Kajetan Parzyszek

Projektowanie Efektywnych Algorytmów

Zadanie projektowe nr 3: Symetryczny i Asymetryczny Problem Komiwojażera. Populacyjne algorytmy metaheurystyczne. Algorytm Genetyczny.

K P
1-21-2019

Spis treści

1. Wstęp teoretyczny

Celem przeprowadzonego eksperymentu było zbadanie efektywności algorytmu *populacyjno metaheurystycznego* dla *symetrycznego* i *asymetrycznego problemu komiwojażera*(dalej nazywane odpowiednio *STSP* i *ATSP)*. Aby tego dokonać najpierw trzeba było zaimplementować odpowiednią strukturę przechowującą listę miast, wybrana została *macierz sąsiedztwa*, oraz algorytm, a mianowicie, *algorytm Genetyczny*. Język programowania jaki został użyty przy programowaniu struktur, algorytmów jak i przy przeprowadzaniu testów to C++.

Badaną zależnością był wpływ doboru parametrów na osiągany wynik przy założonej stałej liczbie iteracji dla każdego z problemów. Drugą z badanych wartości była ilość iteracji jaka potrzebna jest do uzyskania jak najlepszego wyniku przy wykorzystaniu optymalnych parametrów otrzymanych w pierwszym eksperymencie.

1.1. Symetryczny i Asymetryczny Problem Komiwojażera

Problem komiwojażera jest to zagadnienie optymalizacyjne skupiające się na znalezieniu minimalnego cyklu Hamiltona(czyli cyklu w którym każdy wierzchołek, oprócz pierwszego, w grafie odwiedzany jest dokładnie raz) w pełnym grafie ważonym(czyli grafie który posiada wszystkie możliwe krawędzie pomiędzy wierzchołkami, a krawędzie mają przyporządkowany koszt przejścia krawędzi).

Nazwa problemu podchodzi od jego typowej ilustracji, jako sprzedawcy(komiwojażer), który ma odwiedzić konkretną liczbę miast i wrócić do punktu wyjścia. Znane są odległości między miastami, a celem jest znalezienie drogi o najmniejszym koszcie(czy to czasowym, kosztowym, odległościowym).

Problem Komiwojażera dzieli się na symetryczny i asymetryczny, w problemie symetrycznym koszt ścieżki między dwoma miastami A i B jest taki sam dla podróży z A -> B jak i B -> A, natomiast dla problemu asymetrycznego koszt ścieżki z A -> B może być różny od kosztu ścieżki z B ->A.

Największą trudnością problemu komiwojażera jest ilość danych wymagających analizy. Dla *n* miast liczba możliwych kombinacji wynosi $\frac{(n-1)!}{2}$.

1.2. Algorytm Genetyczny

Algorytm *Genetyczny*, rodzaj *heurystyki,* należy do rodziny algorytmów *ewolucyjnych.* Metoda ta polega na analizie alternatywnych rozwiązań dla obecnie posiadanego w celu wyszukania rozwiązań najlepszych. Twórca *John Henry Holland*, zaczerpnął inspirację z dziedziny biologii, stąd też nazwa algorytmu, która odnosi się do biologicznej ewolucji.

Podstawowe słownictwo związane z tym algorytmem wiele wspólnego ma z nazewnictwem biologicznym, jak: pula rozwiązań w obecnej iteracji to *populacja*, każdy osobnik z populacji ma przypisany *genotyp*, który z kolei składa się z *chromosomów,* które to są głównym elementem naszego osobnika, to one przechowują nasze rozwiązanie. *Chromosomy* składają się z pojedynczych *genów*.

