Projektowanie efektywnych algorytmów

Projekt

18.01.2020

248821 Michał Rajkowski

(6)Genetic Algorithm

Spis treści

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu algorytmu genetycznego, rozwiązującego problem komiwojażera w wersji optymalizacyjnej.

Problem komiwojażera (ang. Travelling salesman problem) jest to zagadnienie optymalizacyjne polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym.

Zagadnienie to można podzielić na dwa podtypy ze względu na wagi ścieżek łączących wierzchołki grafu. W symetrycznym problemie komiwojażera (STSP) dla dowolnych dwóch wierzchołków A B krawędź łącząca wierzchołek A z B ma taką samą wagę jak ta łącząca wierzchołek B z A. Natomiast w asymetrycznym problemie komiwojażera (ATSP) wagi te mogą się różnić.

Główną trudnością problemu jest duża liczba danych do analizy. W przypadku problemu symetrycznego dla n miast liczba kombinacji wynosi $\frac{(n-1)!}{2}$.

Dla problemy asymetrycznego jest to aż n!.

2. Metoda

2.1 Opis metody

Algorytm Genetyczny to rodzaj heurystycznego przeszukiwania przestrzeni rozwiązań naśladujący naturalne procesy ewolucji.

Poszukiwanie polega na tworzeniu rozwiązania opartego na współpracy osobników (populacji) wykorzystujących wiedzę o przestrzeni rozwiązań oraz na przekazywaniu jej kolejnym pokoleniom z wykorzystaniem mechanizmów ewolucyjnych.

Ewolucja odbywa się w postaci niewielkich zmian jakości osobników oraz całej populacji, dążących do możliwie najlepszego spełnienia narzuconych warunków przez środowisko.

2.2 Elementy Algorytmu Genetycznego mające wpływ na jakość i czas rozwiązania

Wielkość populacji N

Wielkość populacji wpływa zarówno na jakość, jak i czas otrzymanego rozwiązania. Powiększając populacje zwiększamy w niej różnorodność genów, co powinno prowadzić do otrzymywania lepszych osobników w wyniku selekcji w momencie każdej iteracji algorytmu. Mała wartość populacji zwiększa także szanse na utykanie w minimach lokalnych.

Należy jednak pamiętać, że wraz ze wzrostem populacji proporcjonalnie rośnie czas wykonywania poszczególnych iteracji algorytmu. Należy więc znaleźć optymalną proporcję wielkości populacji do czasu wykonywania algorytmu.

Skład początkowej populacji

Populacja początkowa może być generowana w sposób losowy, bądź za pomocą wykorzystania heurystyki.

Im bardziej losowa startowa populacja (większa mieszanina genów, geny mało podobne do siebie), tym szybciej proces ewolucyjny będzie następował. Startowa populacja o dużej losowości genów pozwala na uniknięcie marnotrawstwa zasobu czasu i utykania w puli podobnych genów, czekając na bardzo korzystne krzyżowanie, bądź mutacje.

Rozpoczęcie algorytmu od heurystyki pozwala na szybkie nakierowanie populacji na rozwiązania bliższe optymalnym, jednak dzieje się to kosztem pułapki wpadania w minimum lokalne.

Najlepsze efekty daje kombinacja tych dwóch metod. W przypadku mojego algorytmu nie korzystam z heurystyki, gdyż trochę przeczy ona idei algorytmu ewolucyjnego. Szybkie zwężanie puli genów prowadzi często szybkiego wpadnięcia w lokalne minimum i otrzymanie dostatecznie dobrego wyniku, jednak z małą możliwością wejścia na ścieżkę ewolucyjną prowadzącą do rozwiązania optymalnego.

Warunek stopu

Warunkiem stopu są najczęściej liczba iteracji, wyczerpanie dopuszczalnego czasu trwania algorytmu, bądź dostatecznie dobre zbliżenie się do wyniku.

Wybrałem jako warunek stopu liczbę iteracji, gdyż pozwoli ona najlepiej zilustrować czas działania algorytmu, oraz można łatwo pokazać jej wpływ na wynik końcowy.

Metody selekcji

Selekcja polega wyborze z puli organizmów, tych, które będą miały możliwość rozmnażania się. Następuje to na podstawie wartości przystosowania (fitness) oraz pewnej dozy losowości.

Istnieje wiele metod selekcji takich jak: Roulette Wheel Selection, Rank Selection, Steady State Selection, Tournament Selection czy Elitism Selection. Przeważnie najlepsze wyniki otrzymuje się poprzez kombinacje kilku metod.

W prezentowanym przeze mnie algorytmie wartość funkcji fitness jest obliczana poprzez odwrotność długości ścieżki danego osobnika. W ten sposób dłuższe ścieżki posiadają gorsze dostosowanie od krótszych.

Metoda selekcji to połączenie koła ruletki z elityzmem. Tworzona jest pula organizmów, które będą mogły się rozmnożyć mająca określony rozmiar. Do puli tej automatycznie dodawane jest x% organizmów o najlepszym fitness (elityzm). Następnie wykonywany jest algorytm ruletki, gdzie każdy organizm ma prawdopodobieństwo przedostania się do puli rozmnażania, proporcjonalne do wartości jego przystosowania fitness. W ten sposób wypełnia się pulę.

