Sprawozdanie z projektu z kursu Projektowanie Efektywnych Algorytmów

Autor: Piotr Kołeczek

Numer albumu: 234940

Prowadzący projekt: Dr inż. Dariusz Banasiak

Grupa zajęciowa: środa 18:55 – 20:35

Termin oddania: 28.11.2019r.

Temat projektu: Implementacja i analiza algorytmu genetycznego, rozwiązującego asymetryczny problem komiwojażera (ATSP).

1. Wstęp.

Rozważanym problemem projektowym jest programowa implementacja asymetrycznego problemu komiwojażera. Ów problem polega na wyznaczeniu takiego cyklu Hamiltona w grafie pełnym ważonym, którego koszt jest możliwie jak najmniejszy. Problem ilustrowany jest często jako problem podróżnika, który ma za zadanie przejść przez wszystkie miasta w krainie przebywając możliwie jak najkrótszą drogę (lub poświęcając na podróż jak najmniej czasu).

Problem projektowy możemy rozważyć w przypadku, gdy droga z miasta **A** do miasta **B** jest taka sama jak z miasta **B** do miasta **A** oraz kiedy w przeciwnym kierunku **może** mieć inną wartość. Ta różnica dzieli problem na symetryczny (STSP) oraz asymetryczny problem komiwojażera (ATSP).

Problem komiwojażera jest problemem, który zalicza się do tak zwanych problemów *NP-trudnych,* czyli takich, dla których nie wynaleziono algorytmów, które rozwiązują ów problem w sposób optymalny o wielomianowej złożoności obliczeniowej. Algorytmy, które wyszukują optymalny cykl Hamiltona w grafie dla większych instancji grafów są bardzo złożone obliczeniowo i są bardzo czasochłonne, dlatego też często wykorzystuje się algorytmy liczące przybliżoną wartość optymalnej drogi.

Niniejsze sprawozdanie dotyczy implementacji programowej algorytmu **Genetycznego** dla asymetrycznego problemu komiwojażera.

2. Opis algorytmu.

Algorytm genetyczny jest przykładem algorytmu, dla którego nie ma pewności znalezienia optymalnego rozwiązania. Wynikiem algorytmu jest w zdecydowanej większości wartość przybliżona rozwiązania optymalnego. Algorytm z każdą kolejną generowaną populacją (zestawem rozwiązań) dąży do „udoskonalania" genetyki populacji, czyli tworzy nowe rozwiązania, będące „potomkami" rodziców poprzedniej generacji, a następnie odrzuca najgorsze rozwiązania. Sposób działania algorytmów genetycznych nieprzypadkowo przypomina zjawisko ewolucji biologicznej, ponieważ ich twórca **John Henry Holland** właśnie z biologii czerpał inspiracje do swoich prac. Obecnie zalicza się go do grupy algorytmów ewolucyjnych.

Algorytm genetyczny jest szczególnym przypadkiem algorytmu heurystycznego, który przeszukuje przestrzeń alternatywnych rozwiązań, by znaleźć to rozwiązanie najlepsze. Charakteryzuje go tak zwana **„ewolucja darwinowska"** – otóż jest to jego podstawowe założenie, które jest zgodne z naturalnym porządkiem, który musi zostać zachowany, aby dany gatunek żyjący przetrwał. Na to składa się kilka czynników:

1. Każde ***środowisko*** ma ograniczone zasoby - określa to maksymalną liczbę osobników (populacja) mogących utrzymać się w nim przy życiu,
2. Liczba osobników zmienia się w czasie - najczęściej rośnie; osobniki rozmnażają się, w wyniku czego populacja się powiększa,
3. Gdy osobniki się rozmnażają, ich liczba w populacji musi być regulowana – środowisko określa ***warunki przeżycia****;* aby dany osobnik przetrwał, musi się dostosować do tych *warunków;*
4. Z powyższego założenia wynika, że najlepiej przystosowane osobniki mają największą szansę na przeżycie i ***rozmnażanie***,
5. W procesie rozmnażania osobniki najczęściej przekazują swoje „przystosowanie" potomstwu - nie jest to regułą i możliwe jest wydanie potomstwa gorzej niż rodzice przystosowanego do warunków określonych przez środowisko,
6. Oprócz cech poprawiających przystosowanie, u potomstwa mogą występować losowe zmiany (mutacje), które mogą prowadzić do zmian ich przystosowania (lepsze lub gorsze przystosowanie),

Powyższe założenia ***ewolucji darwinowskiej*** stanowią tło działania algorytmu genetycznego. Zachowanie takiego naturalnego porządku z czasem (nowymi pokoleniami, osobnikami populacji) prowadzi do coraz lepszego przystosowania populacji do ogólnie panujących warunków, w których owe osobniki koegzystują.

W związku z powyższymi założeniami muszą zostać zdefiniowane pewne procesy, które wpisują się w ewolucję darwinowską. Otóż pojawiły się takie kluczowe stwierdzenia jak ***rozmnażanie, przekazywanie przystosowania potomstwu, losowe zmiany***. Otóż wyjaśnijmy te pojęcia trochę bardziej fachowymi zwrotami:

- **Selekcja** - polega na wyborze z bieżącej populacji osobników, których materiał genetyczny zostanie poddany operacji krzyżowania oraz mutacji i przekazany osobnikom potomnym (kolejna populacja). Wybór następuje na podstawie określonej metody selekcji. Osobniki oceniane są na podstawie tzw. funkcji przystosowania/dopasowania (ang. *fitness function - ff*). Selekcja powinna promować osobniki najlepiej przystosowane - o najwyższej wartości *ff*,
- **Krzyżowanie** - (ang. *crossover*) - polega na wymianie materiału genetycznego pomiędzy losowo wybranymi (podczas selekcji) parami osobników. W wyniku krzyżowania powstają nowe osobniki, które mogą wejść w skład nowej populacji (kolejnego pokolenia). Osobniki powstałe po krzyżowaniu powinny być - i są, jeśli metoda selekcji jest właściwa - lepiej przystosowane (wyższa wartość *ff*) od swoich rodziców. Zachodzi z prawdopodobieństwem p_c.
- **Mutacja** - polega na zamianie wartości losowo wybranego genu (cechy osobnika). Celem użycia operatora mutacji jest zapewnienie zmienności chromosomów. W przypadku wykorzystania AG do poszukiwania rozwiązania np. problemów kombinatorycznych stwarza możliwość wyjścia z optimów lokalnych i/lub zwiększenia intensyfikacji przeszukiwania. Zachodzi z prawdopodobieństwem p_m.