Mając tak zdefiniowane słownictwo, czyli definicje podstawowych obiektów na których pracuje algorytm genetyczny możemy przejść do definicji operacji, które będziemy wykonywać na naszej populacji w celu otrzymania rozwiązania. Celem każdej iteracji algorytmu jest otrzymanie nowej, w założeniu dającej lepsze rozwiązanie problemu, populacji, w tym celu należy dokonać selekcji osobników, metod jest wiele, wybrana została jedna, a mianowicie *metoda koła ruletki*. W tej metodzie każdego osobnika oceniamy na podstawie rozwiązania jakie przechowuje i im jest bliżej rozwiązania tym większy % szans na bycie wylosowanym zostaje mu przypisany. Na podstawie procentowych szans losujemy osobniki które będą tworzyły nowe pokolenie, o takim samym rozmiarze jak poprzednie. Mając wylosowanych osobników tworzących następne pokolenie przechodzimy do głównego sposobu otrzymania nowych wyników a mianowicie operatorów *krzyżowania* oraz *mutacji*. Założono, że pierwsze występuję krzyżowanie, dla każdej pary osobników, na podstawie parametru procentowej szansy zajścia krzyżowania wyliczane jest czy operator ma zadziałać czy nie, jeżeli nie to nie dzieję się nic, jeżeli tak to dane dwa osobniki są ze sobą krzyżowane przy użyciu konkretnej metody, ponownie, jest ich wiele, wybrana została jedna, a mianowicie *Partially-mapped Crossover(PMX)*. Wybierany jest losowo punkt krzyżowania, jeden lub wiele, od początku chromosomu do punktu krzyżowania zamienia się ze sobą geny jednego osobnika z drugim, a następnie brakujące geny ustawiane są za punktem krzyżowania w takiej kolejności w jakiej występowały w osobniku potomnym. Następnym działającym operatorem jest operator *mutacji*, działa on również w zależności od prawdopodobieństwa zajścia mutacji. Operator mutacji służy dywersyfikacji rozwiązań, zamienia on ze sobą dwa losowo wybrane geny w danym osobniku.

Algorytm ten jest stosunkowo prosty w implementacji ze względu na opieranie się na założeniach z życia wziętych, jednakże nie gwarantuje znalezienia optymalnego rozwiązania, a w swojej najprostszej postaci generuje znaczny % błędu dla większych instancji. Czas wykonania algorytmu zależny jest przede wszystkim od ustalonej liczby iteracji jakie ma wykonać, zadanej wielkości populacji, rozmiaru instancji oraz częściowo od zadanych parametrów algorytmu. Teoretyczną złożoność określić możemy jako O(i*p*n), gdzie i to liczba iteracji głównej pętli, p to wielkość populacji dla której przeprowadzane są operacje oraz wyliczana jest najkrótsza droga, n to rozmiar instancji. Najbardziej czasochłonnym zadaniem algorytmu generacji jest właśnie wyliczanie oceny dla każdego osobnika

Schemat działania algorytmu przedstawia się następująco:

1. Generujemy losowe chromosomy dla każdego osobnika z populacji
2. Każdego z osobników oceniamy
3. Z obecnej populacji wybieramy osobników którzy utworzą nową populację, w tym celu korzystamy z wybranej funkcji selekcji
4. Dla każdej pary osobników sprawdzamy czy zachodzi krzyżowanie, zależnie od prawdopodobieństwa, jeżeli tak krzyżujemy ze sobą osobników przy pomocy wybranej funkcji krzyżowania
5. Dla każdego osobnika sprawdzamy czy zachodzi mutacja, zależnie od prawdopodobieństwa, jeżeli tak mutujemy danego osobnika przy pomocy wybranej funkcji mutacji
6. Powtarzamy kroki 2-5 dopóki nie wykonamy zadanej liczby iteracji

Parametrami algorytmu jakie możemy modyfikować są, *liczba iteracji*, ile razy główna pętla programu się wykona, *wielkość populacji*, decydująca o ilości osobników w populacji, *prawdopodobieństwo mutacji oraz krzyżowania*, decydujące o prawdopodobieństwie zadziałania odpowiednio operatora mutacji oraz krzyżowania. Dodatkowymi parametrami mógłby być wybór *funkcji selekcji, mutacji czy też krzyżowania.*