Metody sukcesji i prawdopodobieństwo krzyżowania

Podstawową metodą sukcesji jest sukcesja z całkowitym zastępowaniem. W nowej populacji jest tyle samo osobników co w starej i żaden osobnik ze starej populacji nie przechodzi do nowej.

Bardziej rozbudowaną metodą sukcesji jest częściowe zastępowanie. Do nowej populacji przechodzi część starej populacji, którą można określić w sposób losowy, czy też metodą elity ze ściskiem.

W zaprezentowanym przeze mnie rozwiązaniu, podczas krzyżowania się dwóch osobników istnieje szansa na nie powstanie potomstwa, co skutkuje przypisaniem rodziców do nowego pokolenia. Szansa ta to „prawdopodobieństwo krzyżowania".

Metody krzyżowania

Najczęstszymi metodami krzyżowania są PMX, OX oraz CX. W ogólności polegają one na wyborze 2 rodziców następnie skopiowaniu z rodzica pierwszego fragmentu genu i uzupełnieniu go genem rodzica drugiego. Prowadzi to do połączenia dwóch rozwiązań z pewnym stopniem wariacji i otrzymania potencjalnie lepszego wyniku.

W zaprezentowanym przeze mnie algorytmie zastosowałem metodę Order Crossover Operator (OX). Polega ona na wycięciu fragmentu ścieżki z rodzica pierwszego, pobranie kolejnych wierzchołków ścieżki rodzica drugiego, ignorując wierzchołki występujące już w pierwszym fragmencie. Finalnie obie ścieżki są scalane tworząc cykl hamiltonowski.

Metody i prawdopodobieństwo mutacji

Mutacje powodują zwiększenie dywersyfikacji pomiędzy dwoma kolejnymi pokoleniami. Są to zmiany w organizmach niezależne od ich rodziców i występujące z określonym prawdopodobieństwem.

Ważne jest aby szansa na wystąpienie mutacji była niska. W przeciwnym razie prowadzi to do zanegowania roli dziedziczenia DNA po rodzicach i przemiany algorytmu w prymitywny algorytm losowy.

Obrany przeze mnie sposób mutacji polega na zamianie miejscami 2 losowych wierzchołków ścieżki osobnika, które następuje z określonym prawdopodobieństwem.

2.3 Pseudokod

Algorytm Genetyczny:
1. Wybór populacji początkowej chromosomów (losowy)
2. Ocena przystosowania chromosomów
3. Sprawdzenie warunku zatrzymania
    a. selekcja chromosomów – wybór populacji macierzystej
    b. krzyżowanie chromosomów z populacji rodzicielskiej
    c. mutacje
    d. ocena przystosowania chromosomów
    e. utworzenie nowej populacji
4. Wyprowadzenie najlepszego rozwiązania

2.4 Teoretyczna złożoność obliczeniowa

Na podstawie pseudokodu zastanówmy się nad teoretyczną złożonością obliczeniową algorytmu.

2.4.1 Teoretyczna złożoność obliczeniowa pamięciowa

Zużycie pamięci zależy od liczby przechowywanych osobników oraz długości genów (ścieżek). W każdej iteracji pętli głównej starzy osobnicy nadpisywani są nowymi. Zatem możemy stwierdzić że złożoność pamięciowa powinna przystawać do O (k* N), gdzie k to rozmiar populacji a N to ilość wierzchołków TSP.

2.4.2 Teoretyczna złożoność obliczeniowa czasowa

Zastanówmy się nad złożonością obliczeniową poszczególnych punktów pseudokodu. Przyjęte zostały następujące oznaczenia:

N - liczba wierzchołków TSP

k – rozmiar populacji

m – liczba iteracji pętli głównej

(1) Wybór populacji początkowej, to wygenerowanie startowej ścieżki dla każdego osobnika populacji. Na wygenerowanie jednej ścieżki następuje za pomocą metody vector random shuffle działającej w czasie O(N). Zatem całość odbędzie się w czasie O(k*N).

(2) Ocena przystosowania chromosomów polega na obliczeniu długości ścieżek wszystkich osobników populacji: O(k * N).

(3) Pętla główna z warunkiem zatrzymania. Wykona się m iteracji. Złożoność tego punktu to O(m * X), gdzie X to sumaryczna złożoność operacji wewnątrz pętli.

(a) Elityzm zostaje wykonany w ze złożonością quicksorta – O(k log k). Natomiast ruletka zostaje wykonana ze złożonością O(k* N).

(b) Krzyżowanie to O(k * N), bowiem dla k osobników musimy przejść po całej długości ich genów aby utworzyć nowy organizm.