1. wybór populacji początkowej chromosomów (losowy)
2. ocena przystosowania chromosomów
3. sprawdzanie warunku zatrzymania
   a. selekcja chromosomów - wybór populacji macierzystej (*ang. mating pool*)
   b. krzyżowanie chromosomów z populacji rodzicielskiej
   c. mutacja - może być również wykonana przed krzyżowaniem
   d. ocena przystosowania chromosomów
   e. utworzenie nowej populacji
4. wyprowadzenie ,,najlepszego'' rozwiązania

Rysunek 1: Ogólny opis słowny algorytmu genetycznego dla rozwiązywanych problemów optymalizacyjnych.

3. Opis najważniejszych klas w projekcie.

3.1. Klasa Graph.cs

Jest to klasa reprezentująca załadowany graf do pamięci komputera. Posiada ona takie atrybuty jak macierz kosztów *costMatrix*, najlepszy koszt znalezionego cyklu Hamiltona *BestCycleCost*, liczba wierzchołków grafu *numOfCities*. Dzięki tej klasie możliwe jest załadowanie grafu z pliku tekstowego do programu.

3.2. Klasa Individual.cs

Klasa, która reprezentuje pojedynczego osobnika danej populacji. W tej klasie zdefiniowane są pola *Path*, która reprezentuje cykl Hamiltona (ciąg kolejnych wierzchołków w grafie), *PathCost* – jej wagę oraz flagę logiczną *isParent*, która jest potrzebna do selekcji osobników do rozmnażania - w populacji mogą się znajdować osobniki będące zarówno rodzicami, jak i potomkami, rozmnażać mogą się tylko rodzice.

W tej klasie na potrzeby łatwego porównywania obiektów zostały nadpisane metody ***Equals*** oraz ***HashCode*** z klasy **Object.**

3.3. Klasa Genetic.cs

Ta klasa odpowiada za implementację algorytmu genetycznego. Dziedziczy ona po klasie **Graph.cs** i dodatkowo definiuje takie atrybuty jak generator liczb pseudolosowych (*randomGenerator*) dla generowania pseudolosowych zjawisk tj. punkty odcięcia genów podczas procesu krzyżowania czy losowo wybranych punktów dla mutacji. Dodatkowo w tej klasie zostały zdefiniowane takie atrybuty jak *FinalRoute*, który reprezentuje stos najlepszego znalezionego rozwiązania w trakcie działania algorytmu genetycznego, pola odpowiedzialne za prawdopodobieństwo krzyżowania oraz mutacji (odpowiednio *crossProbability* oraz *mutationProbability*), a także pole *Population*, które jest listą obiektów klasy *Individual* – w taki sposób jest reprezentowana populacja osobników.

Najważniejsze metody tej klasy to:

- **private void CreatePopulation()** – metoda, która jest odpowiedzialna za stworzenie początkowej populacji osobników oraz przeprowadzenia wstępnej selekcji populacji macierzystej,
- **private void Mutate(Individual individual) –** metoda odpowiedzialna za losową mutację genu osobnika, która polega na przestawieniu dwóch losowo wybranych wierzchołków w ścieżce,
- **private int GetPathLength(int[] indexMatrix)** – metoda, która oblicza koszt cyklu Hamiltona na podstawie tablicy, w której zawarte są indeksy wierzchołków grafu, po których kolejno przechodzi „komiwojażer",
- **private Individual PMXChildCreation(Individual firstParent, Individual secondParent)** – metoda, która jest operatorem krzyżowania genów dwóch losowo wybranych rodziców za pomocą algorytmu **PMX** (ang. *Partially Mapped Crossover*). Algorytm krzyżowania PMX jest krzyżowaniem z częściowym odwzorowaniem – jest to odmiana krzyżowania dwupunktowego. Polega ona na wylosowaniu dwóch punktów odcięcia genu jednego z rodziców, które pozwalają nam określić przedział wierzchołków, które mają zostać odziedziczone u potomka. Następnie ten odziedziczony fragment genu przez osobnika potomnego jest uzupełniany o fragment genu drugiego rodzica, który nie powoduje konfliktu (odziedziczony ciąg miast od pierwszego rodzica jest uzupełniony o te miasta od drugiego rodzica, które jeszcze nie zostały „wstawione"). Po utworzeniu nowego potomka obliczana jest wartość jego cyklu Hamiltona, a następnie nowy osobnik jest dostawiany do populacji.
- **private void PopulationSelection(List<Individual> Population)** – dzięki tej metodzie możliwe jest przeprowadzenie selekcji (eugeniki) populacji. Polega ona na posortowaniu listy osobników w populacji względem kosztu cyklu Hamiltona rosnąco, a następnie usunięcie z listy „nadmiaru" osobników w populacji – przyjęto, że rozmiar populacji jest stały, z każdym pokoleniem nie zmienia się.
- **public void StartGeneticAlgorithm(int numOfGenerations) –** ta metoda jest programową implementacją algorytmu genetycznego. Algorytm genetyczny jest rozwiązywany według wszystkich powyższych założeń i przestaje się on wykonywać w momencie, gdy osiągnięta zostanie pewna liczba iteracji algorytmu (*numOfGenrations*), które to z kolei reprezentują kolejne generacje osobników w populacji.

3.3. Klasa Stack.cs

Ta klasa jest programową implementacją struktury stosu. Zostały w niej zdefiniowane typowe metody dla tej struktury:

- **public void Push(int number)** – metoda, która dodaje na szczyt stosu nową liczbę,
- **public void Pop()** – metoda, która strąca ze stosu liczbę,
- **public void Clear()** – metoda usuwająca wszystkie elementy znajdujące się na stosie.