2. Plan eksperymentu

- Językiem programowania użytym w eksperymencie był C++ w standardzie ISO C++11
- Badanie poszczególnych algorytmów wykonane zostało dla *17, 47, 70, 170 miast w przypadku ATSP oraz dla 17, 48, 96, 136 miast w przypadku STSP*
- Dane na temat odległości między miastami jak i optymalne rozwiązania dla wybranych danych testowych zaczerpnięte zostały z *TSPLIB*
- Miasta i odległości między nimi były przechowywane w pamięci komputera z wykorzystaniem *macierzy sąsiedztwa*
- Dane w plikach z testowymi instancjami były przechowywane w formie *grafu pełnego* zapisanym w postaci *macierzy sąsiedztwa* z pierwszą wartością w pliku określającą rozmiar macierzy
- Dla każdej instancji i zestawu parametrów pomiar został powtórzony dwudziestokrotnie
- Otrzymanym wynikiem jest % wartość błędu stosunku otrzymanego do optymalnego rozwiązania problemu zależnie od zestawu parametrów oraz zależnie od liczby iteracji dla testowanego optymalnego zestawu parametrów
- Dla algorytmu *Genetycznego* możliwe zestawy parametrów to:
  - Rozmiar populacji: {10, 30, 50, 70}
  - Prawdopodobieństwo krzyżowania: {0.5, 0.6, 0.7, 0.8, 0.9, 1}
  - Prawdopodobieństwo mutacji: {0.01, 0.05, 0.1, 0.15}
- Dla każdej instancji problemu na podstawie pięciu zestawów parametrów dających najmniejszy % błędu wyliczono średnio ważoną (wagi: 1. Miejsce 5, 2. Miejsce 4, … 5. Miejsce 1) dla każdego z parametrów, którą następnie wykorzystano w pomiarach % błędu zależnego od liczby iteracji
- Wartości losowe generowane były przy użyciu biblioteki *random*, przy wyborze miast oraz punkcie krzyżowania korzystano z *uniform_int_distribution,* a przy testowaniu występienia krzyżowania oraz mutacji z *uniform_real_distribution*
- Początkowa populacja generowana była poprzez wybór kolejnych losowych miast z puli możliwych

3. Wyniki/Wnioski

3.1.Optymalne parametry alogrytmu genetycznego

W poniższych tabelach zawarto po pięć zestawów parametrów, które dawały najmniejszy % błędu dla konkretnych instancji problemu. Pełne wyniki dostępne w repozytorium projektu.

STSP - gr17				STSP - gr48			
Populacja	**Mutacja%**	**Krzyzowanie%**	**Błąd %**	**Populacja**	**Mutacja%**	**Krzyzowanie%**	**Błąd %**
70	0,15	0,90	2,5	70	0,15	1,00	38,8
70	0,15	1,00	3,2	70	0,15	0,60	39,6
70	0,15	0,50	3,4	70	0,15	0,90	41,8
50	0,15	1,00	4,0	50	0,15	0,60	44,3
70	0,15	0,80	4,3	70	0,10	1,00	45,6
67	**0,15**	**0,85**	**:Średnia ważona**	**67**	**0,15**	**0,82**	**:Średnia ważona**

Tabela 1 Wyniki najlepszych zestawów parametrów dla instacji gr17

Tabela 2 Wyniki najlepszych zestawów parametrów dla instancji gr48

STSP - gr96				STSP - pr136			
Populacja	**Mutacja%**	**Krzyzowanie%**	**Błąd %**	**Populacja**	**Mutacja%**	**Krzyzowanie%**	**Błąd %**
10	0,10	0,90	88,4	30	0,05	0,70	118,1
30	0,05	0,60	89,4	10	0,10	1,00	119,2
70	0,15	1,00	90,2	10	0,10	0,50	119,4
10	0,10	0,80	90,4	10	0,10	0,90	120,3
70	0,15	0,80	90,9	30	0,05	1,00	120,4
31	**0,10**	**0,82**	**:Średnia ważona**	**18**	**0,08**	**0,79**	**:Średnia ważona**