(c) Mutacje są sprawdzane dla każdego osobnika, następnie może nastąpić zamiana 2 genów: O(k)

(d) Obliczenie fitness to O(k*N) (patrz punkt 2)

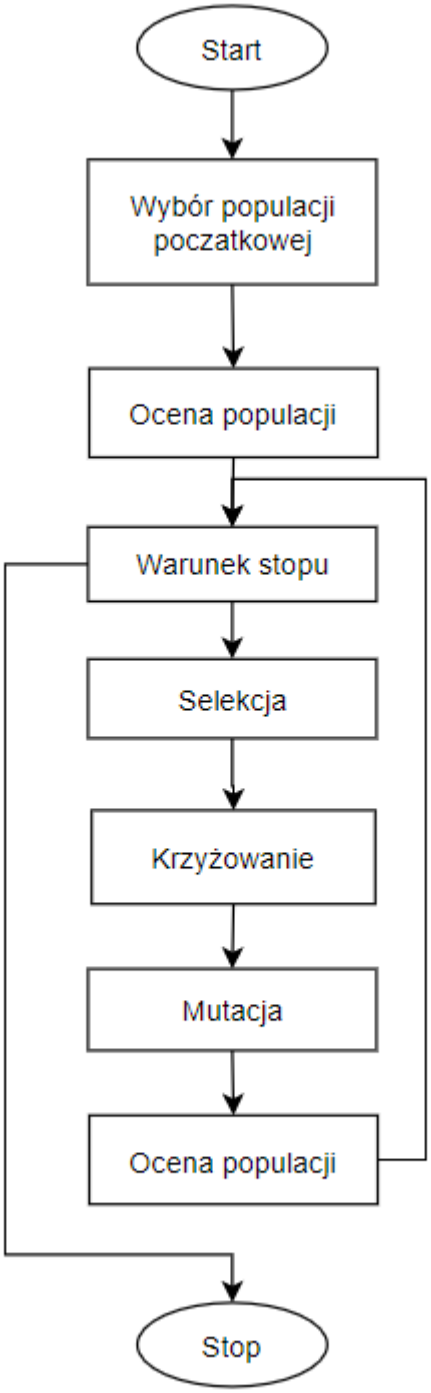
(e) utworzenie nowej populacji to tak naprawdę zamiana populacji starej na nowootrzymaną ze złożonością O(k*N). Ten punkt jest zawarty w krzyżowaniu.

(4) Wyprowadzenie najlepszego rozwiązania: O(1). Jego kalkulacja jest powiązana z kalkulacją fitness

Zatem sumując wszystkie numeryczne punkty 1-4 otrzymujemy całkowitą złożoność:

$$O(m*(k*N + k \log k))$$

3. Algorytm



Rys 1: schemat blokowy algorytmu

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu oraz do badań wybrano następujący zestaw instancji:

- tsp_13.txt
- tsp_17.txt
- gr24.txt
- ftv70.txt
- kroA100.txt
- gr120.txt
- rbg323.txt
- rbg443.txt

tsp_13.txt - tsp_17.txt to pliki testowe pochodzący ze strony doktora Mierzwy:

http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

Pliki gr24.txt ,gr120.txt oraz kroA100.txt zostały utworzone na bazie plików z:

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

ftv70, rbg323.txt oraz rbg443.txt:

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html

i zostały okrojone o niepotrzebne nagłówki pliku aby łatwiej było obsługiwać je w programie, stąd rozszerzenie txt.

5. Procedura badawcza

Pomiary zostały wykonane wielokrotnie, a celem ich było zbadanie wpływu zmiany prawdopodobieństwa krzyżowania, prawdopodobieństwa mutacji, wielkości populacji oraz liczby iteracji na błąd względny oraz czas otrzymania wyniku.

W najbardziej profesjonalnej formie, badanie wpływu zmiany wartości parametrów na czas trwania oraz błąd względny obliczeń byłby optymalizacją funkcji 4 zmiennych. Rozwiązanie to byłoby jednak związane z bardzo długim czasem pomiarów (x^3 razy większym), więc przy kilkugodzinnym wykonywaniu pomiarów dla poprzednich list, pomiary te obliczaneby były kilkadziesiąt godzin. Z tego względu skorzystam z uproszczonego modelu optymalizacji.

Na podstawie niewielkich testów przyjmę parametry metodą a priori. Następnie dla tak otrzymanego zbioru parametrów będę testował wpływ zmiany każdego z nich na wyniki obliczeń. W ten sposób oda mi się eksperymentalnie otrzymać zbiór najlepszy, zakładam bowiem, że mimo iż parametry są od siebie zależne, to wszystkie posiadają przedziały bliskie optymalności, na tyle szerokie aby odnaleźć je tą metodą.

Niech dane a priori będą równe:

- Liczba iteracji: 10 000
- Rozmiar populacji: 100
- Rozmiar elity: 10%
- Mating pool: 50%
- Szansa mutacji: 5%
- Prawdopodobieństwo rozmnażania: 90%

Zmierzono błąd względny oraz czas obliczeń dla danych a priori.

