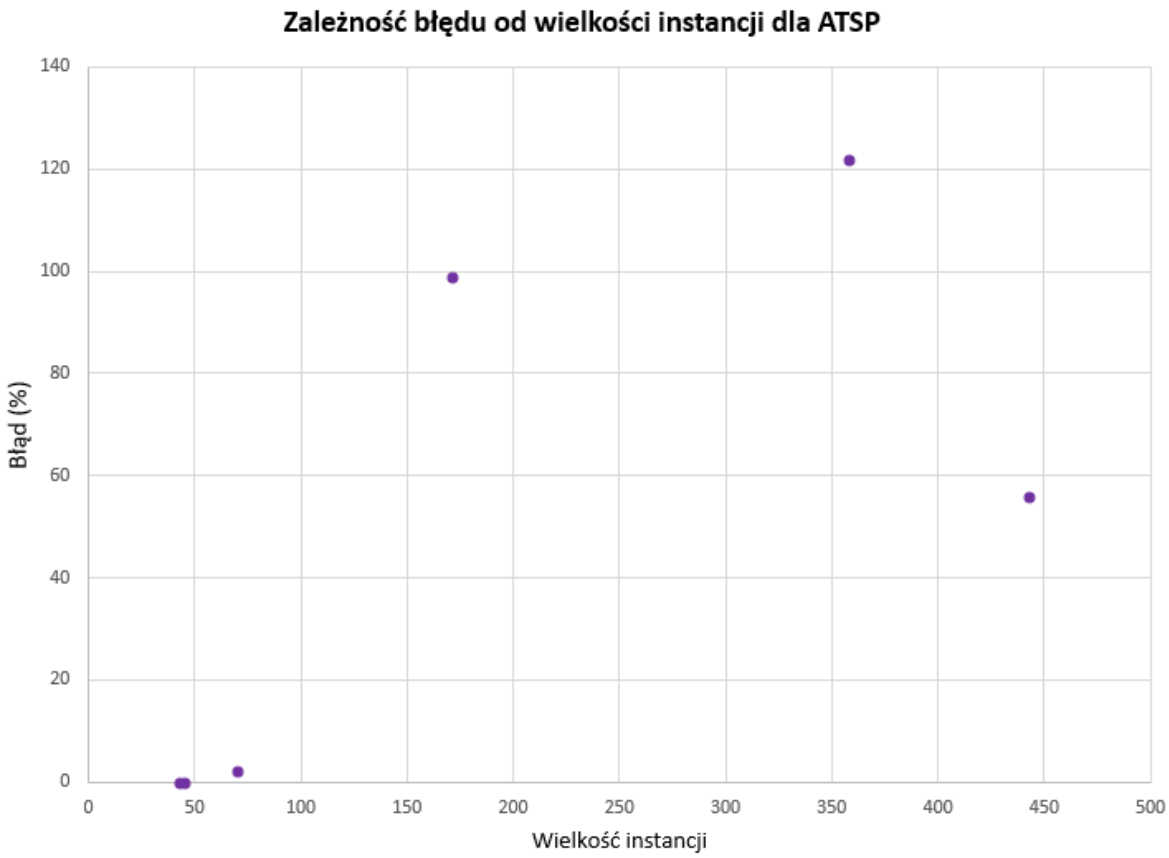
Rysunek 23: Zależność błędu od prawdopodobieństwa krzyżowania dla n=171



