Metody Inteligencji Obliczeniowej w Analizie Danych
Sprawozdanie AE

Filip Langiewicz
nr indeksu 327293

Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

10 czerwca 2025

Spis treści

1. Wstęp

Algorytmy ewolucyjne (AE) stanowią klasę metaheurystycznych metod optymalizacji inspirowanych mechanizmami biologicznej ewolucji, takimi jak selekcja naturalna, krzyżowanie i mutacja. Ich elastyczność oraz zdolność do przeszukiwania dużych i złożonych przestrzeni rozwiązań sprawiają, że są one skutecznym narzędziem w rozwiązywaniu problemów, dla których trudno jest sformułować klasyczne, deterministyczne podejście.

Celem laboratorium było praktyczne zapoznanie się z podstawowymi koncepcjami algorytmów genetycznych — jednej z najczęściej wykorzystywanych odmian AE — oraz zastosowanie ich do wybranych problemów optymalizacyjnych. Realizacja zadań obejmowała zarówno projektowanie struktury algorytmu, jak i implementację odpowiednich operatorów oraz testowanie działania na różnych typach funkcji i danych.

Podejście oparte na algorytmach ewolucyjnych pozwala nie tylko na eksplorację wielu potencjalnych rozwiązań jednocześnie, ale również na adaptacyjne dostosowywanie się do charakterystyki konkretnego problemu. Dzięki temu możliwe jest skuteczne rozwiązywanie zadań, w których występują lokalne minima, ograniczenia geometryczne czy wysokowymiarowe przestrzenie rozwiązań.

2. AE1: Implementacja algorytmu genetycznego

Pierwsza część laboratorium polegała na implementacji podstawowego algorytmu genetycznego, którego celem była optymalizacja wartości funkcji w przestrzeni $\mathbb{R}^n$. Zadanie miało na celu zapoznanie się z klasycznymi elementami algorytmu, takimi jak krzyżowanie jednopunktowe oraz mutacja gaussowska, i ich zastosowanie w praktyce.

Algorytm został przetestowany na dwóch funkcjach celu. Pierwszą z nich była prosta funkcja kwadratowa postaci

$$f(x, y, z) = x^2 + y^2 + 2z^2,$$

która posiada jedno minimum globalne i dobrze nadaje się do weryfikacji poprawności działania algorytmu. Drugim przypadkiem testowym była pięciowymiarowa funkcja Rastrigina, znana z dużej liczby minimów lokalnych. Jest to funkcja trudniejsza optymalizacyjnie, często wykorzystywana jako benchmark dla algorytmów ewolucyjnych, ponieważ pozwala ocenić zdolność algorytmu do unikania lokalnych minimów i znajdowania rozwiązania globalnego.

Celem zadania było zaprojektowanie i uruchomienie algorytmu w taki sposób, aby był on w stanie skutecznie znaleźć minimum zadanych funkcji, a tym samym zweryfikować skuteczność przyjętych operatorów genetycznych oraz przyjętej reprezentacji i parametrów ewolucji.

2.1. Opis wykonanej pracy

W ramach pierwszego zadania zaimplementowano podstawowy algorytm genetyczny służący do optymalizacji funkcji w przestrzeni rzeczywistej $\mathbb{R}^n$. Celem było przetestowanie jego skuteczności na dwóch funkcjach: klasycznej funkcji kwadratowej $f(x, y, z) = x^2 + y^2 + 2z^2$ oraz pięciowymiarowej funkcji Rastrigina, znanej z licznych lokalnych minimów i wykorzystywanej jako testowy benchmark dla algorytmów optymalizacyjnych.

Algorytm został zaimplementowany w języku `Python` i zawiera wszystkie podstawowe elementy algorytmu genetycznego:

- **Reprezentacja osobników**: każdy osobnik populacji to wektor rzeczywistych liczb o dłu-

gości równej liczbie wymiarów problemu.

- **Inicjalizacja populacji**: osobniki są losowo generowane w zadanym zakresie wartości (domyślnie $[-5.12, 5.12]$).
- **Ocena przystosowania**: dla każdego osobnika obliczana jest wartość funkcji celu.
- **Selekcja**: zastosowano selekcję turniejową (turniej z 3 osobników) do wyboru rodziców.
- **Krzyżowanie**: wykorzystano krzyżowanie jednopunktowe (ang. *one-point crossover*), zgodnie z wymaganiami zadania. Z prawdopodobieństwem określonym parametrem `crossover_rate` (domyślnie 0.9), potomkowie dziedziczą część genów od jednego rodzica i resztę od drugiego.
- **Mutacja**: mutacja gaussowska została zaimplementowana jako dodanie losowego zakłócenia z rozkładu normalnego do każdego genu z pewnym prawdopodobieństwem (`mutation_rate`). Wartość mutacji kontroluje parametr `mutation_std` (domyślnie 0.1).
- **Elityzm**: opcjonalnie najlepszy osobnik z danej generacji jest przenoszony do następnej bez zmian.

Podczas ewolucji algorytm rejestruje historię wartości funkcji przystosowania: najlepszej, średniej oraz odchylenia standardowego w każdej generacji. Pozwala to na wizualizację postępów w optymalizacji.

Zaimplementowano również funkcję `plot_fitness_progress()`, która umożliwia graficzne przedstawienie przebiegu ewolucji, co ułatwia ocenę skuteczności i stabilności algorytmu.

2.2. Eksperymenty i ich wyniki

2.2.1. Funkcja kwadratowa

Dla funkcji kwadratowej `simple_quadratic_function` przeprowadzono eksperymenty z różnymi kombinacjami parametrów algorytmu genetycznego. Testowane były następujące wartości:

- **Współczynnik krzyżowania (crossover_rate)**: 0.1, 0.5, 0.9
- **Współczynnik mutacji (mutation_rate)**: 0.1, 0.5, 0.9
- **Odchylenie standardowe mutacji (mutation_std)**: 0.1, 0.5, 1.0
- **Elitaryzm (elitism)**: True, False

Dla każdej możliwej kombinacji tych parametrów przeprowadzono osobne uruchomienie algorytmu na 100 pokoleniach, a następnie zapisano uzyskane najlepsze wartości funkcji celu.

Algorytm został skonfigurowany w sposób następujący:

- Rozmiar populacji: 50
- Wymiar przestrzeni: 3
- Granice przestrzeni: $[-5.12, 5.12]$
- Liczba pokoleń: 100

Uzyskane wartości funkcji celu (im mniejsze, tym lepsze) zestawiono w tabeli 2.1. Analiza wyników znajduje się pod tabelą.

ID	**Crossover Rate**	**Mutation Rate**	**Mutation Std**	**Elitism**	**Wynik [$\times 10^{-6}$]**
0	0.1	0.1	0.1	True	11.58119
1	0.1	0.1	0.1	False	4.362628
2	0.1	0.1	0.5	True	6.734678
3	0.1	0.1	0.5	False	33.85062
4	0.1	0.1	1.0	True	4.619116
5	0.1	0.1	1.0	False	0.3378969
6	0.1	0.5	0.1	True	51.32699
7	0.1	0.5	0.1	False	1115.835
8	0.1	0.5	0.5	True	119.3342
9	0.1	0.5	0.5	False	5582.888
10	0.1	0.5	1.0	True	1376.053
11	0.1	0.5	1.0	False	215.7386
12	0.1	0.9	0.1	True	117.333
13	0.1	0.9	0.1	False	1514.469
14	0.1	0.9	0.5	True	3715.322
15	0.1	0.9	0.5	False	81731.16
16	0.1	0.9	1.0	True	20204.91
17	0.1	0.9	1.0	False	222353.6
18	0.5	0.1	0.1	True	2.527753
19	0.5	0.1	0.1	False	1.836139

ID	Crossover Rate	Mutation Rate	Mutation Std	Elitism	Wynik [$\times 10^{-6}$]
20	0.5	0.1	0.5	True	49.96709
21	0.5	0.1	0.5	False	2.814215
22	0.5	0.1	1.0	True	302.6874
23	0.5	0.1	1.0	False	102.5989
24	0.5	0.5	0.1	True	69.15456
25	0.5	0.5	0.1	False	851.0775
26	0.5	0.5	0.5	True	593.4378
27	0.5	0.5	0.5	False	38397.87
28	0.5	0.5	1.0	True	921.0768
29	0.5	0.5	1.0	False	61348.16
30	0.5	0.9	0.1	True	116.6274
31	0.5	0.9	0.1	False	868.5737
32	0.5	0.9	0.5	True	1354.01
33	0.5	0.9	0.5	False	107130.7
34	0.5	0.9	1.0	True	10750.24
35	0.5	0.9	1.0	False	30392.93
36	0.9	0.1	0.1	True	0.143129
37	0.9	0.1	0.1	False	0.2366659
38	0.9	0.1	0.5	True	14.65252
39	0.9	0.1	0.5	False	47.14494
40	0.9	0.1	1.0	True	18.01236
41	0.9	0.1	1.0	False	22.17698
42	0.9	0.5	0.1	True	2.859598
43	0.9	0.5	0.1	False	795.3711
44	0.9	0.5	0.5	True	534.6016
45	0.9	0.5	0.5	False	67496.26
46	0.9	0.5	1.0	True	455.3582
47	0.9	0.5	1.0	False	6020.471
48	0.9	0.9	0.1	True	59.14241
49	0.9	0.9	0.1	False	1774.816
50	0.9	0.9	0.5	True	2986.853
51	0.9	0.9	0.5	False	45902.7