Tabela 3 Wyniki najlepszych zestawów parametrów dla instacji gr96

Tabela 4 Wyniki najlepszych zestawów parametrów dla instancji pr136

ATSP - br17				ATSP - ftv47			
Populacja	**Mutacja%**	**Krzyzowanie%**	**Błąd %**	**Populacja**	**Mutacja%**	**Krzyzowanie%**	**Błąd %**
50	0,15	0,50	2,6	10	0,15	1,00	53,4
50	0,15	0,70	2,6	10	0,15	0,80	56,2
50	0,15	0,80	2,6	30	0,05	0,90	56,9
50	0,15	0,90	2,6	30	0,10	0,70	57,0
50	0,15	1,00	2,6	10	0,10	1,00	57,1
50	**0,15**	**0,70**	**:Średnia ważona**	**17**	**0,12**	**0,89**	**:Średnia ważona**

Tabela 5 Wyniki najlepszych zestawów parametrów dla instacji br17

Tabela 6 Wyniki najlepszych zestawów parametrów dla instancji ftv47

ATSP - ftv70				ATSP - ftv170			
Populacja	Mutacja%	Krzyzowanie%	Błąd %	Populacja	Mutacja%	Krzyzowanie%	Błąd %
10	0,15	0,60	87,5	30	0,05	0,60	184,7
10	0,15	0,70	87,5	30	0,05	0,50	185,2
10	0,15	1,00	88,4	10	0,10	0,70	186,1
10	0,15	0,80	88,8	10	0,10	0,80	188,8
30	0,05	0,80	88,9	30	0,05	0,90	188,8
11	**0,14**	**0,75**	**:Średnia ważona**	**23**	**0,07**	**0,64**	**:Średnia ważona**

Tabela 7 Wyniki najlepszych zestawów parametrów dla instacji ftv70

Tabela 8 Wyniki najlepszych zestawów parametrów dla instancji ftv170

Średnia arytmetyczna uzyskanych wyników			
TSP	Populacja	Mutacja%	Krzyzowanie%
STSP	46	0,12	0,82
ATSP	25	0,12	0,74
STSP & ATSP	36	0,12	0,78

Tabela 9 Średnia arytmetyczna uzyskanych wyników

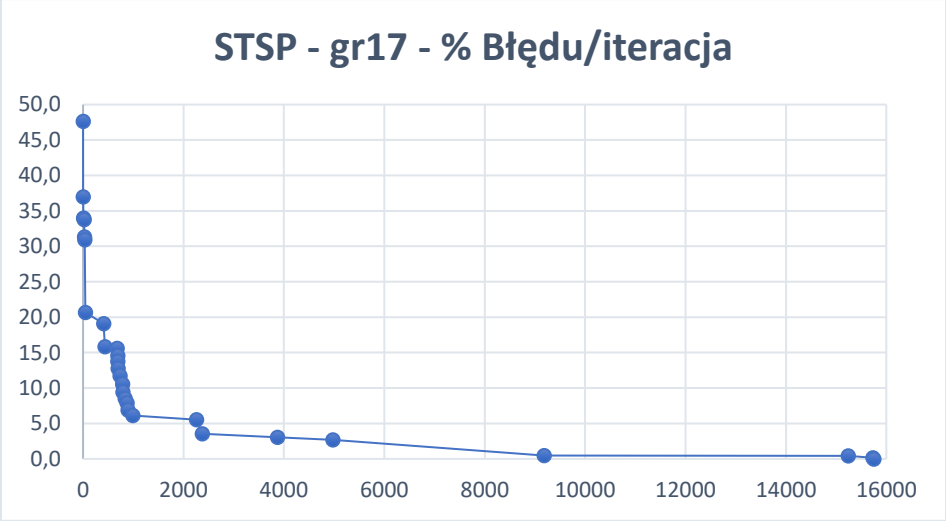
Na podstawie powyższych tabel możemy stwierdzić, co ciekawe, że nie zawsze więcej równa się lepiej. Rozmiar populacji na przykład w większości przypadków nie pomaga wcale uzyskać lepszego rozwiązania, jedynie dla mniejszych instancji problemu większe rozmiary populacji rzędu 50-70 zdają się generować lepsze rozwiązania, dla większych instancji lepiej sprawdzają się mniejsze populacje rzędu 10-30.