4. Eksperyment.

W procesie wykonywania pomiarów posłużyłem się precyzyjnym czasomierzem **Stopwatch,** który zawarty jest w przestrzeni **System.Diagnostics.**

Do wykonania eksperymentu posłużyłem się macierzami grafów:

- **br17.atsp** (39),
- **ft53.atsp** (6905),
- **ftv170.atsp** (2755),
- **rbg323.atsp** (1326),
- **rbg358.atsp** (1163),
- **rbg403.atsp** (2465),
- **dantzig42.tsp** (699),
- **brazil58.tsp** (25395),
- **gr120.tsp** (6942),

W nawiasach zostały zapisane optymalne koszty cyklu Hamiltona dla pliku.

Dla każdej, przetestowanej wartości N zostało wygenerowanych **100** testów, a następnie wyniki działania algorytmu zostały uśrednione (średni koszt uzyskanego cyklu Hamiltona). Następnie dla uzyskanego średniego wyniku został obliczony błąd względny pomiaru, który wyraża się wzorem:

$$|f_{zn} - f_{opt}| \ / \ f_{opt}$$

gdzie:

f_{zn} – najlepsza obliczona wartość podczas działania algorytmu,

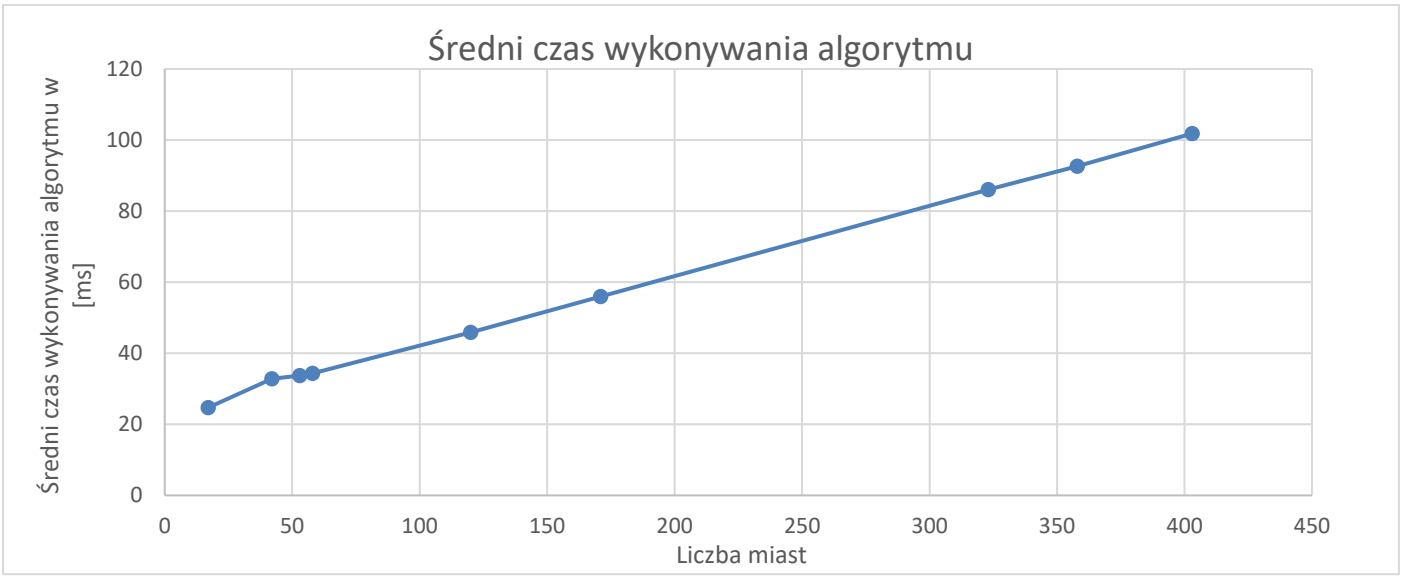
f_{opt} – wartość optymalna, będąca najlepszym znanym rozwiązaniem.

4.1. Wyniki pomiarowe.

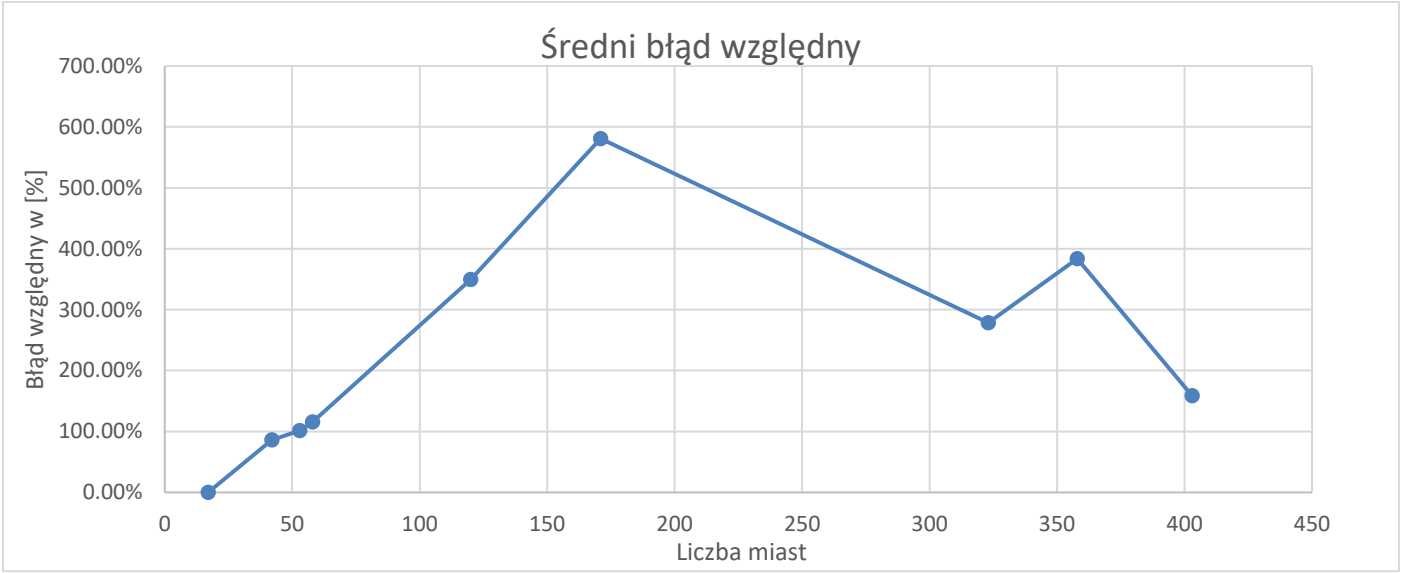
Za parametry testowe dla **100** prób pomiarowych przyjęto rozmiar populacji **sizeOfPopulation = 450** osobników oraz liczbę pokoleń **numberOfGenerations = 550** pokoleń. Poniżej zostały ukazane wyniki uzyskane podczas testów wyżej wymienionych macierzy dla tych parametrów:

Tabela 1: Wyniki 100 prób pomiarowych dla wskazanych macierzy testowych.