ID	Crossover Rate	Mutation Rate	Mutation Std	Elitism	Wynik [$\times 10^{-6}$]
52	0.9	0.9	1.0	True	13494.42
53	0.9	0.9	1.0	False	454003.4

Tabela 2.1: Wyniki dla różnych parametrów

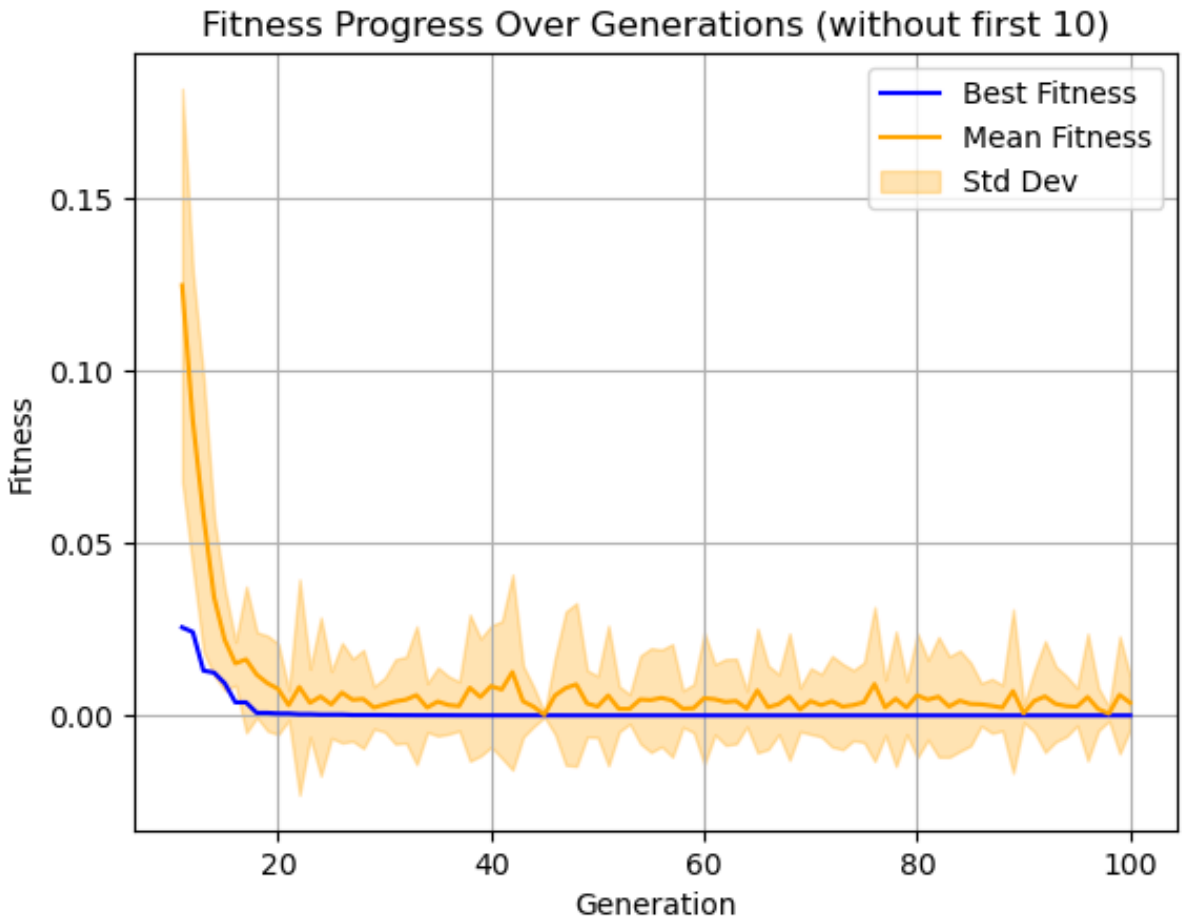
Na podstawie przedstawionych wyników przeprowadzono analizę skuteczności różnych konfiguracji algorytmu genetycznego. Oceniano wpływ parametrów takich jak stopień krzyżowania (*crossover rate*), stopień mutacji (*mutation rate*), odchylenie standardowe mutacji (*mutation std*) oraz zastosowanie elitaryzmu (*elitism*) na wartość funkcji celu.

Najlepsza implementacja

Najniższą wartość funkcji celu uzyskano dla konfiguracji o identyfikatorze **ID = 36**. Wartość funkcji celu wyniosła $0{,}143129 \cdot 10^{-6}$, co wskazuje na wysoką skuteczność tej konfiguracji. Parametry tej implementacji przedstawiają się następująco:

- **Crossover rate**: 0,9
- **Mutation rate**: 0,1
- **Mutation std**: 0,1
- **Elitism**: Tak

Konfiguracja ta charakteryzuje się wysokim współczynnikiem krzyżowania oraz niskimi parametrami mutacji, co sprzyja skutecznej eksploracji przestrzeni rozwiązań przy jednoczesnym zachowaniu stabilności optymalizacji. Włączenie elitaryzmu umożliwiło zachowanie najlepszych osobników, co dodatkowo zwiększyło efektywność algorytmu. Przebieg wyników algorytmu jest ukazany na rysunku 2.1.



Rysunek 2.1: Przebieg algorytmu dla najlepszej konfiguracji dla funkcji kwadratowej

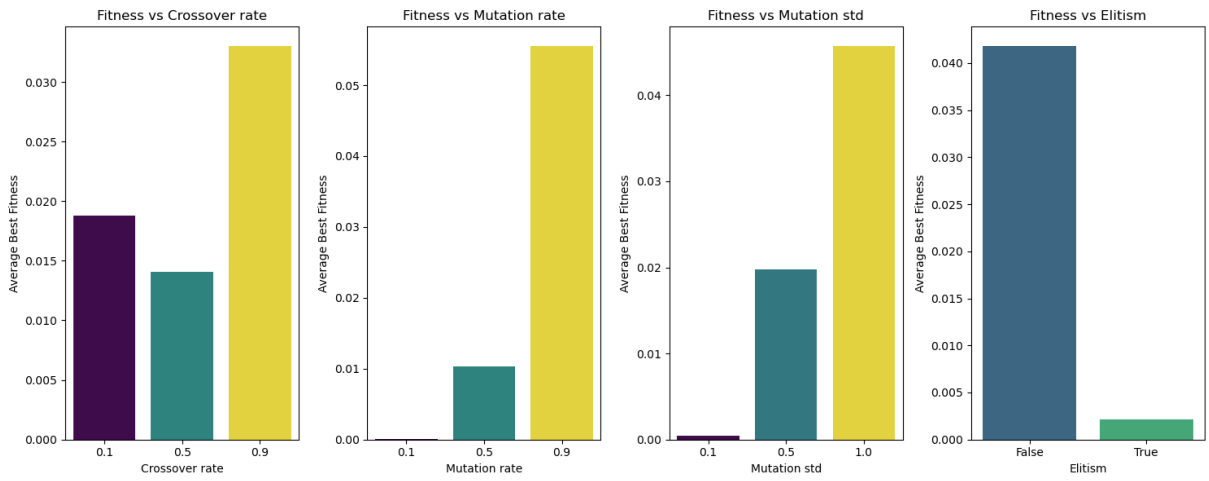
Najgorsza implementacja

Najgorszy wynik uzyskano dla konfiguracji o identyfikatorze **ID = 53**. Wartość funkcji celu wyniosła $454003{,}4 \cdot 10^{-6}$, co jest ponad trzy miliony razy gorszym wynikiem niż w przypadku najlepszej implementacji. Parametry tej konfiguracji były następujące:

- **Crossover rate**: 0,9
- **Mutation rate**: 0,9
- **Mutation std**: 1,0
- **Elitism**: Nie

Choć współczynnik krzyżowania był wysoki, bardzo intensywna mutacja w połączeniu z jej dużym odchyleniem standardowym oraz brakiem elitaryzmu doprowadziły do znacznego pogorszenia wyników. Nadmierna losowość oraz brak mechanizmu zachowującego najlepsze rozwiązania skutkowały destabilizacją procesu optymalizacji.

Na rysunku 2.2 znajduje się wykres agregujący uzyskane wyniki. Na podstawie analizy wykresu można wyciągnąć kilka istotnych wniosków dotyczących wpływu parametrów algorytmu ewolucyjnego. W przypadku współczynnika krzyżowania (crossover rate) najlepsze rezultaty osiągnięto przy wartości 0.5, natomiast najgorsze przy 0.9, co sugeruje, że zbyt intensywne krzyżowanie może pogarszać jakość rozwiązań. Dla współczynnika mutacji (mutation rate) oraz odchylenia standardowego mutacji (mutation std) najniższe wartości fitness zaobserwowano przy wartościach 0.1, co wskazuje, że nadmierna mutacja również negatywnie wpływa na efektywność algorytmu. Najbardziej zauważalny wpływ ma parametr elitaryzmu – włączenie elitaryzmu (True) prowadzi do znacznie lepszych wyników niż jego brak, co potwierdza, że zachowywanie najlepszych osobników z poprzednich pokoleń poprawia skuteczność procesu optymalizacji. Podsumowując, najlepsze rezultaty osiągane są przy umiarkowanym krzyżowaniu, niskim poziomie mutacji oraz przy zastosowaniu elitaryzmu, co podkreśla znaczenie równowagi między eksploracją a eksploatacją w algorytmie ewolucyjnym.