Następnie zmierzono błąd względny oraz czas obliczeń w zależności od:

- liczby iteracji (1000 - 50000)
- Rozmiaru populacji (50 - 1000)
- Prawdopodobieństwa rozmnażania (100% - 50%)
- Szansy mutacji (0% - 50%)

Dane wejściowe przedstawione były w następującej formie: (nazwa pliku, typ pliku, liczba wykonań, optymalne rozwiązanie (typ pliku to specjalne słowo ułatwiające odczytywanie plików z różnym typem zapisu danych)):

tsp_15.txt full 10 291

tsp_17.txt full 10 39

gr24.txt half 10 1272

ftv70.txt full 10 1950

kroA100.txt points 10 21282

gr120.txt gr 10 6942

rbg323.txt full 5 1326

rbg443.txt full 5 2720

final_test.csv

Dane wyjściowe każdych pomiarów zapisywano do osobnych plików. Następnie ubogacano je o wykresy. W następujących plikach można zobaczyć dane obliczeń:

Raw_błąd.xls

Raw_czas.xls

Wyniki.xls

5.1 Metoda pomiaru czasu

Pomiary czasu uzyskane zostały przy pomocy funkcji clock pochodzącej z biblioteki time.h.

Funkcja clock zwraca czas zużyty przez procesor w „clock ticks”. Następnie otrzymane tiki zostają przekonwertowane na milisekundy przy pomocy makra CLOCKS_PER_SEC. Za ostateczne obliczanie czasu odpowiedzialna jest następująca linia kodu:

```
cpu_time = ((double) (end - start)) / (CLOCKS_PER_SEC/1000);
```

5.2 Sprzęt

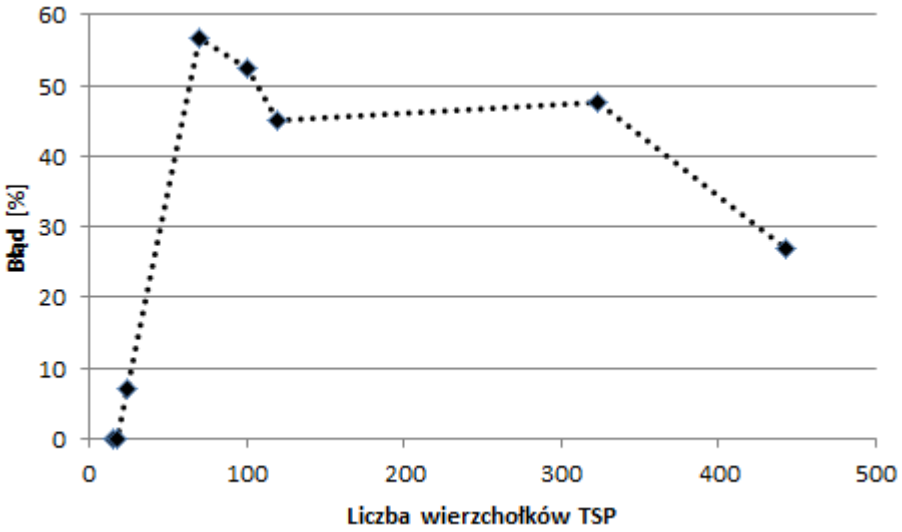
Procesor : Intel Xeon x5450, 4 rdzenie, 3.00 GHz

RAM: 8.00 GB

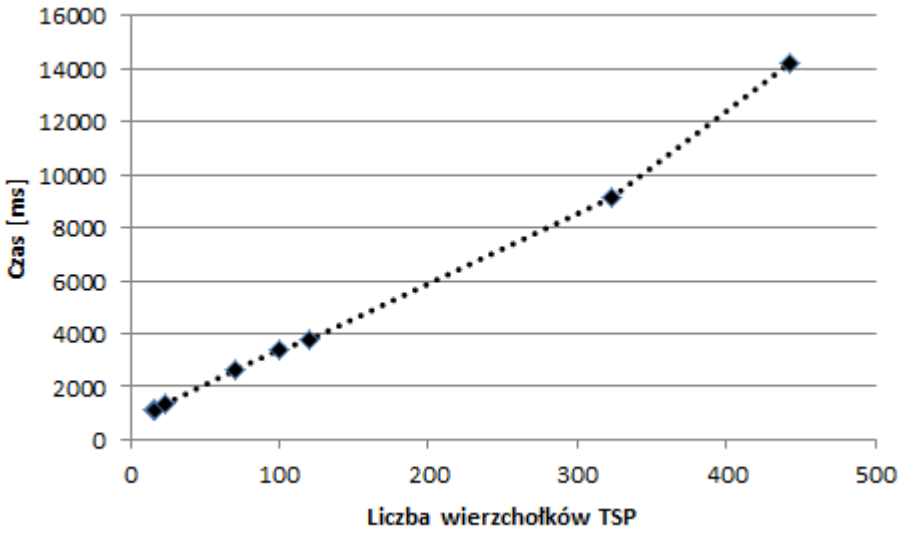
System: 64-bitowy system operacyjny, procesor x64