Nazwa pliku	Liczba wierzchołków grafu	Optymalny koszt cyklu Hamiltona	Uzyskana średnia waga cyklu	Błąd względny	Średni czas wykonywania algorytmu w [ms]
br17.atsp	17	39	39.00	**0.00%**	24.69
ft53.atsp	53	6905	13925.44	**101.67%**	33.72
ftv170.atsp	171	2755	18755.13	**580.77%**	55.93
rbg323.atsp	323	1326	5018.14	**278.44%**	86.02
rbg358.atsp	358	1163	5625.71	**383.72%**	92.64
rbg403.atsp	403	2465	6371.78	**158.49%**	101.83
dantzig42.tsp	42	699	1299.99	**85.98%**	32.83
brazil58.tsp	58	25395	54818.16	**115.86%**	34.32
gr120.tsp	120	6942	31213.67	**349.64%**	45.89



Rysunek 2: Średni czas wykonywania algorytmu GA dla wykonanych 100 prób testowych.



Rysunek 3: Średni błąd względny dla 100 przeprowadzonych testów wskazanych macierzy.

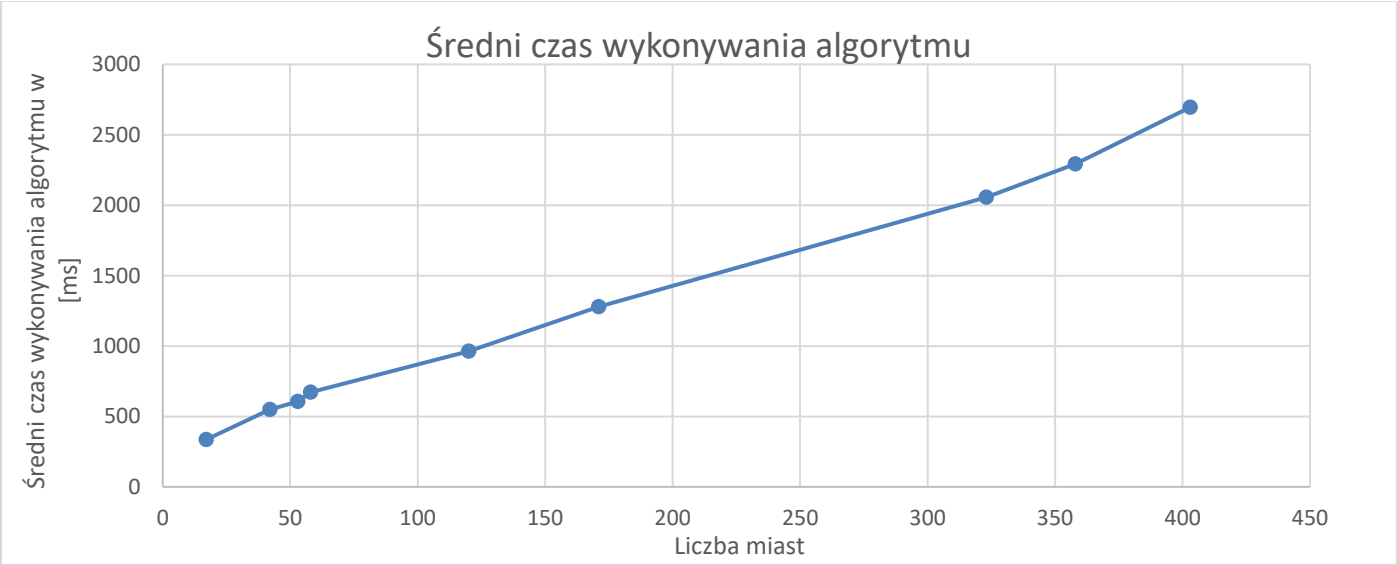
Niestety, pomimo że dobrane parametry dla algorytmu genetycznego pozwoliły mi wykonać serię **100** prób testowych w racjonalnym czasie, uzyskane wyniki są **fatalne** (błędy pomiarowe wynoszące ponad **580%**!), a co za tym idzie te parametry dla algorytmu genetycznego są także bardzo słabe.

Dla wszystkich badanych macierzy przeprowadzono jeden test, w którym ustalone zostały ustawione „bardzo dobre parametry testowe”:

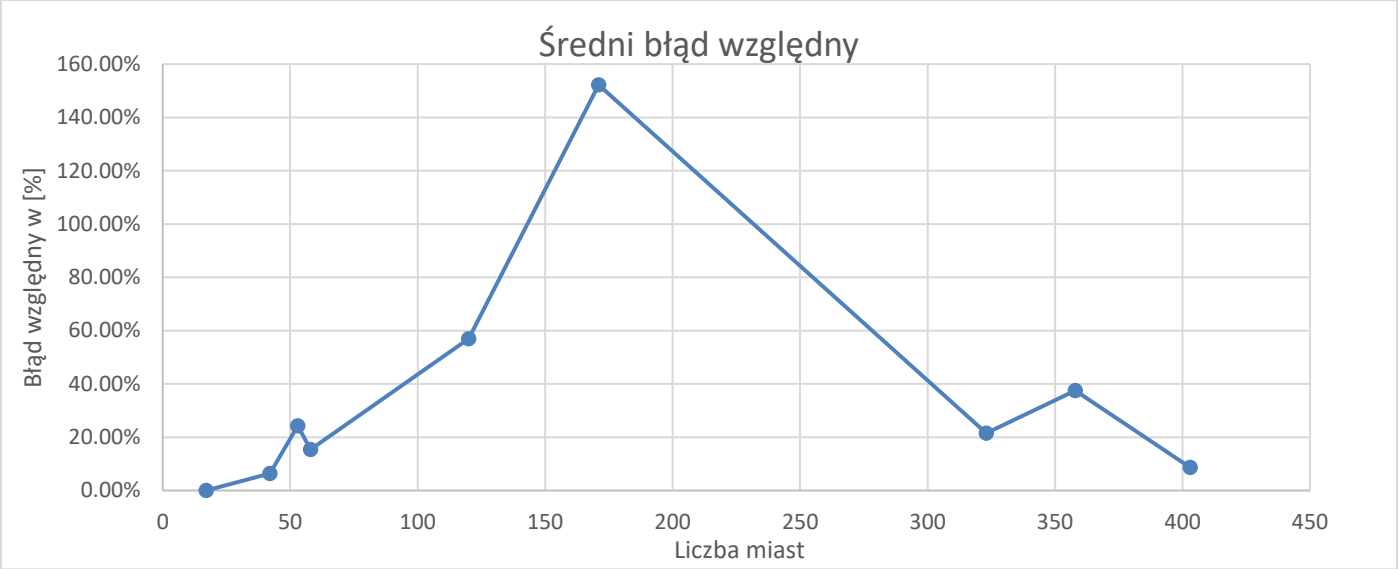
- Liczebność populacji **sizeOfPopulation = 50 osobników,**
- Liczba pokoleń **numberOfGenrations = 20000 pokoleń**.