Rysunek 2.2: Podsumowanie wyników fitness w zależności od poszczególnych parametrów dla funkcji kwadratowej

2.2.2. Funkcja Rastrigina

Dla pięciowymiarowej funkcji Rastrigina przeprowadzono eksperymenty z różnymi kombinacjami parametrów algorytmu genetycznego. Testowane były następujące wartości:

- **Współczynnik krzyżowania (crossover_rate)**: 0.1, 0.5, 0.9
- **Współczynnik mutacji (mutation_rate)**: 0.1, 0.5, 0.9
- **Odchylenie standardowe mutacji (mutation_std)**: 0.1, 0.5, 1.0
- **Elitaryzm (elitism)**: True, False

Dla każdej możliwej kombinacji tych parametrów przeprowadzono osobne uruchomienie algorytmu na 200 pokoleniach, a następnie zapisano uzyskane najlepsze wartości funkcji celu.

Algorytm został skonfigurowany w sposób następujący:

- Rozmiar populacji: 100
- Wymiar przestrzeni: 5
- Granice przestrzeni: $[-5.12, 5.12]$
- Liczba pokoleń: 200

Uzyskane wartości funkcji celu (im mniejsze, tym lepsze) zestawiono w tabeli 2.2. Analiza wyników znajduje się pod tabelą.

ID	**Crossover Rate**	**Mutation Rate**	**Mutation Std**	**Elitism**	**Wynik**
0	0.1	0.1	0.1	True	5.969884
1	0.1	0.1	0.1	False	5.969811
2	0.1	0.1	0.5	True	0.000220
3	0.1	0.1	0.5	False	0.002139
4	0.1	0.1	1.0	True	0.000858
5	0.1	0.1	1.0	False	0.013821
6	0.1	0.5	0.1	True	6.976933
7	0.1	0.5	0.1	False	2.761284
8	0.1	0.5	0.5	True	0.933825
9	0.1	0.5	0.5	False	6.679431
10	0.1	0.5	1.0	True	4.850527
11	0.1	0.5	1.0	False	22.678524
12	0.1	0.9	0.1	True	13.169957
13	0.1	0.9	0.1	False	6.639949
14	0.1	0.9	0.5	True	4.840986
15	0.1	0.9	0.5	False	11.390553
16	0.1	0.9	1.0	True	5.375057
17	0.1	0.9	1.0	False	20.387391
18	0.5	0.1	0.1	True	4.974797
19	0.5	0.1	0.1	False	3.979876

ID	Crossover Rate	Mutation Rate	Mutation Std	Elitism	Wynik
20	0.5	0.1	0.5	True	0.000584
21	0.5	0.1	0.5	False	0.000316
22	0.5	0.1	1.0	True	0.001600
23	0.5	0.1	1.0	False	0.006349
24	0.5	0.5	0.1	True	0.999821
25	0.5	0.5	0.1	False	3.087334
26	0.5	0.5	0.5	True	1.256846
27	0.5	0.5	0.5	False	15.143483
28	0.5	0.5	1.0	True	1.645807
29	0.5	0.5	1.0	False	7.965414
30	0.5	0.9	0.1	True	3.153290
31	0.5	0.9	0.1	False	5.036378
32	0.5	0.9	0.5	True	6.077476
33	0.5	0.9	0.5	False	13.147771
34	0.5	0.9	1.0	True	8.768176
35	0.5	0.9	1.0	False	33.671284
36	0.9	0.1	0.1	True	2.984905
37	0.9	0.1	0.1	False	1.990028
38	0.9	0.1	0.5	True	0.000587
39	0.9	0.1	0.5	False	0.000933
40	0.9	0.1	1.0	True	0.001132
41	0.9	0.1	1.0	False	0.001562
42	0.9	0.5	0.1	True	4.985275
43	0.9	0.5	0.1	False	2.753935
44	0.9	0.5	0.5	True	1.791237
45	0.9	0.5	0.5	False	16.291262
46	0.9	0.5	1.0	True	2.155001
47	0.9	0.5	1.0	False	20.087567
48	0.9	0.9	0.1	True	2.201775
49	0.9	0.9	0.1	False	2.864325
50	0.9	0.9	0.5	True	5.428632
51	0.9	0.9	0.5	False	16.288318

ID	Crossover Rate	Mutation Rate	Mutation Std	Elitism	Wynik
52	0.9	0.9	1.0	True	7.388179
53	0.9	0.9	1.0	False	30.426025

Tabela 2.2: Wyniki dla różnych parametrów

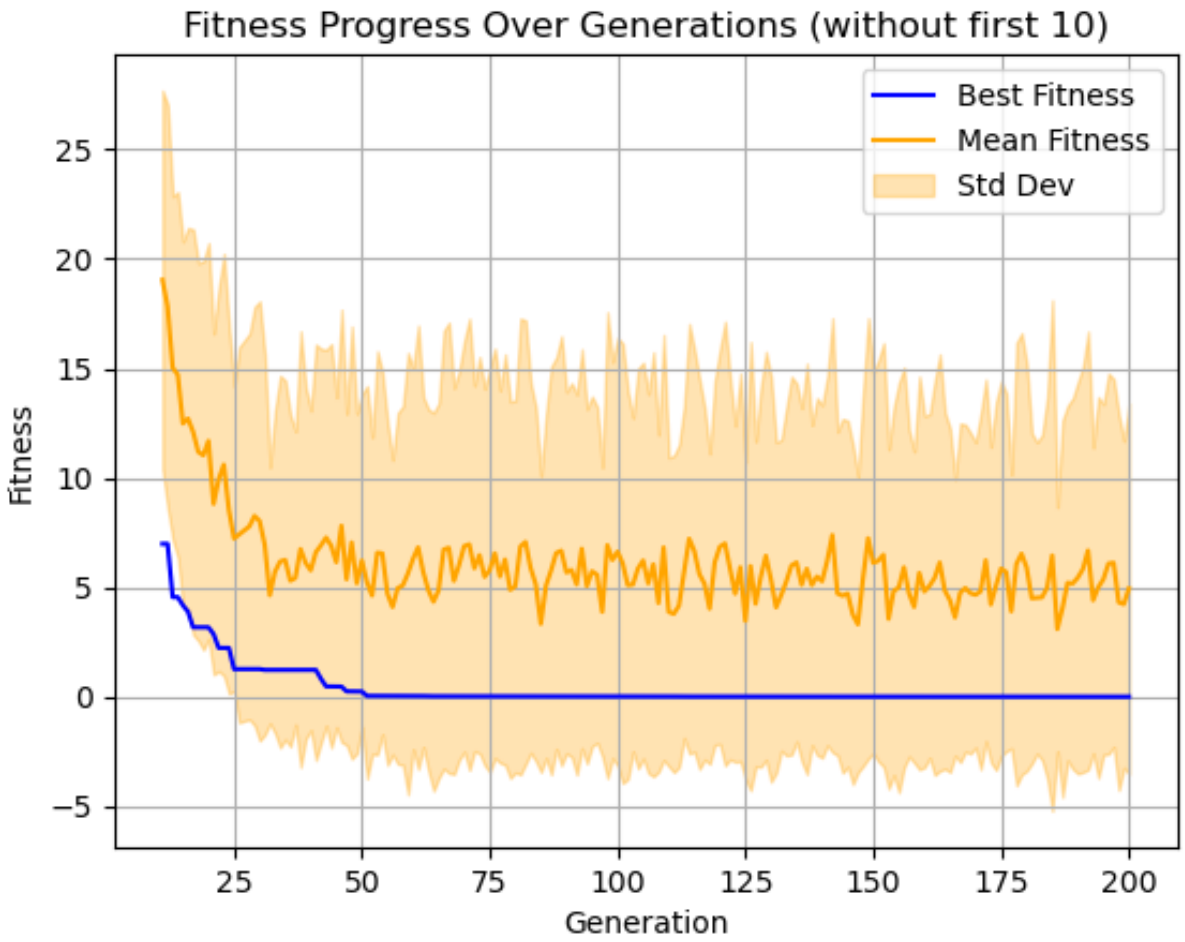
Na podstawie przedstawionych wyników przeprowadzono analizę skuteczności różnych konfiguracji algorytmu genetycznego. Oceniano wpływ parametrów takich jak stopień krzyżowania (*crossover rate*), stopień mutacji (*mutation rate*), odchylenie standardowe mutacji (*mutation std*) oraz zastosowanie elitaryzmu (*elitism*) na wartość funkcji celu.

Najlepsza implementacja

Najniższą wartość funkcji celu uzyskano dla konfiguracji o identyfikatorze **ID = 2**. Wartość funkcji celu wyniosła 0,000220, co wskazuje na wysoką skuteczność tej konfiguracji. Parametry tej implementacji przedstawiają się następująco:

- **Crossover rate**: 0,1
- **Mutation rate**: 0,1
- **Mutation std**: 0,5
- **Elitism**: Tak

Konfiguracja ta charakteryzuje się niskim współczynnikiem krzyżowania oraz niskimi parametrami mutacji, co sprzyja skutecznej eksploracji przestrzeni rozwiązań przy jednoczesnym zachowaniu stabilności optymalizacji. Włączenie elitaryzmu umożliwiło zachowanie najlepszych osobników, co dodatkowo zwiększyło efektywność algorytmu. Przebieg wyników algorytmu jest ukazany na rysunku 2.3.