Rysunek 24: Zależność błędu od prawdopodobieństwa krzyżowania dla n=358



Rysunek 25: Zależność błędu od prawdopodobieństwa krzyżowania dla n=443

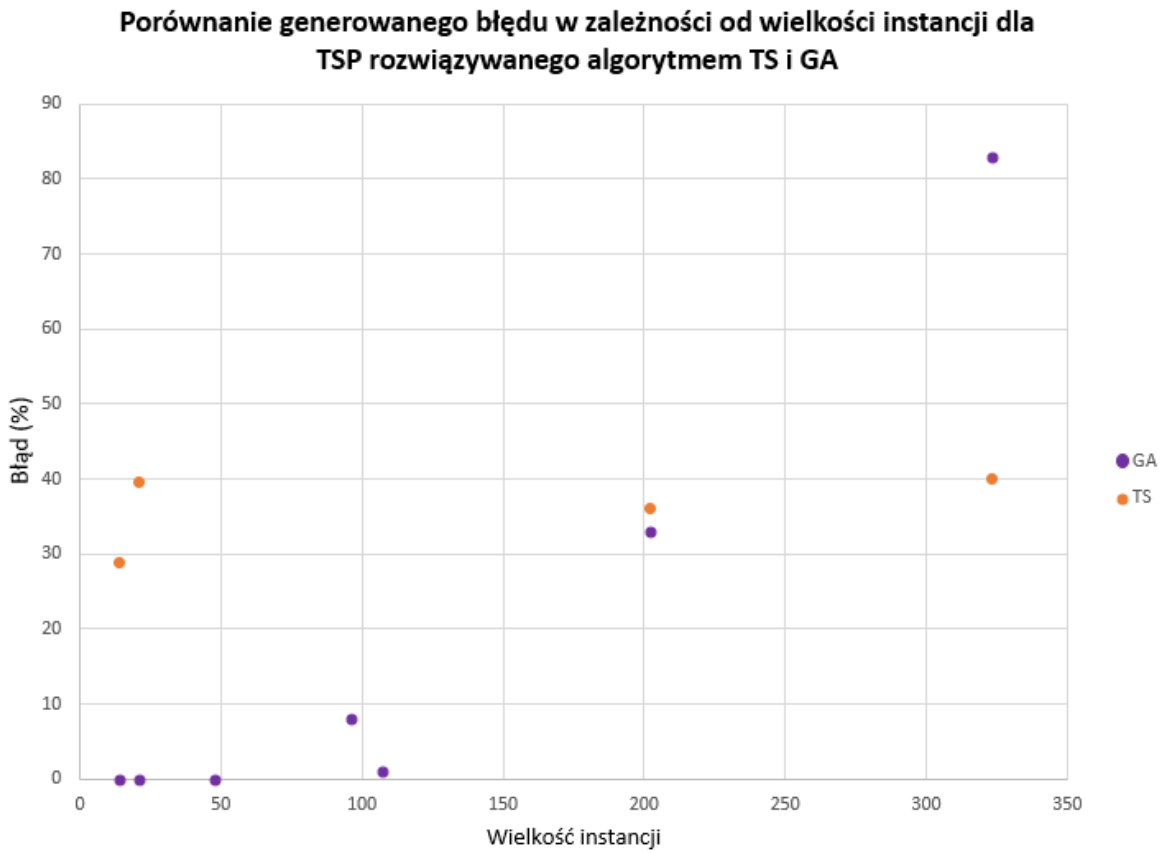


Rysunek 26: Zależność błędu od wielkości instancji

6.3 Porównanie z Tabu Search

Wielkość instancji (n)		Tabu Search	Algorytm genetyczny	Błąd(%)
	14	29	0	
	21	40	0	
	202	36	33	
	323	40	83	