Optymalny poziom mutacji określa się w okolicach 12% szans na wystąpienie mutacji i tutaj podobnie jak w przypadku rozmiaru populacji, dla większych instancji zdaje się lepiej funkcjonować szansa mutacji mniejsza, ~10%, natomiast dla instancji mniejszych, dla których ważniejsze jest zróżnicowanie osobników, wyższa szansa mutacji, rzędu 15% przynosi lepsze rezultaty. 1% szansa nie pojawiła się w wynikach ani razu, podobnie 5% szans pojawia się sporadycznie co prowadzi do wniosku, że mutacja jest niezbędnym operatorem w algorytmie genetycznym.

Optymalna szansa zadziałania operatora krzyżowania, który jest kluczowym aspektem algorytmu genetycznego, wacha się w okolicach 78%. Świadczy to o tym że mimo iż szansa rzędu 50% czasami w wynikach występuje, tak sugerowanymi wartościami są te z okolic 75-85%, gdyż jest to główny element algorytmu i jedynie w niektórych przypadkach optymalniejszym rozwiązaniem jest zachowanie niezmienionego osobnika rodzicielskiego.

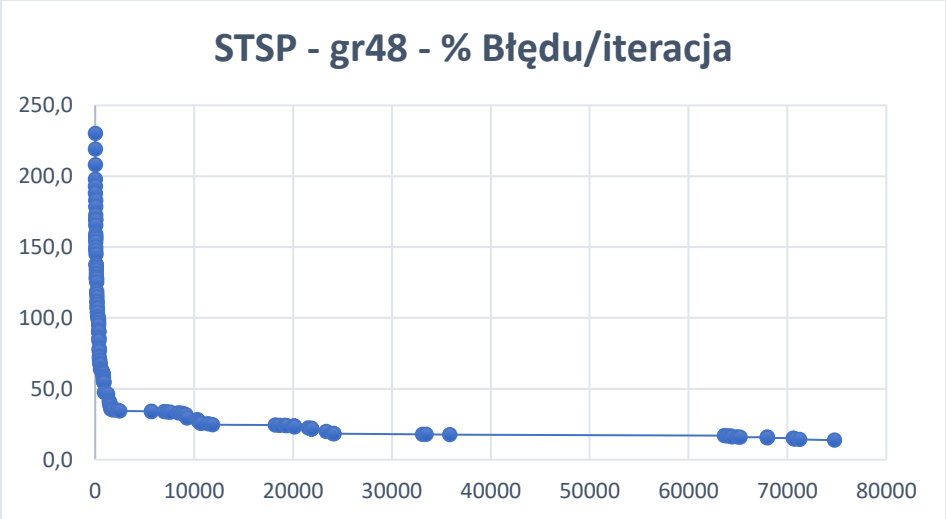
3.2. Otrzymany wynik a liczba iteracji

Na poniższych wykresach zawarto zależność % błędu od ilości iteracji algorytmu dla konkretnych instancji problemu przy wykorzystaniu parametrów wyliczonych w punkcie 3.1.. Dla każdego wykresu załączono tabele z informacją o poglądowym czasie[*s*] działania algorytmu oraz minimalnym % błędu jaki udało się uzyskać. Pełne wyniki dostępne w repozytorium projektu.