6. Wyniki

6.1 Parametry „a priori”

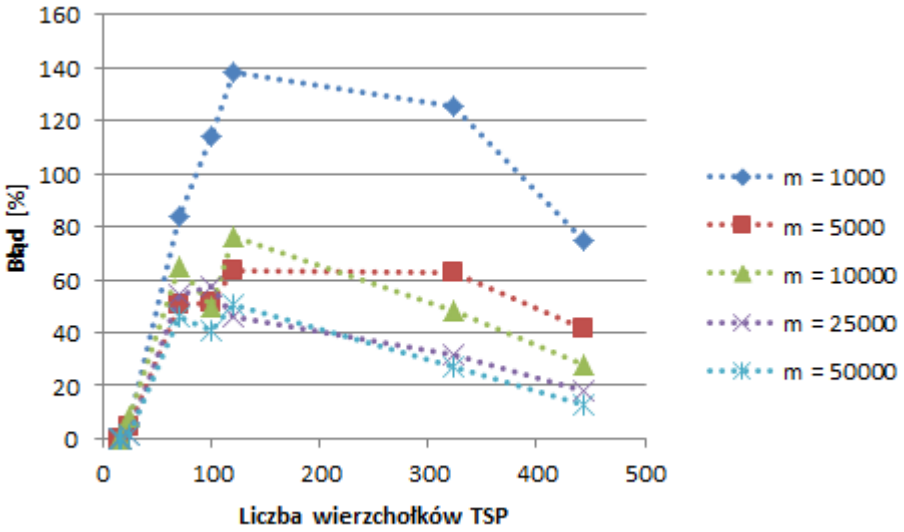


Wykres 1: Wykres zależności błędu względnego od rozmiaru instancji TSP dla danych a priori

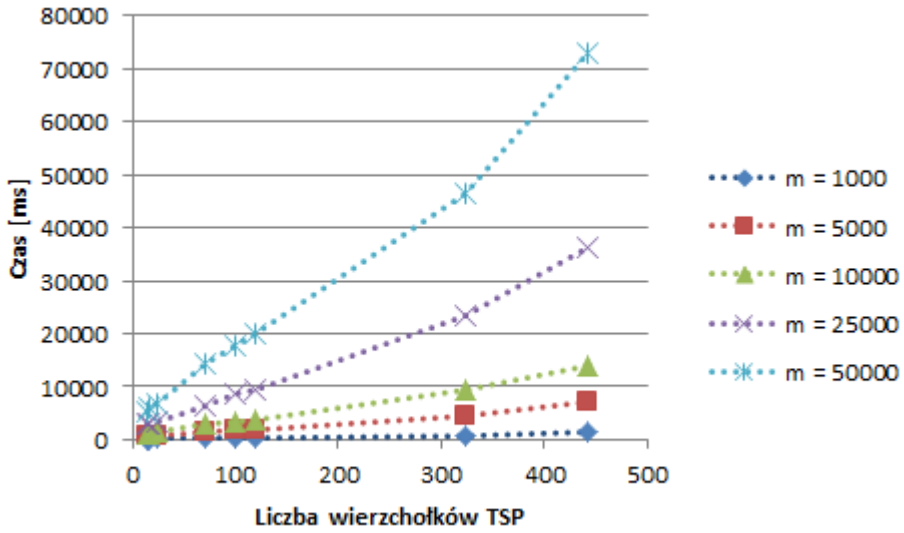


Wykres 2: Wykres zależności czasu obliczeń od rozmiaru instancji TSP dla danych apriori

6.2 Liczba iteracji

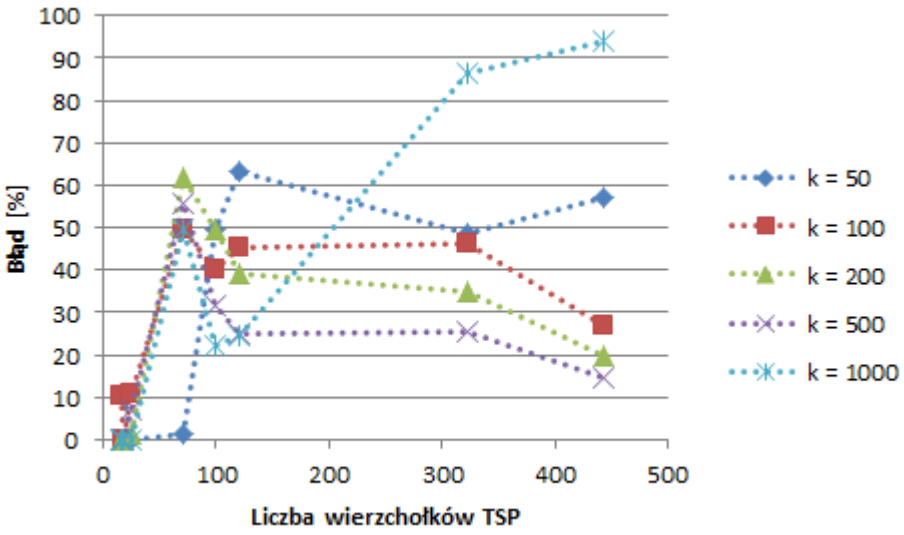


Wykres 3: Wykres zależności błędu względnego od rozmiaru instancji TSP i liczby iteracji algorytmu