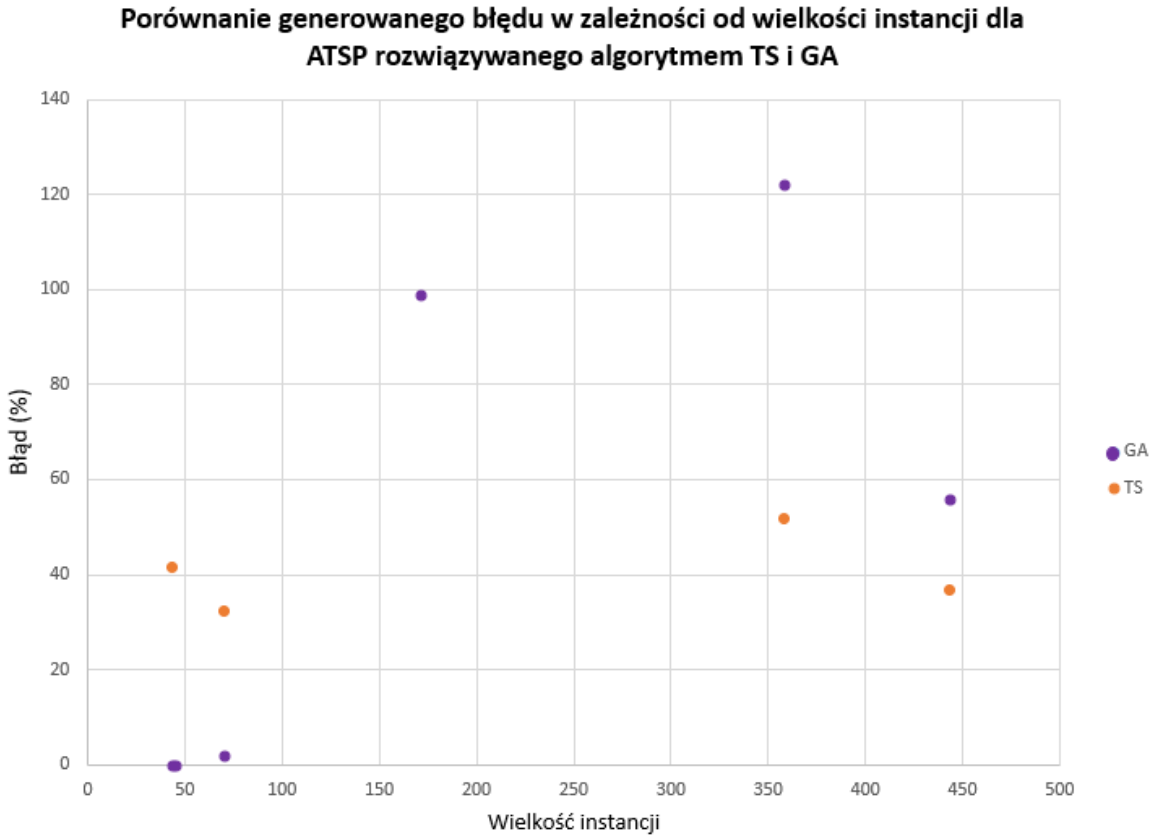
Rysunek 27: Tabela błędów generowanych w algorytmie Tabu Search a algorytmie genetycznym dla TSP



Rysunek 28: Porównanie generowanego błędu w algorytmie Tabu Search a algorytmie genetycznym dla TSP

Wielkość instancji (n)		Tabu Search	Algorytm genetyczny	Błąd(%)
	43	42	0	
	70	32	2	
	358	52	122	
	443	37	56	

Rysunek 29: Tabela błędów generowanych w algorytmie Tabu Search a algorytmie genetycznym dla ATSP



Rysunek 30: Porównanie generowanego błędu w algorytmie Tabu Search a algorytmie genetycznym dla ATSP

7 Analiza wyników i wnioski

Procent błędu rozwiązania problemu algorytmem genetycznym rośnie wraz z wielkością instancji. Dla instancji do n=70 są to błędy rzędu do 2%. Dodatkowo warto dodać, że dla tych instancji błąd nie zmieniał się znacznie w zależności od dobranych parametrów. Natomiast dla instancji o n $>$ 70 błąd różni się w zależności od instancji. Dla n=107 TSP udało się uzyskać błąd minimalny wynoszący 1%. Natomiast w przypadku problemu asymetrycznego błąd dla n=358 wyniósł aż 122%. Spowodowane jest to najprawdopodobniej nienajlepiej dobranymi parametrami, ponieważ dla n=443 błąd spadł aż do 56%. Dla wszystkich instancji przed przeprowadzeniem właściwych testów zostały dobrane eksperymentalnie najlepsze parametry. W przypadku dużej części instancji najlepiej funkcjonującym parametrem prawdopodobieństwa mutacji okazało się 0,1. Dla TSP dobrym parametrem prawdopodobieństwa krzyżowania często okazywały się wartości z przedziału 0,6-0,9, natomiast w przypadku ATSP z przedziału 0,1-0,3. Dla instancji n=171, n=323, n=358 nie udało się dobrać wystarczająco dobrych parametrów i otrzymany procent generowanego błędu nie jest na poziomie zadowalającym. W przypadku instancji n=171 jest to najprawdopodobniej spowodowane charakterystyką samej instancji, która sprawiała problemy również w poprzednich testach dla innych algorytmów.

