Obliczenia Ewolucyjne – projekt nr 2

Urszula Wąsowska, Bartosz Krawczyk, Maciej Kapitan

1. Użyte Technologie

Do przygotowania projektu użyliśmy języka Python w wersji 3.7 wraz z biblioteką numpy ułatwiającą niektóre obliczenia – takie jak np. losowanie liczb. Do generowania wykresów użyliśmy biblioteki matplotlib

Uruchomienie projektu wymaga zainstalowanego interpretera pythona wraz z odpowiednimi bibliotekami

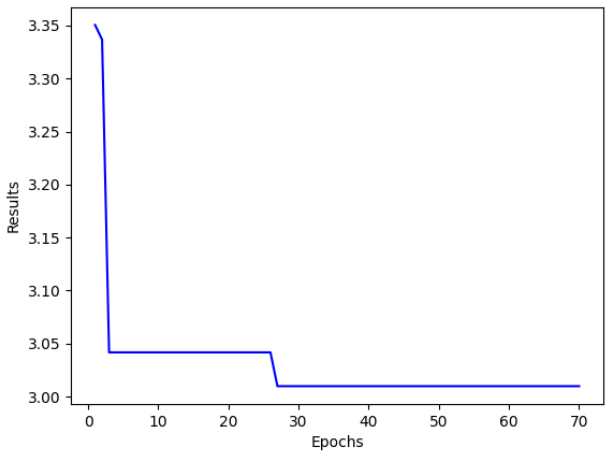
2. Uwagi na temat implementacji:

Aby algorytm genetyczny był efektywny, szybki i dopasowany do wielu różnych parametrów, wprowadziliśmy w naszej aplikacji kilka modyfikacji
- krzyżujemy osobniki dopóki rozmiar nowego pokolenia nie będzie taki sam jak rozmiar poprzedniego
- po każdej epoce dodawany jest najlepszy osobnik z poprzedniej, po to żeby najlepsze rozwiązanie nie zostało zgubione na drodze operacji genetycznych

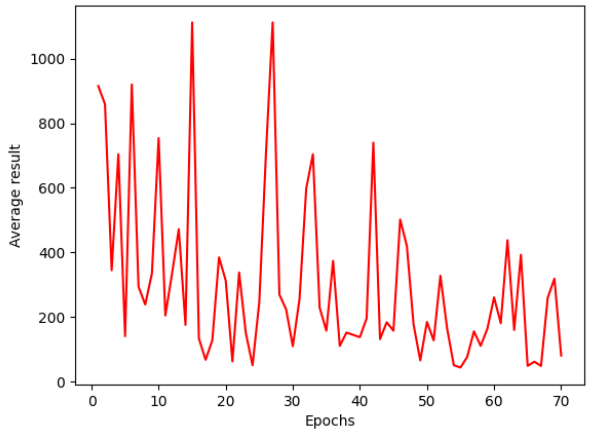
3. Wykresy

Funkcją jaką badamy w projekcie jest funkcja Goldsteina – ma ona minimum w punkcie (-1, 0), którego wartość to 3

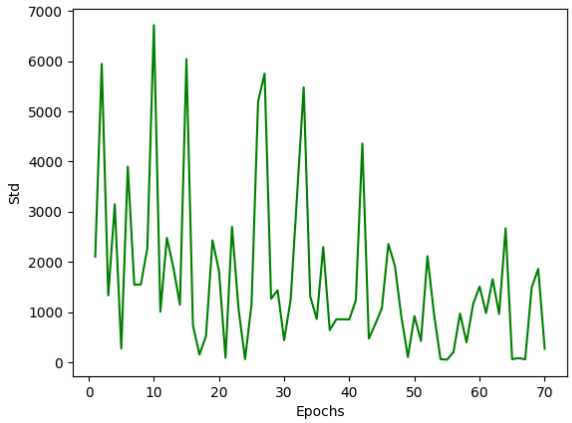
- Najlepsza wartość funkcji w kolejnych epokach



- Średnia wartość funkcji w kolejnych epokach



- Odchylenie standardowe w kolejnych epokach



Analizując wykresy można zauważyć że najlepszy wynik stopniowo maleje – znajdowane są kolejene, lepsze rowiązania. Jeśli chodzi o średnie wyniki funkcji i odchylenie standardowe, maleją one wolniej i ich wartość zmienia się skokowo – na drodze mutacji i krzyżowania powstają często zbyt duże wartości, co może dawać zaburzone wyniki

4. Porównanie wyników dla różnych konfiguracji algorytmu

Porównanie procentu selekcji:

Wysoki procent selekcji (90):
Srednia: 3.123971052560095, Mediana: 3.053967841197597

Niski procent selekcji (20):
Srednia: 3.04649451427685, Mediana: 3.000277939303478

Procent selekcji warunkuje ile osobników z poprzedniego pokolenia będzie poddane krzyżowaniu, jednak w naszej implementacji, nowe osobniki będą krzyżowane dopóki nowe pokolenie będzie w takim samym rozmiarze jak stare. Lepsze wyniki uzyskał algorytm z niższym procentem selekcji – jest to spowodowane tym że krzyżujemy tylko najlepszych osobników z każdej generacji, więc można stwierdzić łatwiej osiągnąć spodziewany wynik.