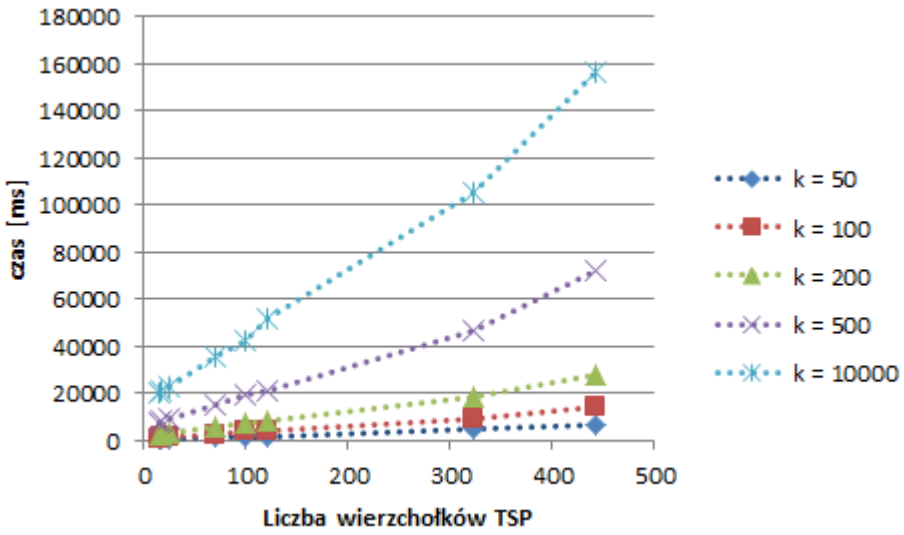


Wykres 4: Wykres zależności czasu obliczeń od rozmiaru instancji TSP i liczby iteracji algorytmu

6.3 Rozmiar populacji

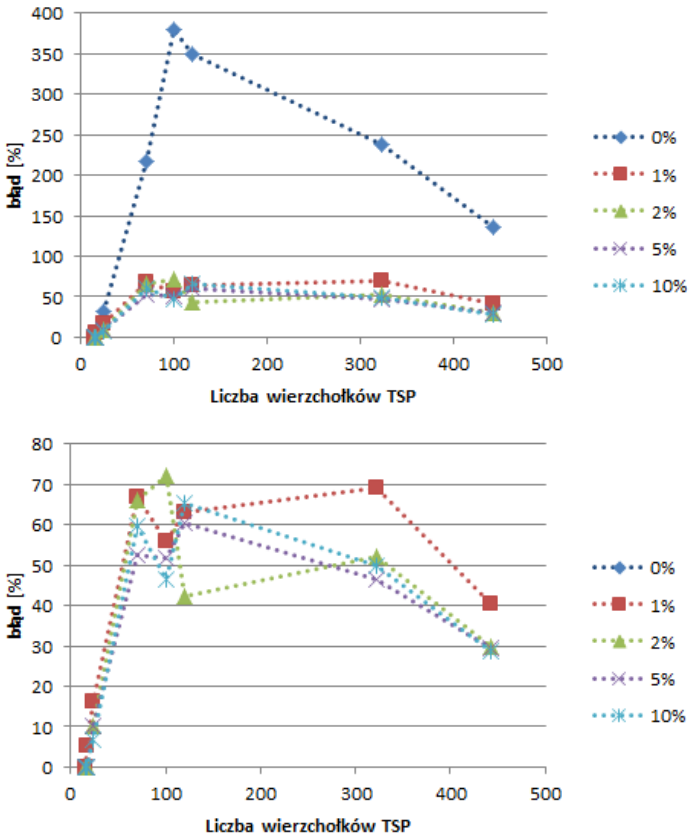


Wykres 5: Wykres zależności błędu względnego od rozmiaru instancji TSP i rozmiaru populacji

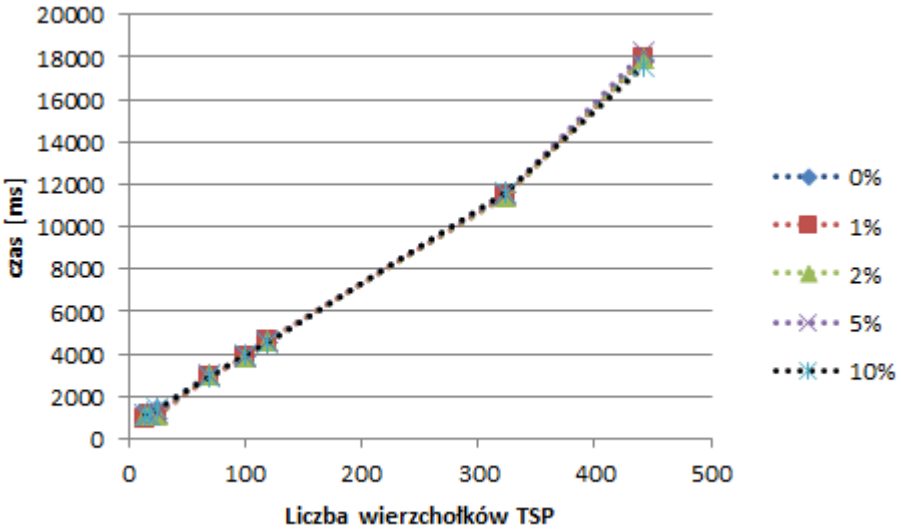


Wykres 6: Wykres zależności czasu obliczeń od rozmiaru instancji TSP i rozmiaru populacji