Tabela 2: Zestawienie wyników dla dokładniejszej próby działania algorytmu genetycznego.

Nazwa pliku	Liczba wierzchołków grafu	Optymalny koszt cyklu Hamiltona	Uzyskana waga cyklu	Błąd względny	Czas wykonywania algorytmu w ms
br17.atsp	17	39	39	**0.00%**	335.47
ft53.atsp	53	6905	8580	**24.26%**	605.71
ftv170.atsp	171	2755	6948	**152.20%**	1279.42
rbg323.atsp	323	1326	1612	**21.57%**	2057.21
rbg358.atsp	358	1163	1599	**37.49%**	2291.58
rbg403.atsp	403	2465	2679	**8.68%**	2694.72
dantzig42.tsp	42	699	744	**6.44%**	548.52
brazil58.tsp	58	25395	29289	**15.33%**	671.30
gr120.tsp	120	6942	10895	**56.94%**	962.5432



Rysunek 4: Czas wykonywania algorytmu GA dla lepszych parametrów działania algorytmu.



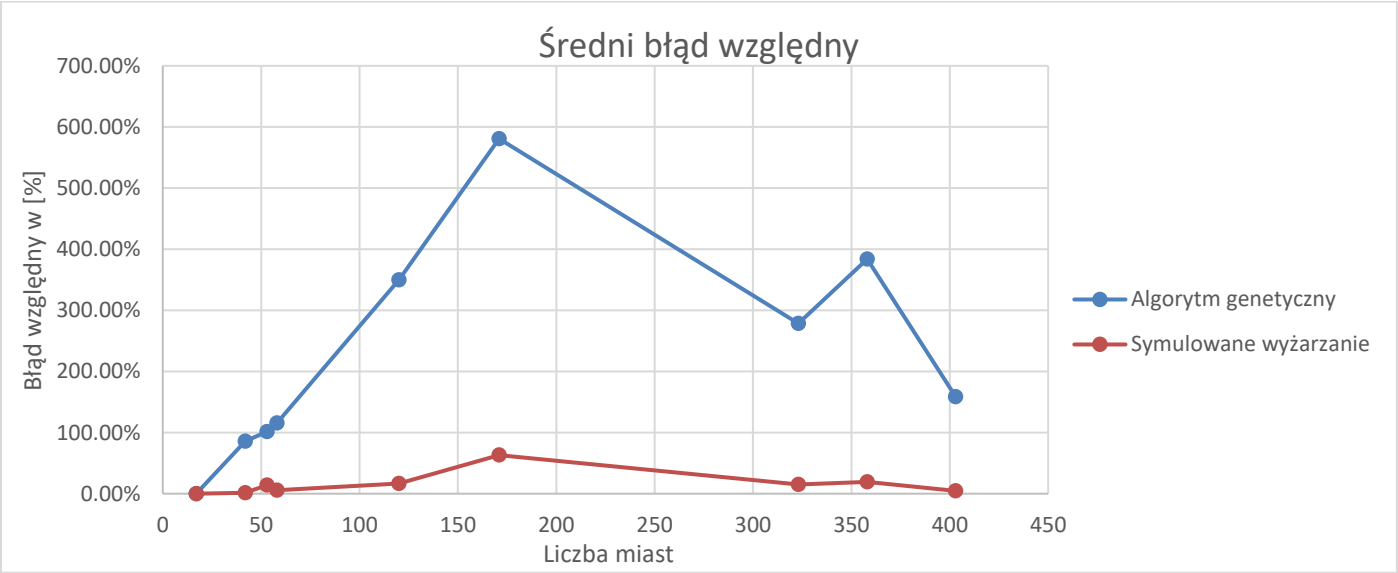
Rysunek 5: Błąd względny dla wyników działania algorytmu GA z lepszymi parametrami.

Jak można zauważyć, procentowy błąd względny przy zastosowaniu lepszych parametrów dla **GA** radykalnie zmalał.

Dla powyższych wyników algorytmu genetycznego dokonano także zestawienia porównawczego błędu względnego dla symulowanego wyżarzania. Za parametry testowe dla algorytmu symulowanego wyżarzania przyjęto:

- Temperatura początkowa wyżarzania $\mathbf{T_0}$ **= 100000 stopni,**
- Temperatura końcowa wyżarzania $\mathbf{T_{min}}$ **= 0.0001 stopni**,
- Współczynnik wyżarzania $\mathbf{T_{coefficient}}$ **= 0.99999**.