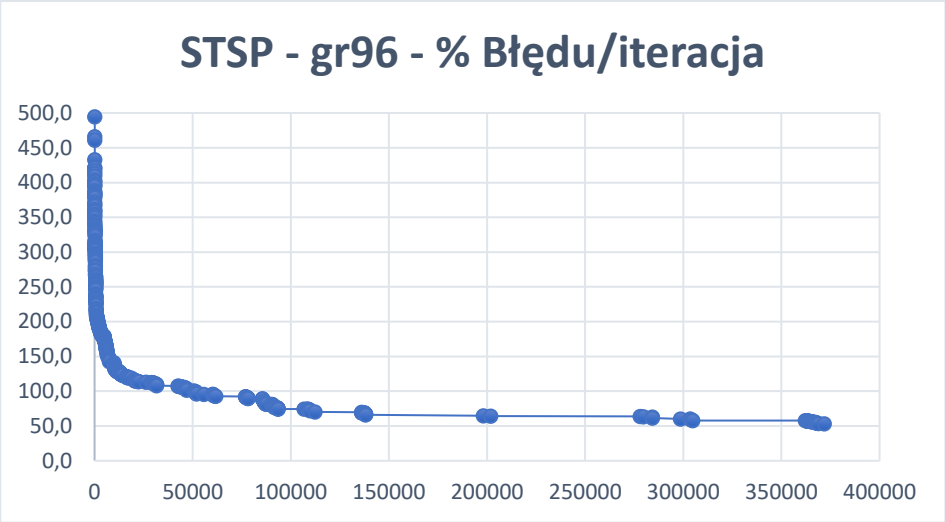
CZAS:	*3s*
Min % błędu	0,0

Wykres 1 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji STSP gr17



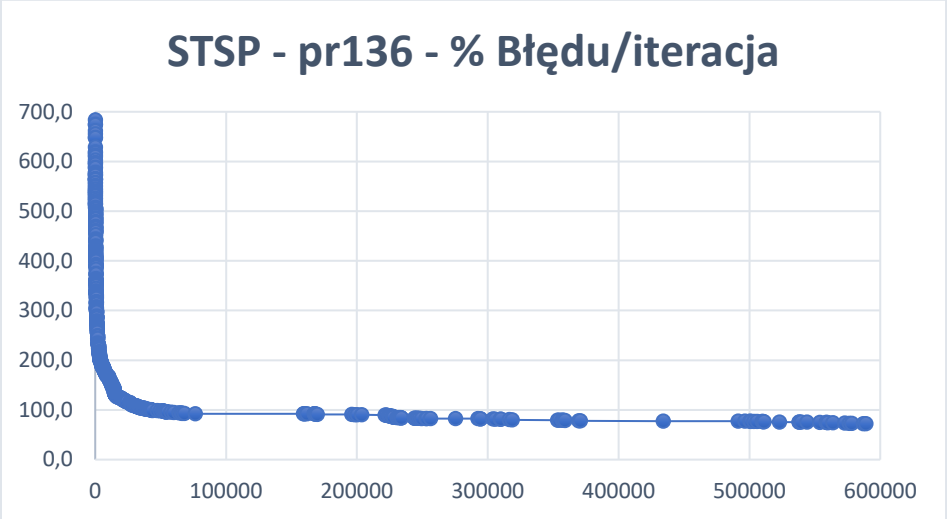
CZAS:	*15s*
Min % błędu	13,7

Wykres 2 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji STSP gr48



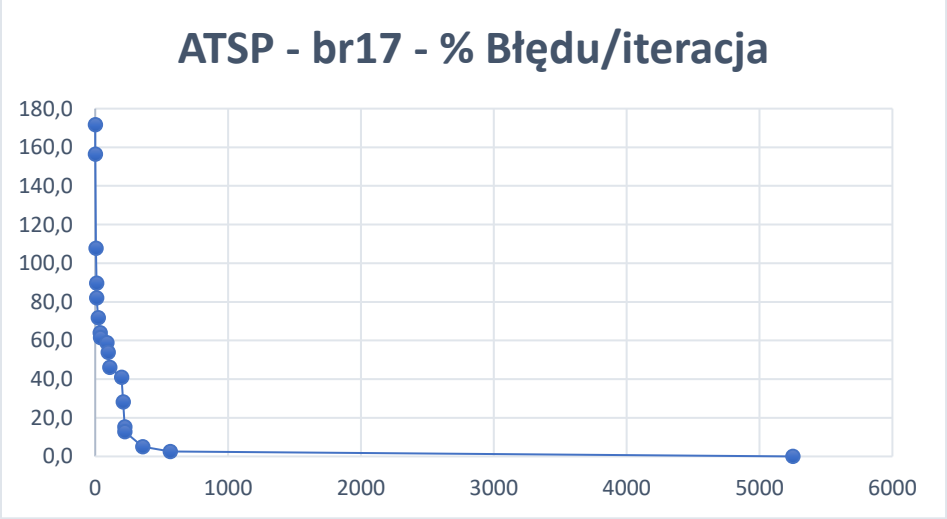
CZAS:	*20s*
Min % błędu	53,4

Wykres 3 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji STSP gr96



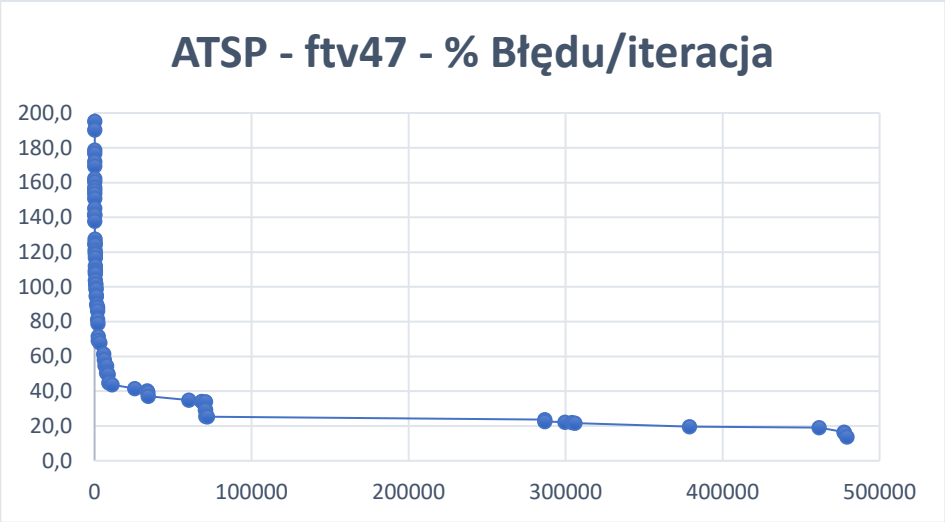
CZAS:	*30s*
Min % błędu	72,4

Wykres 4 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji STSP pr136



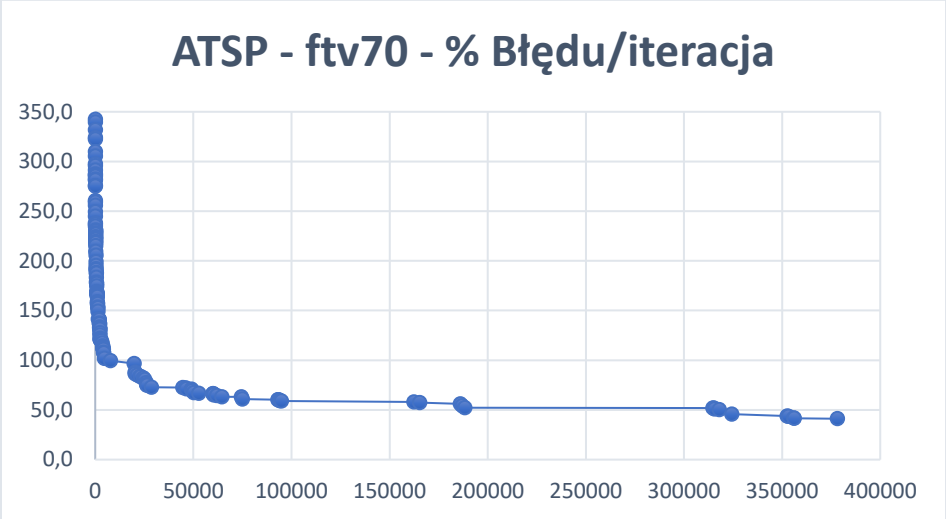
CZAS:	*1s*
Min % błędu	0,0

Wykres 5 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji ATSP br17



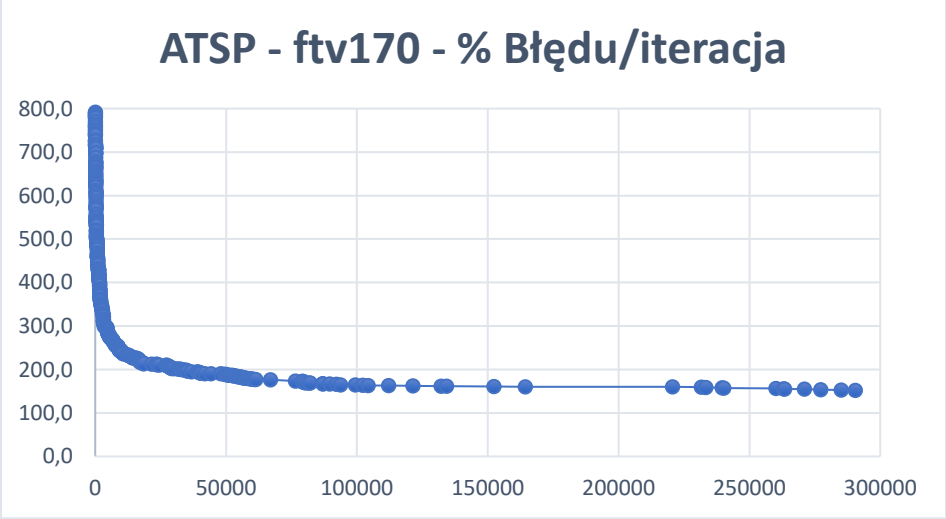
CZAS:	*10s*
Min % błędu	13,8

Wykres 6 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji ATSP ftv47



CZAS:	*10s*
Min % błędu	41,1

Wykres 7 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji ATSP ftv70



CZAS:	*20s*
Min % błędu	152,2