Rysunek 2.3: Przebieg algorytmu dla najlepszej konfiguracji dla funkcji Rastrigina

Najgorsza implementacja

Najgorszy wynik uzyskano dla konfiguracji o identyfikatorze **ID = 35**. Wartość funkcji celu wyniosła 33,6713, co jest dużo gorszym wynikiem niż w przypadku najlepszej implementacji. Parametry tej konfiguracji były następujące:

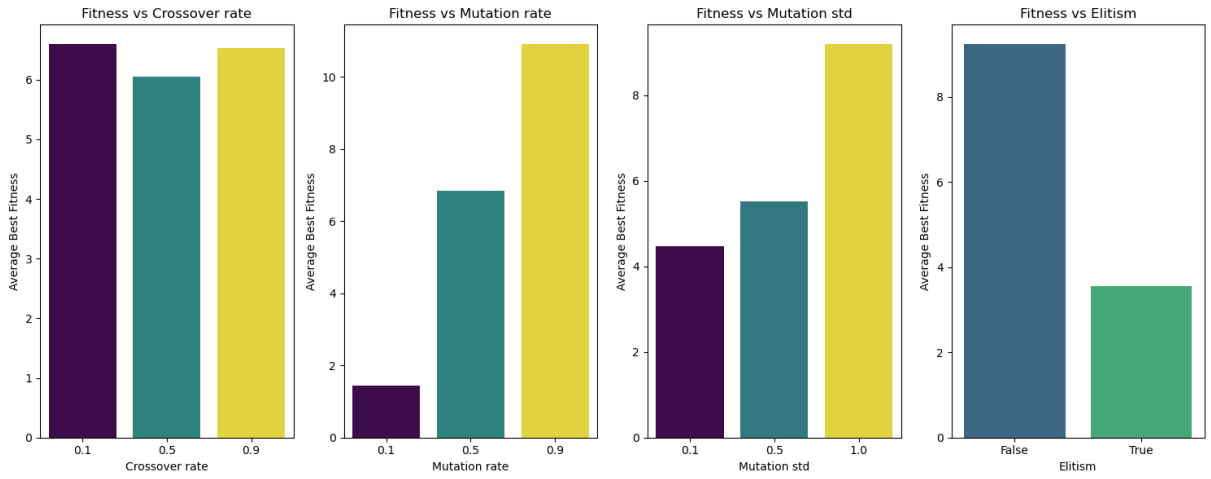
- **Crossover rate**: 0,5
- **Mutation rate**: 0,9
- **Mutation std**: 1,0
- **Elitism**: Nie

Choć współczynnik krzyżowania był wysoki, bardzo intensywna mutacja w połączeniu z jej dużym odchyleniem standardowym oraz brakiem elitaryzmu doprowadziły do znacznego pogorszenia wyników. Nadmierna losowość oraz brak mechanizmu zachowującego najlepsze rozwiązania skutkowały destabilizacją procesu optymalizacji.

Na rysunku 2.4 znajduje się wykres agregujący uzyskane wyniki. Analiza przedstawionych wykresów słupkowych ukazuje wpływ różnych parametrów algorytmu genetycznego na średnią najlepszą wartość funkcji przystosowania (fitness), przy czym należy pamiętać, że im niższa ta wartość, tym lepszy wynik optymalizacji. W przypadku współczynnika krzyżowania (crossover rate) najlepsze rezultaty osiągnięto dla wartości 0.5, co sugeruje, że umiarkowany poziom krzyżowania sprzyja efektywnemu przeszukiwaniu przestrzeni rozwiązań. Zarówno zbyt niski (0.1), jak i zbyt wysoki (0.9) poziom krzyżowania prowadził do pogorszenia wyników.

Dla współczynnika mutacji (mutation rate) najniższa średnia fitness została uzyskana przy wartości 0.1, natomiast wyższe wartości tego parametru (0.5 i 0.9) znacząco pogarszały jakość rozwiązań. Sugeruje to, że nadmierna losowość wprowadzana przez mutację może zakłócać proces stopniowego doskonalenia populacji. Podobny trend zaobserwowano w przypadku odchylenia standardowego mutacji (mutation std), gdzie najniższe wartości fitness wystąpiły przy najmniejszym testowanym odchyleniu (0.1), a ich wzrost powodował pogorszenie wyników, co również potwierdza negatywny wpływ zbyt dużej losowości.

Analiza wpływu elityzmu wskazuje, że jego zastosowanie (wartość True) skutkowało wyraźnie lepszymi wynikami niż w przypadku jego braku. Taka konfiguracja sprzyjała uzyskaniu najniższych wartości funkcji celu, a więc najbardziej efektywnej optymalizacji.



Rysunek 2.4: Podsumowanie wyników fitness w zależności od poszczególnych parametrów dla funkcji Rastrigina

2.3. Wnioski

Na podstawie przeprowadzonych eksperymentów można stwierdzić, że skuteczność algorytmu genetycznego silnie zależy od doboru parametrów ewolucji oraz typu funkcji celu. W przypadku

funkcji kwadratowej, będącej prostym problemem optymalizacyjnym, większość konfiguracji pozwalała na znalezienie wartości bliskich minimum globalnego, jednak zauważalne różnice pomiędzy najlepszymi i najgorszymi wynikami podkreślają znaczenie odpowiedniego doboru parametrów. Szczególnie istotne okazało się zastosowanie elitaryzmu, który znacząco poprawiał stabilność i efektywność procesu ewolucyjnego, chroniąc najlepsze osobniki przed utratą.

Dla bardziej wymagającej funkcji Rastrigina różnice między konfiguracjami były jeszcze wyraźniejsze. Intensywne mutacje oraz brak elitaryzmu prowadziły do znacznego pogorszenia wyników, co świadczy o destrukcyjnym wpływie nadmiernej losowości i braku mechanizmu zachowującego jakość rozwiązań. Z kolei najlepsze rezultaty uzyskiwano przy niskim poziomie mutacji i umiarkowanym współczynniku krzyżowania, co sprzyjało równowadze pomiędzy eksploracją przestrzeni rozwiązań a eksploatacją dotychczas uzyskanych wyników.

Podsumowując, wyniki eksperymentów potwierdzają, że odpowiednio zaprojektowany algorytm genetyczny może być skutecznym narzędziem do optymalizacji zarówno prostych, jak i złożonych funkcji. Kluczowe znaczenie mają jednak takie aspekty, jak umiejętny dobór parametrów, obecność elitaryzmu oraz kontrola intensywności działań losowych, które wpływają na zdolność algorytmu do unikania pułapek lokalnych minimów i skutecznego dążenia do optimum globalnego.

3. AE2: Wypełnianie koła prostokątami

3.1. Opis tematu

Drugie zadanie laboratorium stanowiło wariant problemu znanego w literaturze jako *cutting stock problem*, w którym celem jest optymalne rozmieszczenie elementów o zadanych wymiarach i wartościach w ograniczonej przestrzeni. W tym przypadku przestrzenią do zagospodarowania było koło o zadanym promieniu, natomiast elementami były prostokąty opisane przez trzy parametry: szerokość, wysokość oraz wartość.

Celem optymalizacji było takie rozmieszczenie prostokątów wewnątrz koła, aby zmaksymalizować łączną sumę ich wartości, przy zachowaniu kilku istotnych ograniczeń:

- boki prostokątów muszą być równoległe do osi układu współrzędnych,
- prostokąty nie mogą nachodzić na siebie (mogą się stykać bokami),
- każdy prostokąt może być użyty wielokrotnie.

Zadanie wymagało opracowania odpowiedniego sposobu kodowania rozwiązań, a także zaprojektowania operatorów mutacji i krzyżowania dostosowanych do specyfiki problemu geometrycznego. Ze względu na złożoność przestrzeni rozwiązań oraz brak jednoznacznego sposobu obliczania optymalnego ułożenia, zastosowano algorytm genetyczny jako metodę poszukiwania rozwiązań przybliżonych.

Rozwiązanie zostało przetestowane na zestawie danych wejściowych zawierających różne zbiory prostokątów oraz promienie kół. Dane dostarczono w plikach CSV, gdzie każda linia opisywała pojedynczy prostokąt. W celu uzyskania pełnej punktacji należało osiągnąć określone progi sumarycznej wartości prostokątów w kilku przypadkach testowych.

3.2. Opis wykonanej pracy

3.2.1. Reprezentacja rozwiązania

Podstawowym elementem rozwiązania jest klasa `Individual`, reprezentująca pojedynczą konfigurację ułożenia prostokątów. Każdy prostokąt reprezentowany jest przez klasę `Rectangle`, która przechowuje informacje o współrzędnych środka prostokąta, jego szerokości, wysokości oraz przypisanej wartości. Prostokąty są pozycjonowane w dwuwymiarowym układzie współrzędnych tak, aby ich boki były równoległe do osi układu oraz aby mieściły się całkowicie wewnątrz koła o zadanym promieniu.