6.4 Prawdopodobieństwo mutacji

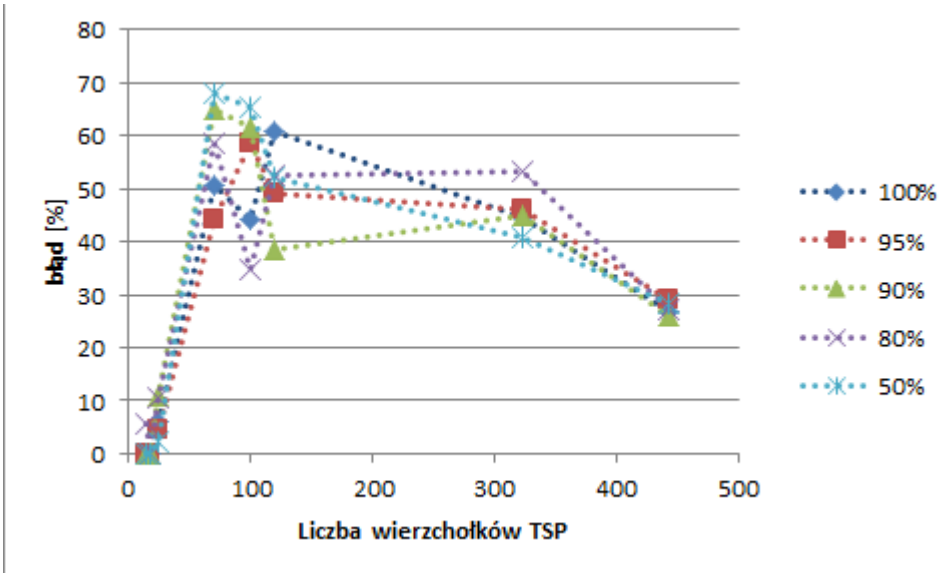


Wykres 7: Wykres zależności błędu względnego od rozmiaru instancji TSP i szansy mutacji

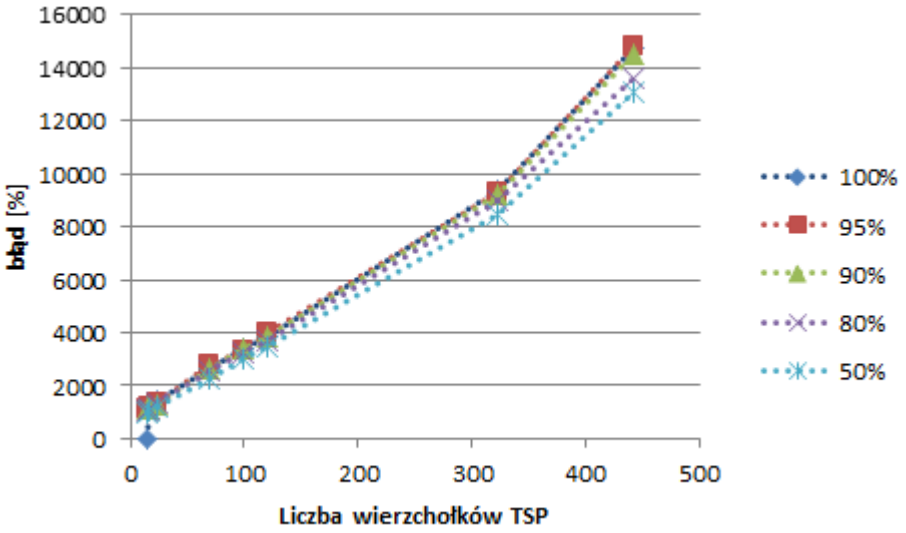


Wykres 8: Wykres zależności czasu obliczeń od rozmiaru instancji TSP i szansy mutacji

6.5 Prawdopodobieństwo krzyżowania

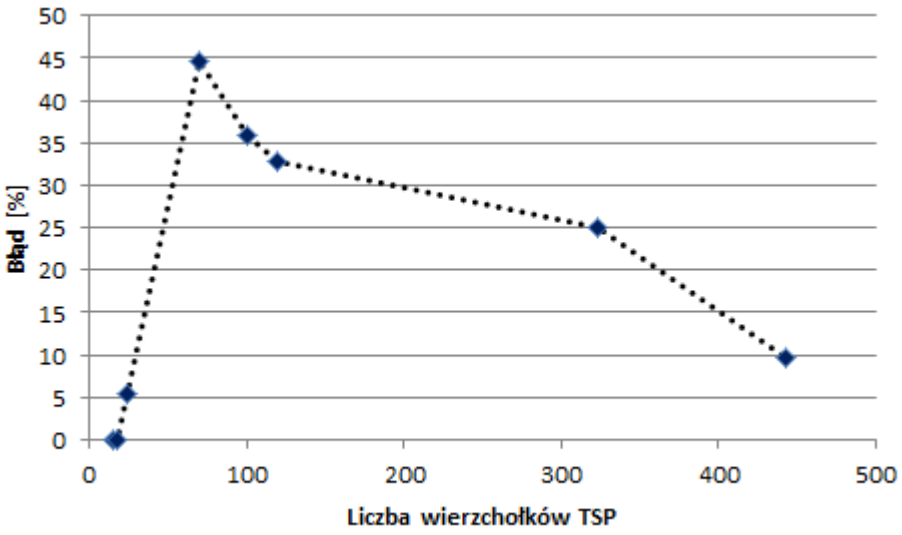


Wykres 9: Wykres zależności błędu względnego od rozmiaru instancji TSP i prawdopodobieństwa krzyżowania