Wykres 8 Zależność % błędu rozwiązania od liczby iteracji algorytmu dla instancji ATSP ftv170

Z powyższych wykresów a konkretniej wykresu 1 oraz wykresu 5 możemy odczytać, że jedynie dla najmniejszych instancji problemu udało się uzyskać bezbłędny wynik w zadowalającym czasie 1s, która jest subiektywną granicą responsywności programu.

Dla reszty instancji, poza największymi badanymi, udało uzyskać się stosunkowo niski stopień błędu rzędu 10-50% w stosunkowo krótkim czasie rzędu 10-20s. W porównaniu jednak do uprzednio implementowanych algorytmów metaheurystycznych tabu search oraz simmulated annealing nie są to wyniki zadowalające, gdyż w przypadku wymienionych algorytmów otrzymywane rozwiązania obarczone były mniejszym % błędu, a otrzymywane były w krótszym czasie.

Na wykresach można zauważyć, że już po niewielkiej liczbie iteracji otrzymujemy rozwiązanie, które wraz z dłuższym działaniem algorytmu nie przechodzi już znaczących zmian, a odstępy między znalezieniem lepszego rozwiązania od obecnie posiadanego stają się z biegiem czasu coraz większe

4. Podsumowanie

- Algorytm Genetyczny jest stosunkowo łatwy w implementacji gdyż zasada jego działania opiera się na założeniach wziętych z podstaw biologii znanych prawie każdemu człowiekowi.

- Trudniejszą, lecz kluczową sprawą jest dobór odpowiednich parametrów działania algorytmu. Jest to skomplikowana sprawa ze względu na to, że algorytm ten inaczej zachowuje się w przypadku różnych instancji problemu i dla każdej z instancji inne parametry okazują się optymalne. Udało się uzyskać pewne zadowalające wyniki dla niektórych instancji, nie są one jednak optymalne ze względu na zawężony zakres testowanych zestawów danych.

- Czas wykonywania algorytmu nie był badaną wielkością ze względu na zbyt dużą liczbą zmiennych wpływającą na wynik oraz to że nie czas jest tu kluczowym czynnikiem, a wykorzystanie odpowiednich parametrów oraz operatorów mutacji i krzyżowania, co może prowadzić do znacznego zredukowania czasu wykonania algorytmu poprzez osiąganie wyników bliższych optimum w krótszym czasie