3.2.2. Sprawdzanie poprawności rozwiązań

Implementacja zawiera mechanizmy zapewniające poprawność układu prostokątów. Metoda `inside_circle` sprawdza, czy wszystkie wierzchołki prostokąta mieszczą się w obszarze koła, natomiast metoda `overlaps_with` wykrywa nakładanie się prostokątów poprzez analizę ich granic. Dzięki temu zapewnione jest, że prostokąty nie nachodzą na siebie oraz nie wychodzą poza granice koła.

3.2.3. Operatory genetyczne

Do ewolucji populacji zastosowano klasyczne operatory genetyczne, dostosowane do specyfiki problemu:

- **Mutacja** — implementowana jest za pomocą kilku strategii, w tym:
  - dodawanie nowych losowych prostokątów w obszarze koła,
  - przesuwanie istniejących prostokątów z zachowaniem warunków poprawności,
  - zastępowanie prostokątów nowymi, losowymi,
  - „ślizganie" prostokątów po przekątnej w granicach koła, aż do osiągnięcia maksymalnej możliwej pozycji.
- **Krzyżowanie** — polega na podziale przestrzeni prostokątów wzdłuż losowo wybranej osi (x lub y) na dwie części oraz wymianie odpowiednich fragmentów między dwoma osobnikami. Nowe prostokąty dodawane są tylko wtedy, gdy nie naruszają warunków poprawności układu.

3.2.4. Inicjalizacja i ewolucja populacji

Populacja początkowa jest generowana poprzez wielokrotne losowe umieszczanie prostokątów w kole, z dodatkową próbą ich przesuwania do lewego górnego narożnika w celu zagęszczenia ułożenia. Algorytm następnie przeprowadza proces ewolucji przez zadaną liczbę pokoleń, wykorzystując selekcję proporcjonalną do wartości dopasowania (sumy wartości prostokątów), operatory krzyżowania i mutacji oraz zachowując najlepszą część populacji (elitę) bez zmian.

3.2.5. Wizualizacja i analiza wyników

Efektem działania algorytmu jest najlepsze znalezione rozwiązanie, które można zwizualizować za pomocą graficznej reprezentacji prostokątów ułożonych w kole wraz z ich wartościami. Dodatkowo, algorytm generuje wykres prezentujący zmiany najlepszego dopasowania na przestrzeni kolejnych generacji, co pozwala ocenić postęp i stabilność procesu optymalizacji.

—

Powyższa implementacja stanowi efektywne narzędzie do rozwiązywania problemu wypełniania koła prostokątami z wielokrotnym użyciem elementów, umożliwiając osiągnięcie wyników spełniających podane wymagania minimalne.

3.3. Eksperymenty i ich wyniki

W trakcie realizacji zadania przeprowadzono szereg eksperymentów mających na celu uzyskanie jak najlepszych wartości funkcji celu dla problemu umieszczania prostokątów w kole o zadanym promieniu. W wyniku wielu prób wprowadzono dwie istotne modyfikacje algorytmu, które okazały się kluczowe dla osiągnięcia wymaganych rezultatów.

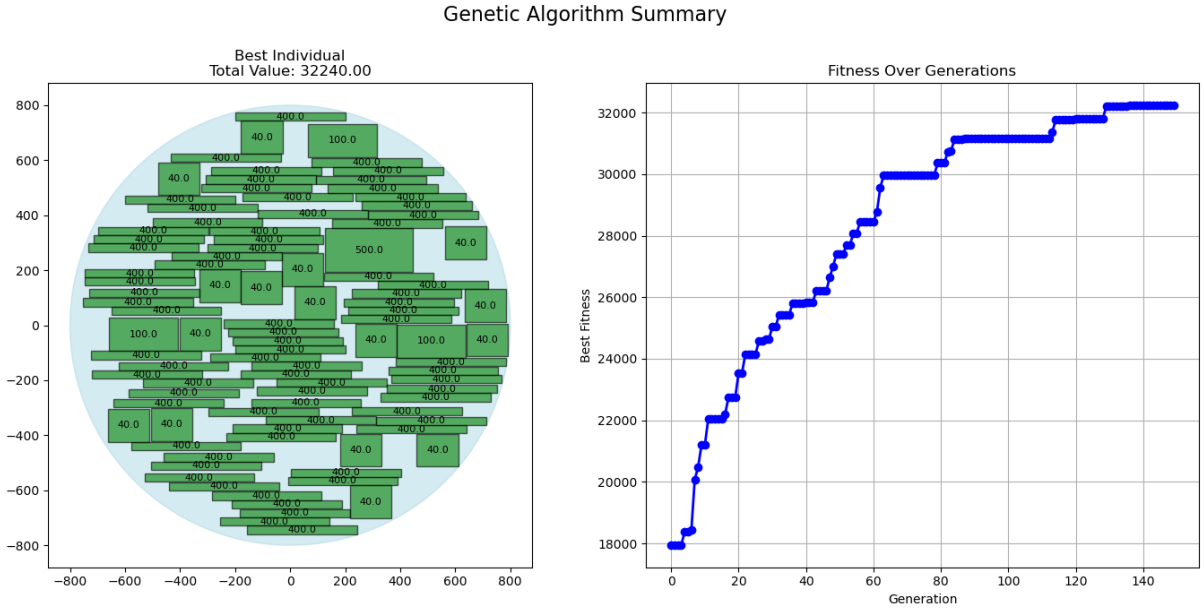
Pierwszym usprawnieniem było dodanie mechanizmu przesuwania prostokątów podczas ich wstawiania do układu, tak aby znajdowały się one możliwie najbliżej lewego górnego narożnika koła. Strategia ta pozwala na maksymalizację wykorzystania dostępnej przestrzeni przez eliminację zbędnych „luk" między prostokątami, co w efekcie poprawia ogólną wartość funkcji celu.

Drugim, bardziej zaawansowanym usprawnieniem, było wprowadzenie mechanizmu losowego doboru prostokątów podczas inicjalizacji populacji, z preferencją dla tych o wyższych wagach (wartościach). Dzięki temu algorytm skuteczniej koncentruje się na umieszczaniu prostokątów o większej wartości w obszarze koła, co znacząco zwiększa sumaryczną wartość rozwiązania. Kombinacja obu tych technik – przesuwania prostokątów oraz ważonego losowania – pozwoliła na uzyskanie wyników spełniających, a często znacznie przekraczających, wymagania zawarte

w poleceniu.

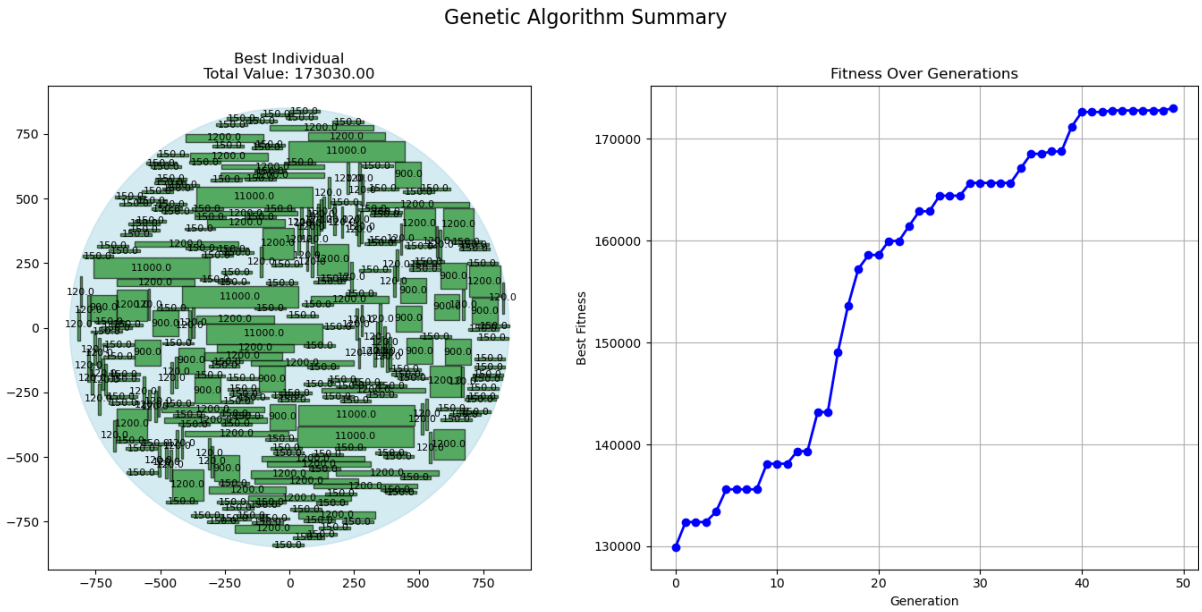
3.3.1. Wyniki eksperymentów dla poszczególnych zestawów danych

Promień 800 Dla koła o promieniu 800 podstawowa wersja algorytmu bez dodatkowych usprawnień była w stanie osiągnąć wymaganą wartość funkcji celu. W kolejnych pokoleniach obserwowano stopniowy wzrost najlepszej wartości, która finalnie ustabilizowała się na poziomie 32 240. Dodatkowe mechanizmy przesuwania prostokątów i ważonego doboru nie były konieczne w tym przypadku.