Poniżej zostało zaprezentowane porównanie obu tych algorytmów:

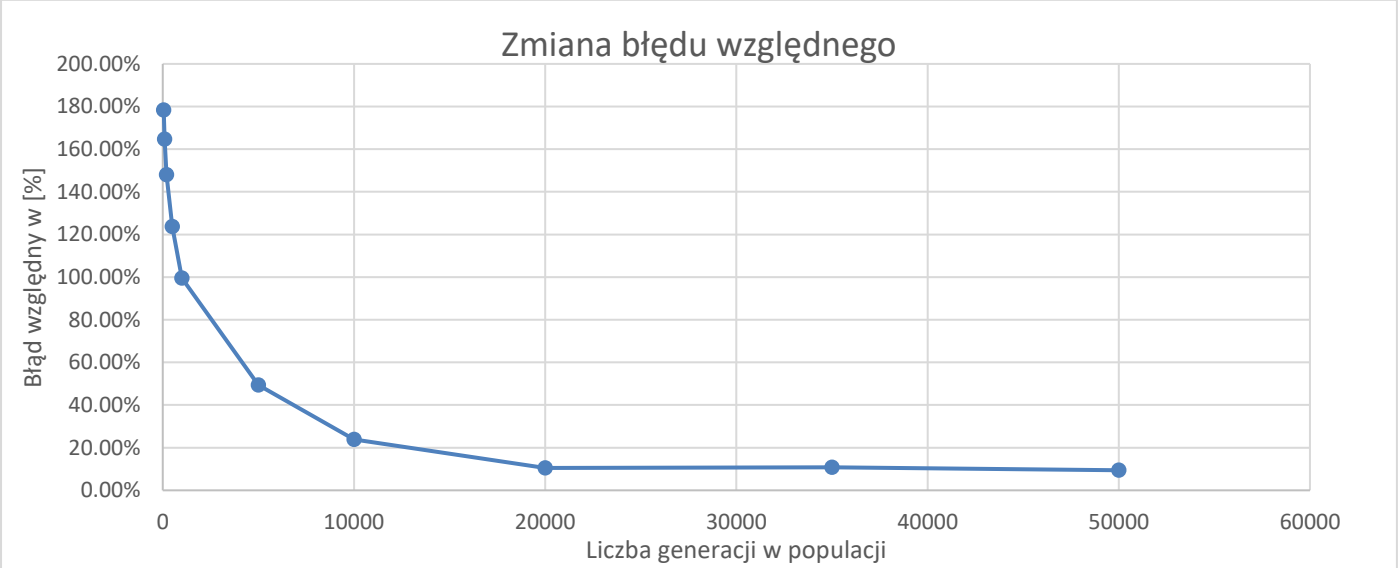


Rysunek 6: Zestawienie błędów względnych algorytmu SA oraz GA.

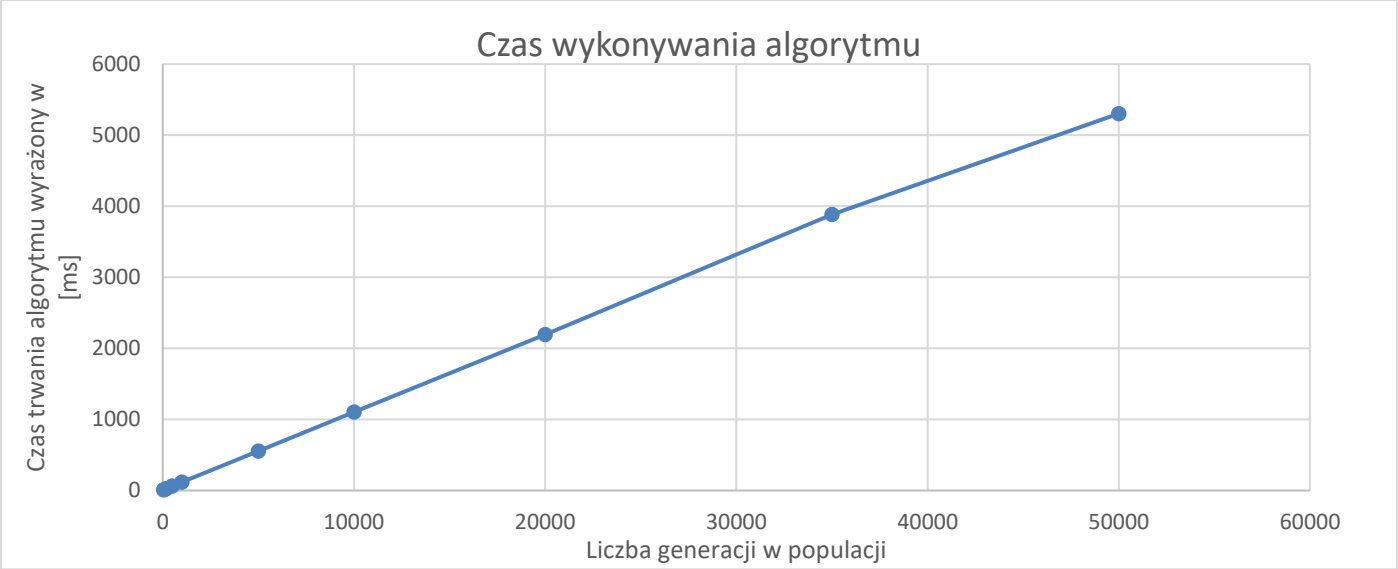
Wykonany został także pojedynczy pomiar testowy dla macierzy o **403** wierzchołkach (**rbg403.atsp**), ustawiając populację początkową na **100 osobników**. Poniżej został ukazany wykres zmiany błędu względnego w zależności o współczynnika wyżarzania:

Tabela 3: Wyniki testów algorytmu genetycznego dla różnych liczb generacji populacji.

Liczba generacji	Błąd względny w [%]	Optymalna wartość cyklu Hamiltona	Uzyskana wartość cyklu Hamiltona	Czas wykonywania algorytmu w [ms]
50	178.30%	2465	6860	10.79
100	164.75%	2465	6526	13.97
200	147.95%	2465	6112	25.88
500	123.73%	2465	5515	61.57
1000	99.47%	2465	4917	117.21
5000	49.41%	2465	3683	555.05
10000	23.81%	2465	3052	1106.43
20000	10.47%	2465	2723	2195.23
35000	10.75%	2465	2730	3883.29
50000	9.41%	2465	2697	5303.08



Rysunek 7: Wykres zmiany błędu względnego wyrażonego w [%} w zależności od zmiany liczby generacji populacji.



Rysunek 8: Czas wykonywania algorytmu genetycznego w zależności od zmiany liczby generacji w populacji.