Porównanie prawdopodobienstwa mutacji:

Wysokie prawdopodobienstwo mutacji: (0.8)
Srednia: 3.0911261025417662, Mediana: 3.0814038343650023

Niskie prawdopodobienstwo mutacji: (0.1)
Srednia: 3.4022401887323284, Mediana: 3.1344885683150867

Wyniki algorytmu dla większego procentu mutacji i są o wiele lepsze, niż dla niższych prawdopodobieństw. Z tej obserwacji można wysnuć wniosek, że mutacja i inwersja są niezbędne do działania algorytmu genetycznego – zwiększają znacząco szanse na znalezienie optymalnego rozwiązania

Porównanie metod krzyzowania:

Krzyzowanie arytmetyczne
Srednia: 3.0538631434464585, Mediana: 3.013940432761328

Krzyzowanie mieszajace alfa
Srednia: 3.0000110009033425, Mediana: 3.0000002064210882

Krzyzowanie mieszajace alfa beta
Srednia: 3.0001593583402237, Mediana: 3.0000052420435104

Krzyzowanie usredniajace
Srednia: 3.1106179073093587, Mediana: 3.0737472825884486

Krzyzowanie linearne
Srednia: 3.239395199640878, Mediana: 3.1949422005613686

Z metod krzyżowania najlepiej wypadło krzyżowanie mieszające, natomiast najgorze krzyżowanie linearne. Również krzyżowanie uśredniające ma nico gorszy wynik

Porównanie metod seleckji:

Seleckja najlepszych
Srednia: 3.0895688295679253, Mediana: 3.0320236380924497

Selekcja turniejowa
Srednia: 3.0090772262157293, Mediana: 3.0063036981338387

Seleckja ruletki
Srednia: 3.0014809839645835, Mediana: 3.000220682203398

Porównując średnie algorytmów można zauważyć delikatną przewagę selekcji turniejowej i selekcji ruletki nad selekcją najlepszych. Może być to spowodowane tym, że te dwie metody wprowadzają większą losowość, przez co jest większa szansa że wpadniemy na optymalne rozwiązanie

Porównanie metod mutacji:
Mutacja jednorodna
Srednia: 5.409768500162891, Mediana: 4.054060783481481

Mutacja gaussa
Srednia: 3.025425371638131, Mediana: 3.0146009788578643

Znacznie lepsza okazała się mutacja Gaussa. Mutacja jednorodna polega na losowaniu liczby z przedziału i zmutowany osobnik nie ma nic wspólnego z poprzednim, więc nowe osobniki nie będą podobne do najlepszych tylko całkowicie losowe.

Wpływ liczby epok:
Liczba epok: 200
Srednia: 3.0270005577059944, Mediana: 3.0126438138611684

Liczba epok: 20
Srednia: 3.6728790415893764, Mediana: 3.2266594596929865

Analizując wyniki, można wyraźnie zauważyć wpływ liczby epok na działanie algorytmu genetycznego – o wiele lepszy wynik został osiągnięty dla dwustu epok – każda kolejna epoka daje większą szansę na znalezienie optymalnego rozwiązania

Wpływ wielkości populacji:
Liczba osobnikow: 400
Srednia: 3.0255920169480883, Mediana: 3.0150463659773674

Liczba osobnikow: 40
Srednia: 3.0419157826959067, Mediana: 3.001591378311052

Tutaj z kolei można zauważyć że liczba osobników ma niewielki duży wpływ na działanie algorytmu – im więcej osobników tym lepszy algorytm.

5. Porównanie z implementacją binarną

Algorytm rzeczywisty:

Srednia: 3.42098096351637, Mediana: 3.029092835467729
time: 3.3896093368530273

Algorytm binarny:

Srednia: 6.377429321458425, Mediana: 3.0000094710214205
time: 15.302298307418823

Porównując wyniki algorytmu widać że o wiele szybszy i skuteczniejszy okazał się algorytm genetyczny oparty na implementacji rzeczywistej. W każdym z testów został uruchomiony 20 razy algorytm z populacją liczącą 100 osobników i 70 epok

6. Porównanie prostych metaheurystyk

Porównania prostych metaheurystyk dokonałem analizując średnią liczbę iteracji po których każdy algorytm znajduje optymalne rozwiązanie (dla minimalizacji funkcji goldsteina za optymalne rozwiązanie przyjęliśmy liczbę mniejszą niż 3.5). Programy zostałe przekształcone tak żeby optymalizowały funkcję Goldsteina oraz zwracały liczbę iteracji po jakiej został osiągnięty wynik przyjęty przez nas za optymalny.

Algorytm wspinaczkowy

Average: 20510.05 , Median: 445.5

Random Sampling:

Average: 64500.25 , Median: 73614.0

Random Walking:

Average: 90366.5 , Median: 100000.0

Simulated annealing:

Average: 72498.95 , Median: 100000.0

Najlepszym wynikiem wykazał się algorytm wspinaczkowy. Algorytmy Random walking i Simulated annealing w większości przypadków nie były w stanie znaleźć optymalnego rozwiązania w 100000 iteracjach. O wiele mniej iteracji do znalezienia optymalnego rozwiązania wymagały oba algorytmy genetyczne.