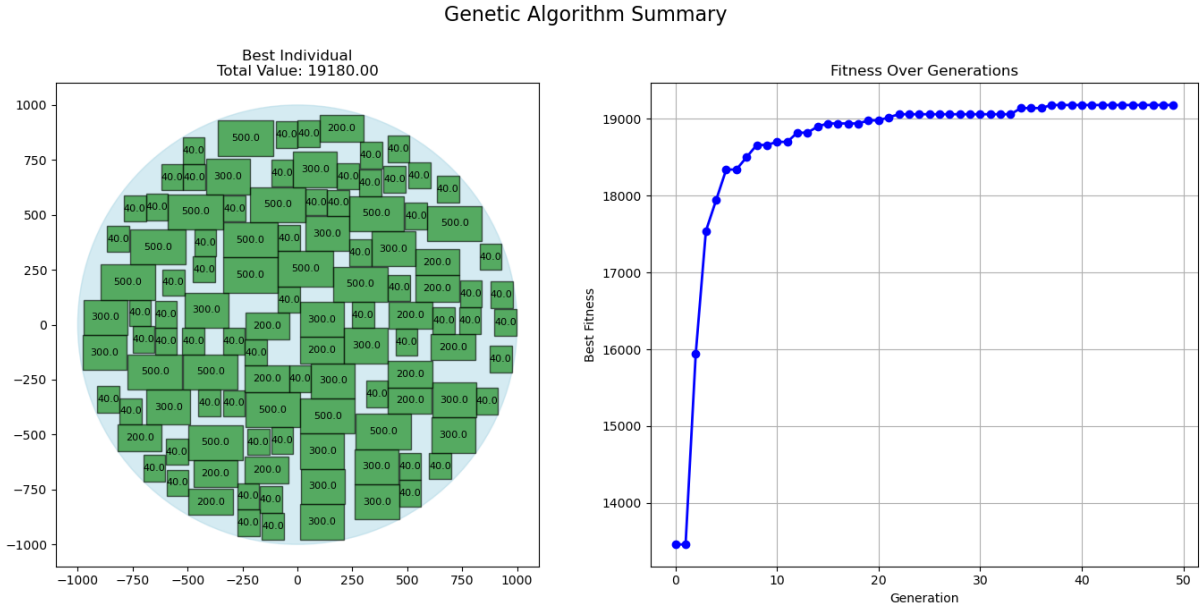
Rysunek 3.1: Finalny wynik dla koła o promieniu 800

Promień 850 W przypadku koła o promieniu 850 uzyskano wysokie wartości funkcji celu (końcowa wartość około 173 000). Jednakże, ze względu na brak szczegółowych wymagań dotyczących tej konfiguracji, nie prowadzono dalszej optymalizacji ani testów. Wynik ten został zaprezentowany jako przykład działania algorytmu na większej instancji problemu.



Rysunek 3.2: Finalny wynik dla koła o promieniu 850

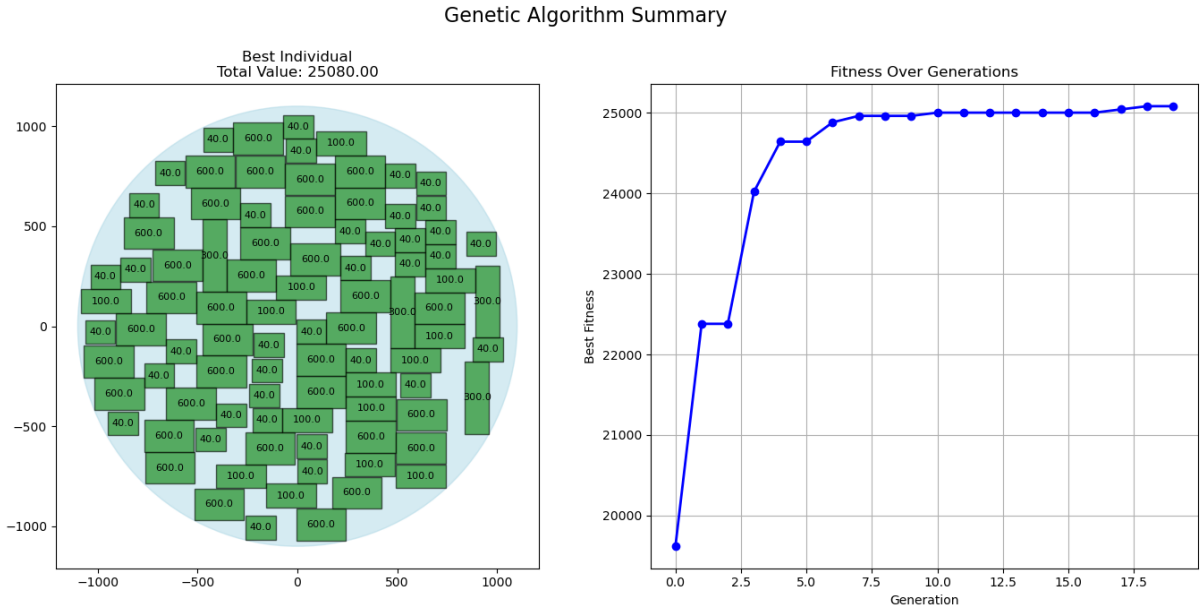
Promień 1000 Dla promienia 1000 podstawowy algorytm bez przesuwania prostokątów i ważonego doboru prostokątów nie przekraczał wymaganej wartości 17 500. Dopiero wprowadzenie mechanizmu przesuwania prostokątów w górę i w lewo pozwoliło na osiągnięcie wartości około 19 180, co jest satysfakcjonujące.



Rysunek 3.3: Finalny wynik dla koła o promieniu 1000

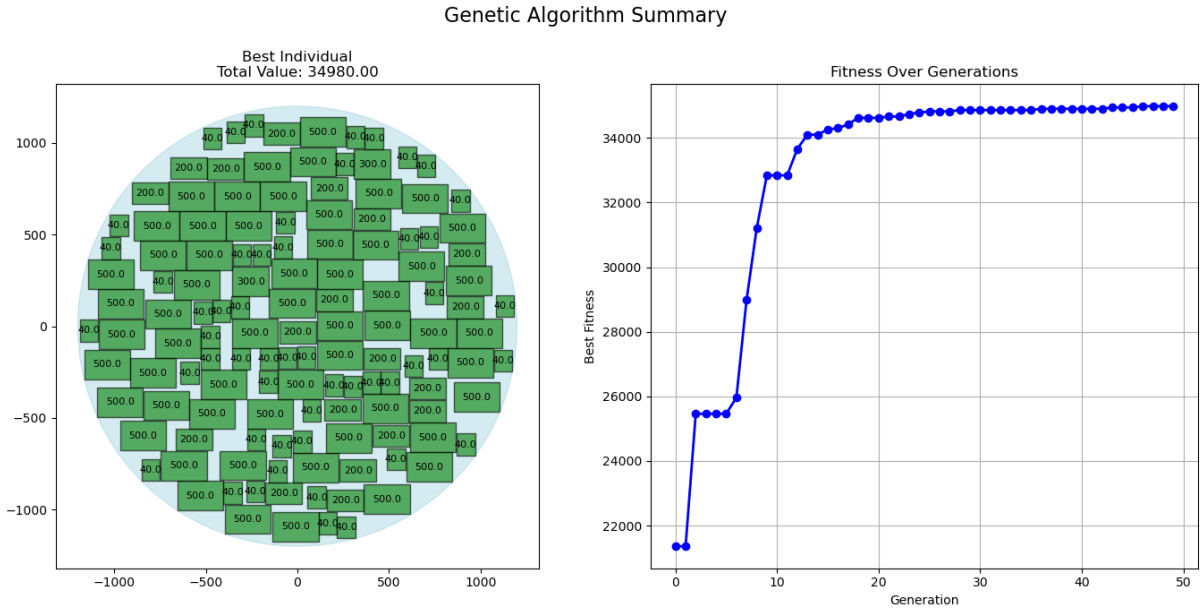
Promień 1100 Dla koła o promieniu 1100 zastosowanie przesuwania prostokątów wraz z ważonym doborem prostokątów pozwoliło na wyraźne zwiększenie wartości funkcji celu. Najlepszy

wynik osiągnięty po 20 pokoleniach wyniósł 25 080, co potwierdza skuteczność obu mechanizmów i pozwala na zrealizowanie wymagań zadania w tym przypadku.



Rysunek 3.4: Finalny wynik dla koła o promieniu 1100

Promień 1200 W największym rozpatrywanym przypadku (promień 1200) również zastosowanie obu usprawnień zaowocowało wartościami funkcji celu znacznie przekraczającymi wymagania. Po 50 pokoleniach najlepszy wynik osiągnął poziom około 34 980. Uzyskane rezultaty potwierdzają, że algorytm jest skalowalny i efektywny również na większych instancjach problemu.



Rysunek 3.5: Finalny wynik dla koła o promieniu 1200

3.4. Wnioski

Podsumowując, kluczowym elementem osiągnięcia wysokiej jakości rozwiązań było zastosowanie kombinacji dwóch usprawnień: przesuwania prostokątów do lewego górnego narożnika oraz ważonego losowania prostokątów, które preferuje te o większej wartości. Bez tych modyfikacji standardowy algorytm genetyczny nie był w stanie sprostać wymaganiom w problemach o większych promieniach koła.

W każdym z rozpatrywanych przypadków, poza promieniem 850, gdzie wynik nie był przedmiotem dalszej optymalizacji, uzyskano co najmniej wymaganą wartość funkcji celu, a często rezultaty znacznie ją przekroczyły. Eksperymenty potwierdzają poprawność i efektywność opracowanego rozwiązania.