4.2. Porównanie wyników z pozostałymi algorytmami.

Dokonano porównania czasów znajdowania rozwiązań optymalnych algorytmu Brute Force, programowania dynamicznego, symulowanego wyżarzania oraz algorytmu genetycznego. Przyjęto, że pomiar czasu wykonywania algorytmu genetycznego i symulowanego wyżarzania dobiegnie końca wtedy, gdy zostanie znaleziona właśnie optymalna wartość (warunek kończący wykonywanie algorytmu został zmodyfikowany w taki sposób, że jeśli znaleziony cykl Hamiltona przyjmuje optymalną wartość (która jest znana) to następuje przerwanie wykonywania algorytmu i zatrzymanie czasomierza). Zatem zmierzony czas dla symulowanego wyżarzania i algorytmu genetycznego jest czasem, po którym zostało znalezione rozwiązanie **optymalne.**

Ponieważ badane instancje grafów są bardzo małe (mniejsze niż **30** wierzchołków grafu) przyjęto za parametry dla symulowanego wyżarzania wartości:

- $\mathbf{T_0 = 100}$ stopni,
- $\mathbf{T_{min} = 0.0001}$ stopnia,
- Współczynnik schładzania $\mathbf{T_{coefficient} = 0.99999999}$.

Dla algorytmu genetycznego przyjęto parametry:

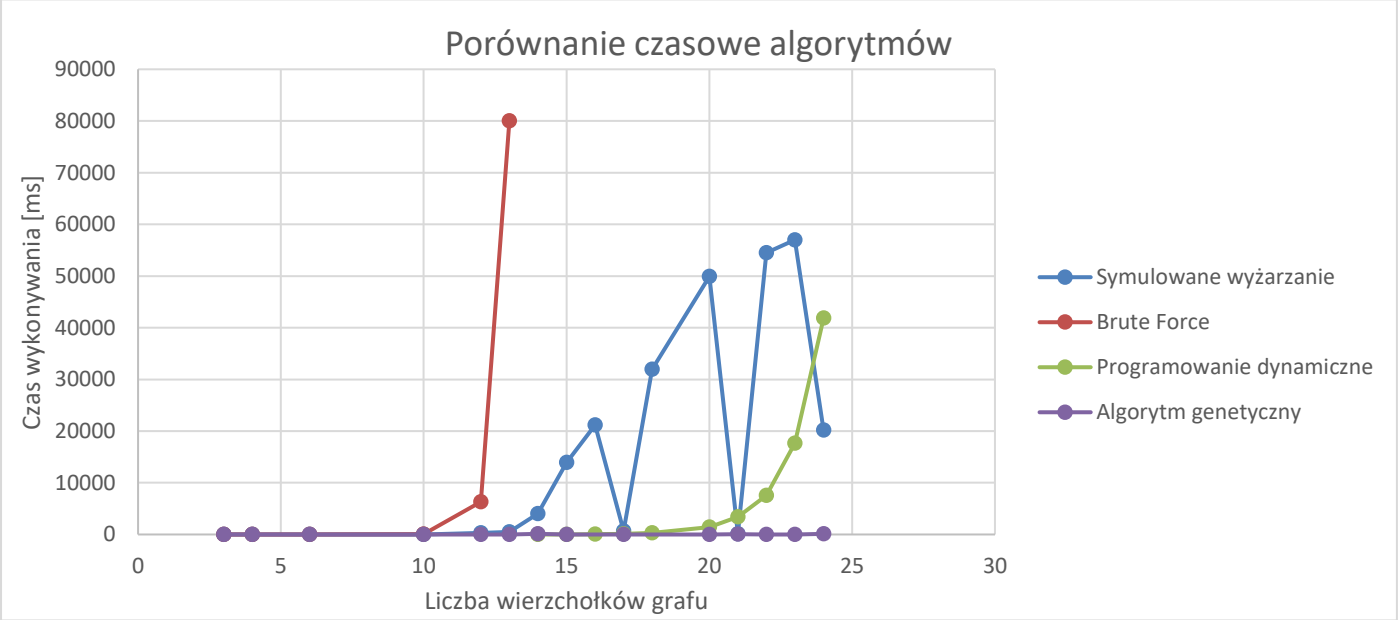
- Liczebność populacji **sizeOfPopulation = 75 osobników,**

Dla przeglądu zupełnego skończono pomiary czasowe na macierzy o **13** wierzchołkach (**tsp_13.txt**), ponieważ dla większych macierzy dla tego algorytmu czas wykonywania był zbyt długi. Z kolei dla programowania dynamicznego testy rozpoczęto od macierzy o **14** wierzchołkach (**tsp_14.txt**), gdyż dla mniejszych instancji czasy wykonywania algorytmu były bliskie 0 sekund. Dla symulowanego wyżarzania oraz algorytmu genetycznego zostały przeprowadzone testy dla wszystkich mniejszych instancji grafów.

Poniżej zostały zaprezentowane wyniki dla wszystkich czterech algorytmów:

Tabela 4: Zależności czasowe między rozmiarem grafu, a czasem rozwiązywania TSP przy użyciu algorytmów BF, DP oraz SA.

Nazwa pliku	Liczba wierzchołków grafu	Czas znajdywania optymalnego rozwiązania przez BF w [ms]	Czas znajdywania optymalnego rozwiązania przez DP w [ms]	Czas znajdywania optymalnego rozwiązania przez SA w [ms]	Czas znajdywania optymalnego rozwiązania przez GA w [ms]
tsp_3.txt	3	0.037	---	0.55	0.033
tsp_4.txt	4	0.04	---	0.68	0.041
tsp_6_2.txt	6	0.053	---	0.68	0.31
tsp_10.txt	10	51.15	---	5.06	1.15
tsp_12.txt	12	6314.35	---	280.07	4.30
tsp_13.txt	13	80019.76	---	488.48	8.10
tsp_14.txt	14	---	6.34	4051.01	121.48
tsp_15.txt	15	---	14.35	13907.9	5.05
data16.txt	16	---	35.71	21179.55	---
br17.atsp	17	---	95.51	688.51	9.81
data18.txt	18	---	274.84	31936.44	---
tsp_20.txt	20	---	1415.76	49891.51	14.24
gr21.tsp	21	---	3430.76	108.11	38.14
tsp_22.txt	22	---	7557.96	54485.85	7.79
tsp_23.txt	23	---	17656.16	56996.78	8.12
gr24.tsp	24	---	41841.83	20190.59	102.07



Rysunek 9: Porównanie czasowe dla algorytmów przeglądu zupełnego, programowania dynamicznego, symulowanego wyżarzania oraz algorytmu genetycznego w zależności od rozmiaru instancji grafu.