Zarówno dla problemu symetrycznego jak i asymetrycznego można zauważyć, że wraz z wielkością instancji rośnie także procent błędu. Dla problemu asymetrycznego należy odrzucić odbiegające wyniki dla n=171, n=358. Można zauważyć, że większość instancji osiąga mniejszy procent błędu dla populacji początkowej równej n/4. Jedynym odstępstwem od tej reguły jest n=107, gdyż dla tej instancji najlepszy wynik został osiągnięty dla populacji początkowej równej n/2. Analizując wykresy zależności błędu od prawdopodobieństwa krzyżowania i wielkości populacji początkowej sporządzone dla każdej z instancji można zauważyć tendencyjny spadek procentu błędu wraz ze zmniejszaniem wielkości populacji początkowej. Można zauważyć, że dla instancji do n=107 spadek procentu błędu dzięki dobraniu właściwych parametrów był niewielki, ponieważ błąd dla tych instancji nie był duży, stąd nie potrzeba było dużej poprawy. Natomiast w przypadku instancji większych dzięki dobieraniu właściwych parametrów udawało się zmniejszyć procent błędu nawet o 140%, co doskonale pokazuje wpływ właściwego dostrojenia algorytmu na jakość otrzymanego rozwiązania. Warto także zwrócić uwagę na fakt, że dobrane parametry algorytmu, które były odpowiednie dla konkretnej instancji, w innych przypadkach nie sprawdzały się równie dobrze. Dla problemu symetrycznego badany procent błędu był mniejszy niż dla problemu asymetrycznego, zwłaszcza w przypadku dużych instancji. Nie można jednak porównać dokładnie zbadanego problemu symetrycznego z asymetrycznym, ponieważ dla każdego z nich badane były inne zestawy danych.

Dalsza optymalizacja parametrów algorytmu może pomóc w osiągnięciu jeszcze lepszych wyników dla różnych instancji problemu. Eksperymenty z różnymi kombinacjami parametrów, takimi jak prawdopodobieństwo mutacji, prawdopodobieństwo krzyżowania, rozmiar populacji początkowej czy warunek czasowy wykonywania algorytmu, naj-

prawdopodobniej pozwoliłyby na zmiejszenie generowanego procentu błędu. W celu bardzo dokładnego zweryfikowania skuteczności algorytmu można także przeprowadzić ewaluację na różnorodnych przypadkach zestawów danych.

Algorytm genetyczny dla problemu komiwojażera osiąga zdecydowanie mniejszy procent błędu niż algorytm Tabu Search. Na wykresie nr 28 i 30 widać to bardzo wyraźnie zwłaszcza dla instancji n<300. Dla większych instancji nie jest to aż tak wyraźnie widoczne ze względu na nie do końca właściwe dobranie parametrów dla n=323 i n=358. W przypadku algorytmu genetycznego możliwe jest również badanie zdecydowanie większych instancji niż dla Tabu Search, Branch and Bound czy Brute force, ponieważ algorytm genetyczny wypada zdecydowanie lepiej pod względem czasowym, gdyż czas wykonania zależy od ustalonego przez nas warunku stopu. Przy właściwym dostrojeniu parametrów można osiągnąć niskie procenty błędu w satysfakcjonującym czasie, co w przypadku Brute force i Branch and Bound nie byłoby możliwe, a w przypadku Tabu Search zajęłoby zdecydowanie więcej czasu. Porównując więc te cztery algorytmy, zdecydowanie najlepiej pod wględem jakości rozwiązania i jednocześnie satysfakcjonującego czasu wykonania prezentuje się algorytm genetyczny.

Źródła:

[1] https://pl.wikipedia.org/wiki/Graf_pe%C5%82ny
[2] https://pl.wikipedia.org/wiki/Cykl_Hamiltona
[3] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Spis rysunków