Ze względu na charakter zadania, nie prowadzono szczegółowych testów, koncentrując się na maksymalizacji funkcji celu. Podejście to pozwoliło na skupienie się na dopracowaniu algorytmu i osiągnięciu optymalnych wyników w zadanych warunkach.

4. AE3: Optymalizacja wag w sieci MLP z użyciem algorytmu genetycznego

4.1. Opis tematu

Trzecia część laboratorium koncentrowała się na zastosowaniu algorytmu genetycznego do optymalizacji parametrów sieci neuronowej typu MLP (ang. *Multi-Layer Perceptron*). W odróżnieniu od klasycznych metod bazujących na propagacji wstecznej i pochodnych funkcji błędu, w tym przypadku do wyznaczania wartości wag wykorzystano mechanizmy ewolucyjne.

Algorytm genetyczny był wykorzystywany do minimalizacji błędu predykcji sieci na danych uczących. Każdy osobnik w populacji reprezentował kompletny zestaw wag sieci neuronowej. Proces optymalizacji obejmował standardowe operatory, takie jak krzyżowanie (np. jednopunktowe lub arytmetyczne) oraz mutacja (np. losowa zmiana wartości wag). Funkcja oceny mierzyła jakość rozwiązań poprzez obliczenie błędu sieci na zbiorze treningowym.

Uczenie zostało przeprowadzone na trzech zbiorach danych:

- **Iris** — klasyfikacja gatunków kwiatów na podstawie cech morfologicznych,
- **multimodal-large** — zbiór wykorzystywany podczas wcześniejszych ćwiczeń z sieci neuronowych,
- **Auto MPG** — regresyjny zbiór danych, w którym przewidywana była wartość zużycia paliwa (mpg) na podstawie parametrów technicznych pojazdów.

Zadanie miało na celu zarówno stworzenie poprawnie działającego systemu optymalizacji, jak i ocenę skuteczności algorytmu genetycznego jako alternatywnego podejścia do klasycznego uczenia sieci neuronowych.

4.2. Opis wykonanej pracy

W ramach realizacji tego etapu rozszerzono wcześniej stworzoną implementację klasycznej sieci neuronowej typu MLP (Multilayer Perceptron) o możliwość treningu z wykorzystaniem

algorytmu genetycznego. Celem tego podejścia było zastosowanie metod ewolucyjnych jako alternatywy dla klasycznego uczenia sieci z wykorzystaniem algorytmów gradientowych, takich jak backpropagation. Rozwiązanie to pozwala na przeprowadzenie procesu optymalizacji wag i biasów sieci neuronowej bez konieczności wyznaczania pochodnych, co czyni je odpornym na problemy lokalnych minimów i zastoje w nauce charakterystyczne dla metod opartych na gradiencie.

Każda warstwa sieci, zgodnie z przyjętym projektem, przechowuje własną macierz wag oraz wektor biasów. W celu umożliwienia optymalizacji genetycznej, wszystkie te parametry zostały spłaszczone i połączone w jeden wektor, który pełni rolę chromosomu — reprezentacji osobnika w populacji algorytmu genetycznego. Na początku działania algorytmu tworzona jest początkowa populacja takich chromosomów, która może być zainicjalizowana losowo lub na podstawie istniejących wag z klasycznego treningu (z dodanym szumem).

Funkcją dopasowania (fitness) w tym podejściu jest funkcja błędu wyliczana na podstawie działania sieci z określonym zestawem wag. Dla zadań klasyfikacyjnych wykorzystano funkcję kosztu cross-entropy, natomiast w przypadku regresji — błąd średniokwadratowy (MSE). W każdej generacji populacja jest oceniana, a najlepiej przystosowane osobniki (czyli te, których zestawy wag prowadzą do najmniejszego błędu) są selekcjonowane i częściowo kopiowane bez zmian do następnego pokolenia (tzw. elity). Pozostali osobnicy są tworzeni poprzez operacje krzyżowania (crossover) oraz mutacji, przy czym zastosowano jednopunktowe krzyżowanie i mutację gaussowską o określonym prawdopodobieństwie.

Proces ten jest iterowany przez ustaloną liczbę generacji. W każdej iteracji zapisywana jest najlepsza wartość funkcji błędu, co umożliwia późniejszą analizę zbieżności algorytmu. Po zakończeniu uczenia, najlepszy znaleziony zestaw wag zostaje przypisany do modelu, a przebieg uczenia jest wizualizowany za pomocą wykresu przedstawiającego zmniejszanie się błędu w kolejnych generacjach.

4.3. Eksperymenty i ich wyniki

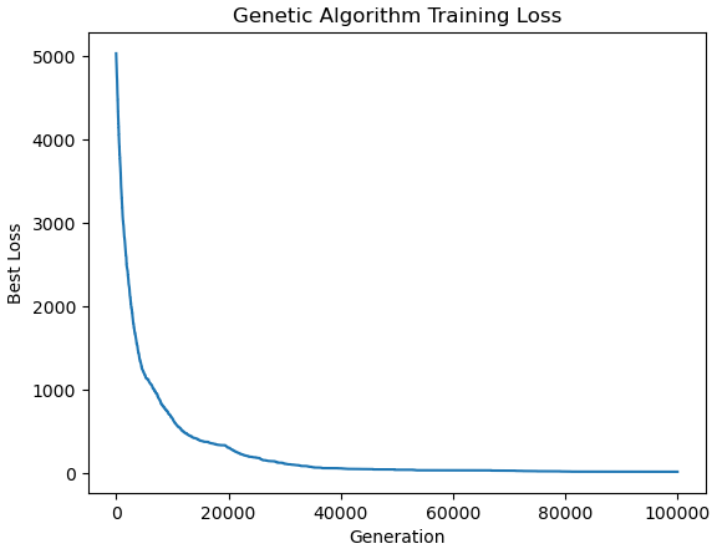
W celu oceny skuteczności zaprojektowanej sieci MLP uczonej za pomocą algorytmu genetycznego, przeprowadzono serię eksperymentów na trzech zróżnicowanych zbiorach danych: Multimodal-large, Auto-MPG oraz Iris. Celem testów było sprawdzenie, jak dobrze opracowany model radzi sobie zarówno z problemami regresyjnymi, jak i klasyfikacyjnymi, przy jednoczesnym uniknięciu użycia metod gradientowych.

Zbiór: Multimodal-large

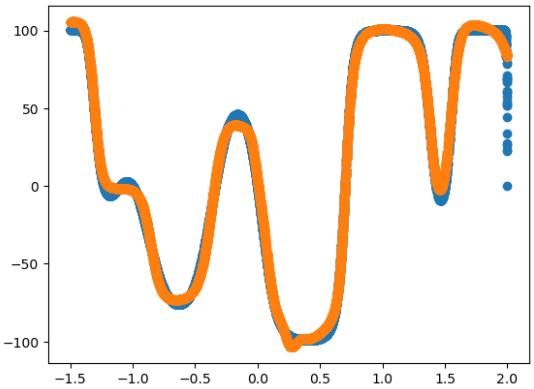
Pierwszy test został przeprowadzony na sztucznym zbiorze danych zawierającym jednowy-

miarowe wejścia oraz wartości wyjściowe, generowane z nieliniowej funkcji o wielu ekstremach lokalnych. Sieć została zaprojektowana jako dwuwarstwowy MLP z 29 neuronami aktywacji typu tangens hiperboliczny w warstwie ukrytej oraz liniową funkcją aktywacji w warstwie wyjściowej.

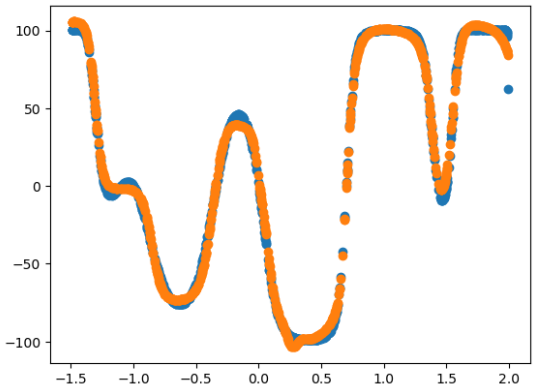
Trening przeprowadzono z populacją o bardzo małym rozmiarze (3 osobniki) i dużą liczbą generacji (100 000), co miało na celu sprawdzenie możliwości eksploracyjnych algorytmu w ekstremalnych warunkach. Mimo bardzo ograniczonej populacji, sieć zdołała stopniowo poprawiać swoją wydajność, osiągając końcowo błąd średniokwadratowy (MSE) na poziomie 15.88 dla zbioru treningowego oraz 13.85 dla zbioru testowego. Wyniki te potwierdzają zdolność modelu do uogólniania nieliniowych zależności w danych wejściowych.