Wykres 10: Wykres zależności czasu obliczeń od rozmiaru instancji TSP i prawdopodobieństwa krzyżowania

6.6 Najlepsze parametry



Wykres 11: Wykres zależności błędu względnego od rozmiaru instancji TSP dla optymalnych parametrów

Instancja	Rozmiar instancji	Wynik optymalna	Wynik otrzymany	Błąd względny [%]
tsp_15.txt	15	291	291	0,00
Tsp_17.txt	17	39	39	0,00
gr24.txt	24	1272	1342	5,50
Ftv70.txt	70	1950	2820	44,61
kroA100.txt	100	21282	28939	35,97
Gr120.txt	120	6942	9219	32,80
rbg323.txt	323	1326	1658	25,03
Rbg443.txt	443	2720	2982	9,63

Tab 1. Podsumowanie obliczeń dla ustalonych najlepszych parametrów

Ustalone parametry to:

- Liczba iteracji: 50 000
- Rozmiar populacji: 100
- Rozmiar elity: 10%
- Mating pool: 50%
- Szansa mutacji: 5%
- Prawdopodobieństwo rozmnażania: 100%

7. Obserwacje i wnioski

Po raz kolejny okazało się, że wykres błędu względnego w zależności od liczby wierzchołków TSP nie jest ściśle monotoniczną. (Wykres 11). Widzimy zatem, że dokładność wyników algorytmu genetycznego również zależy nie tylko od liczby wierzchołków ale także od tego jak wygląda wnętrze grafu, chaotyczność jego wag.

Na podstawie wyników pomiarów udało się udowodnić empirycznie kilka zależności związanych ze złożonością obliczeniową czasową algorytmu genetycznego:

- Czas wykonywania algorytmu zależy od rozmiaru populacji oraz liczby iteracji (Wykresy 4, 6). Jest to zgodne z obliczoną teoretyczną złożonością obliczeniową: O(m*(k*N + k log k)). Udało się także eksperymentalnie zaobserwować liniową zależność czasu od liczby wierzchołków (Wykres 2) co także jest zgodne z wydedukowanym wzorem.
- Prawdopodobieństwo krzyżowania oraz prawdopodobieństwo mutacji nie wpływa na czas wykonania algorytmu. Jest to zgodne z teoretycznym wzorem. Jest to także zależność łatwa do wydedukowania, bowiem szanse te używane są jedynie do modyfikacji danych, nie powodują powstawania szeregów operacji w pętlach.

Jak udało się zauważyć parametry algorytmu wydedukowane a priori były dużo gorsze od tych otrzymanych metodą a posteriori. Poprzez zmianę parametrów udało się zmieścić poniżej progu błędu względnego dla wszystkich danych testowych (Tab. 1).

Udało się dostrzec jak kluczowe są mutację dla poprawnego działania algorytmu. Wydawać by się mogło, że organizmy z losowej początkowej puli genów są w stanie dotrzeć blisko wartości optymalnej po długim czasie. Jak się jednak okazuje, bez losowych mutacji algorytm zwraca bardzo niedokładne wyniki (Wykres 7).

Podczas etapu projektowego udało się także natrafić na kilka ciekawych zjawisk, całkowicie sprzecznych z początkowymi przypuszczeniami:

- Najlepszą wartością parametru prawdopodobieństwa krzyżowania jest 100%. Nie jest to prawda dla ogółu danych, jednak wartość ta okazała się najlepsza, by zmniejszyć błąd względny mniejszych instancji kosztem nieznacznego jego zwiększenia dla instancji większych. Jest to jednak dziwne, że najlepszą wartością jest dokładnie 100% a nie 95% czy 90%. Nieczęsto zdarza się aby parametr przyjmował wartości ekstremalne. Nie jestem w stanie podać dokładnej przyczyny tego zjawiska. Moja hipoteza jest jednak taka, że jest to związane z elityzmem zastosowanym w programie. Najlepszym osobnikom udaje się oszukać ruletkę ewolucyjną, więc może nie ma potrzeby późniejszego ich zachowywania podczas krzyżowania.
- Wzrost rozmiaru populacji nie zawsze jest korzystny. Wydawać by się mogło, że im większa pula genów tym lepsze można z niej wyselekcjonować. Jednak badania pokazały (Wykres 5), że dla bardzo dużych populacji wyniki mogą być gorsze niż dla tych mniejszych. Możliwe, że posiadając bardzo dużą pulę genów trudniej jest utworzyć „zwarte grupy” i iść ich liniami ewolucyjnymi. Być może nie następuje wtedy ewolucja gatunku a po prostu mieszanie się losowych osobników. Jest to jednak jedynie moja hipoteza.