Uzyskane wyniki dla algorytmu symulowanego wyżarzania są bardzo ciekawe. Nadzwyczajne wyniki, które zostały uzyskane są dla macierzy **br17.atsp** (**17** wierzchołków), **gr21.tsp** (**21** wierzchołków) oraz **gr24.tsp** (**24** wierzchołki). Uzyskane takie czasy dla tych macierzy mają swoje uzasadnienie w specyfice grafów:

- Dla macierzy **br17.atsp** istnieją krawędzie łączące dwa wierzchołki o wadze **0**, a to sprawia, że istnieje kilka **optymalnych** cykli Hamiltona i to jednoznacznie przekłada się na wyższe prawdopodobieństwo wyliczenia optymalnego cyklu Hamiltona szybciej,
- Macierz **gr21.tsp** oraz **gr24.tsp** są macierzami, które reprezentują problem symetrycznego problemu komiwojażera – koszt dojścia z wierzchołka **A** do wierzchołka **B** ma taką samą wartość jak droga z wierzchołka **B** do wierzchołka **A**. Taka sytuacja sprawia, że także mamy wyższe prawdopodobieństwo obliczenia optymalnego cyklu Hamiltona szybciej.

Dla algorytmu genetycznego dla wszystkich zbadanych macierzy wszystkie uzyskane wyniki są co najmniej bardzo dobre. Dla niewielkich instancji macierzy ze względu na małą liczebność wierzchołków w grafie ewolucja darwinowska w bardzo krótkim czasie pozwala osiągnąć optymalne wyniki.

4.3. Analiza wydajności algorytmu genetycznego.

Dla algorytmu genetycznego dla wszystkich zbadanych macierzy wszystkie uzyskane wyniki są co najmniej bardzo dobre. Dla niewielkich instancji macierzy ze względu na małą liczebność wierzchołków w grafie ewolucja darwinowska w bardzo krótkim czasie pozwala osiągnąć optymalne wyniki.

Tak dobre wyniki zostały uzyskane ze względu na fakt, że procesy krzyżowania oraz mutacji miały kluczowe znaczenie podczas generowania wyników. Dla dużych instancji grafów (np. 300 i więcej) uzyskanie takich wyników w dobrym czasie graniczyłoby z cudem, ze względu na to, iż poziom „skomplikowania" organizmu, jakim jest nasza macierz grafu powoduje, że istnieje niewielkie prawdopodobieństwo tego, że podczas tych wyżej wymienionych operacji pseudolosowych zostaną uzyskane wartości optymalne.

5. Wnioski oraz podsumowanie.

Zadanie projektowe miało na celu zapoznać studenta z tematyką algorytmów populacyjnych, jakimi są m.in. algorytm mrówkowy (ACO, ang. *Ant Colony Optimization*) oraz zaimplementowany algorytm genetyczny (GA, ang. *Genetic Algorithm).* Algorytm genetyczny jest algorytmem, dla którego kluczowy jest dobór parametrów. W algorytmie przeglądu zupełnego (Brute Force) wystarczyło wczytać graf do pamięci komputera i obliczyć ścieżkę, identyczna sytuacja była w przypadku algorytmu programowania dynamicznego, a to ze względu na fakt, iż te algorytmy są algorytmami, które zwracają **zawsze** wynik **optymalny**, zatem gdy te algorytmy zakończą swoje działanie mamy stuprocentową pewność, iż uzyskane wyniki będą wynikami możliwie najlepszymi. Jednak mają one jedną, poważną wadę, która niemalże całkowicie przekreśla ich użyteczność – złożoność obliczeniowa. Algorytm Brute Force dla grafów o liczbie wierzchołków **13** i więcej jest już algorytmem bezużytecznym – liczba permutacji wierzchołków, która jest wymagana do sprawdzenia rośnie superwykładniczo (złożoność **O(n!)**).

Dla programowania dynamicznego sytuacja jest delikatnie lepsza, ponieważ zamysłem programowania dynamicznego jest podział problemu na wiele atomowych pod problemów i zapamiętywanie rozwiązań tych pod problemów – w przypadku tej metody możliwym było poprawienie szybkości znajdowania rozwiązania i ostatecznie zwiększenie liczby wierzchołków grafu do liczby **25.**

Dla symulowanego wyżarzania istnieje ogromne prawdopodobieństwo, że dla większych problemów (np. 100 wierzchołków i więcej) nie znajdziemy rozwiązania optymalnego w akceptowalnym czasie, jednakże możliwym jest uzyskanie wyniku co najmniej bardzo dobrego. Rozwiązując problem dla np. macierzy o **1000** wierzchołków, przy dobraniu odpowiednich parametrów działania algorytmu możliwe jest uzyskanie dosyć dobrego rozwiązania w akceptowalnym czasie.

Z kolei algorytm genetyczny jest algorytmem tak jak **SA,** uczącym się. Z każdą kolejną iteracją działania algorytmu generowane są coraz to lepsze wyniki, ale w przypadku algorytmu genetycznego ta poprawa dzieje się na podstawie naturalnych procesów biologicznych, tj. mutacji, krzyżowania się genów osobników populacji oraz samej eugeniki populacji, gdyż każdą populację charakteryzuje to, iż mogą przetrwać tylko najsilniejsze osobniki, gdyż to właśnie one muszą się dostosować do warunków panujących w otoczeniu.

Podsumowując – algorytm genetyczny jest dosyć dobrym algorytmem znajdującym rozwiązania przybliżone w akceptowalnym czasie. Implementując go do różnych, bardziej złożonych problemów (jakim jest na przykład Problem Komiwojażera) musimy się liczyć z tym, że uzyskiwane wyniki w znakomitej większości przypadków **nie będą** rozwiązaniami optymalnymi.