Rysunek 4.1: Przebieg algorytmu dla zbioru multimodal-large



Rysunek 4.2: Dopasowanie przewidzianych wartości do danych treningowych



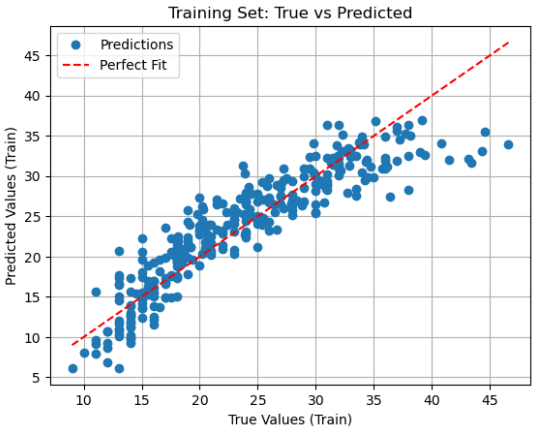
Rysunek 4.3: Dopasowanie przewidzianych wartości do danych testowych

Zbiór: Auto-MPG

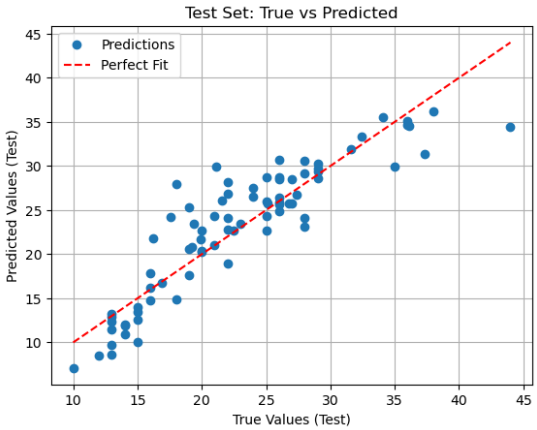
Drugim przypadkiem testowym był klasyczny zbiór danych Auto-MPG, zawierający cechy samochodów (m.in. moc silnika, masa, rok produkcji) oraz odpowiadające im wartości zużycia paliwa (MPG). Problem miał charakter regresyjny, a sieć MLP zawierała jedną warstwę ukrytą z 1 neuronem (ReLU) oraz warstwę wyjściową z liniową aktywacją.

W tym przypadku wykorzystano znacznie większą populację (300 osobników) oraz 30 000 generacji. Pomimo ograniczonej liczby warstw i niewielkiej głębokości sieci, model osiągnął zadowalające wyniki: MSE = 10.98 na danych treningowych oraz MSE = 10.74 na danych testowych. Wyniki te sugerują, że zastosowany model jest w stanie skutecznie odwzorować nieliniowe zależności między wieloma zmiennymi wejściowymi a wartością MPG, nawet przy uproszczonej architekturze.

Rysunek 4.4: Przebieg algorytmu dla zbioru auto-mpg



Rysunek 4.5: Dopasowanie przewidzianych wartości do danych treningowych

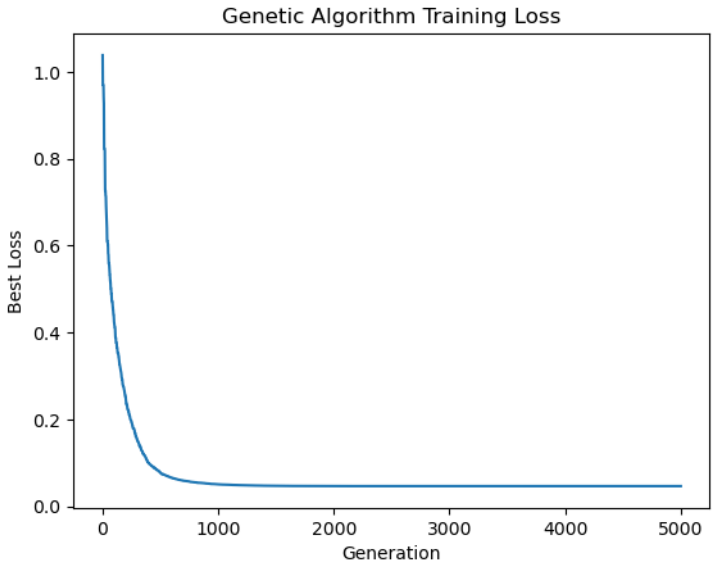


Rysunek 4.6: Dopasowanie przewidzianych wartości do danych testowych

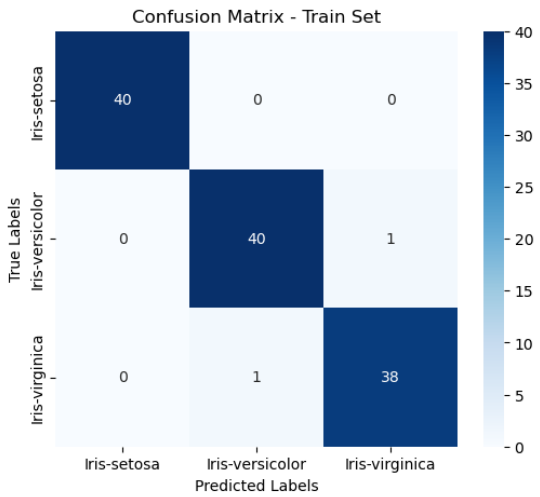
Zbiór: Iris

Ostatni eksperyment przeprowadzono na popularnym zbiorze Iris, zawierającym dane o trzech gatunkach kwiatów klasyfikowanych na podstawie czterech cech morfologicznych. W tym przypadku problem miał charakter klasyfikacyjny, a sieć została odpowiednio przystosowana do pracy w trybie klasyfikacji: warstwa ukryta zawierała 1 neuron z aktywacją ReLU, a warstwa wyjściowa miała 3 neurony z aktywacją softmax.

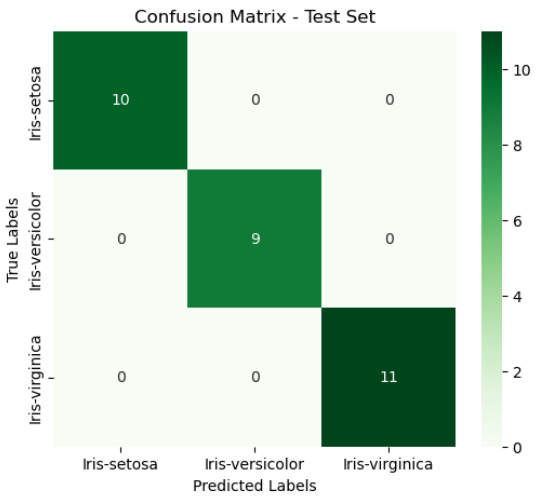
Trening trwał 5000 generacji i również wykorzystano populację 300 osobników. Efekty uczenia były bardzo pozytywne — uzyskano F1-score = 0.9833 dla danych treningowych i F1-score = 1.0000 dla danych testowych. Wysoka jakość klasyfikacji została również potwierdzona przez analizę macierzy pomyłek, w której model poprawnie sklasyfikował wszystkie przypadki testowe. Wynik ten świadczy o wysokiej zdolności sieci do uogólniania oraz dużej precyzji nawet przy ograniczonej liczbie neuronów.



Rysunek 4.7: Przebieg algorytmu dla zbioru iris



Rysunek 4.8: Dopasowanie przewidzianych wartości do danych treningowych



Rysunek 4.9: Dopasowanie przewidzianych wartości do danych testowych

4.4. Wnioski

Zrealizowane ćwiczenie potwierdziło, że algorytm genetyczny może skutecznie pełnić rolę mechanizmu optymalizacyjnego w procesie uczenia sieci neuronowych typu MLP. Pomimo braku wykorzystania klasycznego algorytmu propagacji wstecznej i gradientów, sieć była w stanie osiągać dobre rezultaty zarówno w zadaniach regresyjnych, jak i klasyfikacyjnych.

Zaletą podejścia genetycznego jest jego odporność na typowe problemy związane z lokalnymi minimami, zastoje w nauce czy niestabilność wynikającą z nieciągłości funkcji aktywacji lub kosztu. Dzięki reprezentacji chromosomów jako spłaszczonych wektorów wag i biasów możliwe było płynne zastosowanie operatorów krzyżowania i mutacji, a elitarność populacji pozwoliła zachować najbardziej obiecujące rozwiązania w kolejnych generacjach.

Wyniki eksperymentów potwierdzają skuteczność podejścia:

- w zbiorze multimodal-large sieć poprawnie odwzorowała silnie nieliniowe zależności przy bardzo małej populacji, co świadczy o dobrej eksploracji przestrzeni rozwiązań;
- w zbiorze Auto-MPG uzyskano niskie wartości błędu mimo ograniczonej głębokości sieci, co sugeruje efektywność podejścia także w typowych zadaniach regresyjnych;
- w zbiorze Iris osiągnięto doskonałą jakość klasyfikacji (F1 = 1.0), co potwierdza zdolność algorytmu do rozróżniania klas nawet przy bardzo ograniczonej liczbie neuronów.

Ze względu na charakter zadania, nie prowadzono szczegółowych testów, koncentrując się na

maksymalizacji funkcji celu. Podejście to pozwoliło na skupienie się na dopracowaniu algorytmu i osiągnięciu optymalnych wyników w zadanych warunkach.

Mimo relatywnie dłuższego czasu trenowania w porównaniu do metod gradientowych, elastyczność i uniwersalność algorytmu genetycznego czyni go interesującą alternatywą, szczególnie w sytuacjach, gdzie klasyczne metody zawodzą lub są trudne do zastosowania. Ostatecznie, zastosowanie podejścia ewolucyjnego pozwala nie tylko osiągnąć dobrą jakość predykcji, ale również poszerza wachlarz technik możliwych do wykorzystania przy trenowaniu modeli sztucznej inteligencji.