# Sprawozdanie z algorytmu ewolucyjnego – Aleksander Pizoń

## Wstęp

Algorytm ewolucyjny jest narzędziem optymalizacyjnym wykorzystywanym do minimalizacji funkcji celu w problemach ciągłych. Algorytm ewolucyjny działa na zasadzie analogii do biologicznego procesu ewolucyjnego opartego na mechanizmach selekcji, krzyżowana, mutacji oraz sukcesji. Proces selekcji ma za zadanie wybranie do krzyżowania lepszych punktów Pt z większym prawdopodobieństwem niż gorsze. Krzyżowanie ma za zadanie wygenerować punkt „pośredni” pomiędzy rodzicami. Tak stworzony potomek jest następnie mutowany co polega na losowej zmianie jego chromosomów. Na końcu zachodzi sukcesja której zadaniem jest zdecydowanie które osobniki przejdą do następnej generacji. Cykl ten jest powtarzany aż do osiągnięcia warunku stopu. Bardzo ważnym pojęciem dotyczącym algorytmu ewolucyjnego jest funkcja dopasowania (fitness-function) na której podstawie jest przeprowadzany proces selekcji oraz sukcesji.

## Opis algorytmu i eksperymentów

Napisany przeze mnie program wykonywany jest poprzez uruchomienie pliku main.py. Pogram polega na wykonaniu 6 eksperymentów (3 dla funkcji Rastringa i 3 dla funkcji Griewanka) dotyczących różnego doboru parametrów algorytmu ewolucyjnego. Zestawy argumentów do wykonania algorytmu są zwracane przez odpowiednie funkcje z pliku parameters.py w postaci obiektu dataclass. Implementacje funkcji wykorzystywanych w eksperymentach znajdują się w pliku functions.py. W pliku plots.py znajdują się funkcje odpowiedzialne za tworzenie i zapisywanie wykresów pokazujących przebiegi eksperymentów, a w pliku tools.py funkcje i klasy pomocnicze potrzebne do prawidłowego działania programu.

Sam algorytm ewolucyjny jest zaimplementowany w pliku solver.py. Znajdują się w nim 3 klasy – Function, Individual oraz Evolution. Klasa Function służy do tworzenia obiektów funkcji celu. Za jej pomocą obliczana jest wartość funkcji celu. Obiekty klasy Individual reprezentują osobniki wchodzące w skład populacji. Za pomocą tej klasy możliwe jest wyliczanie wartości funkcji celu oraz dopasowania dla danego osobnika. Klasa ta przechowuje również genom osobnika czyli jego współrzędne. Kluczową jest klasa Evolution w której zaimplementowany jest właściwy algorytm ewolucyjny. Podczas inicjacji obiektu klasy Evolution, na podstawie dostarczonego zestawu argumentów tworzona jest startowa populacja (tworzona z rozkładu normalnego gdzie wartość średnia oraz sigma są podane w argumentach) a następnie obliczane są wartości funkcji dopasowania dla każdego z osobników. Dzięki metodzie evolve możliwe jest przeprowadzanie kolejnych cyklów ewolucji. Metoda składa się z kolejnych elementów: selekcja osobników do reprodukcji (selekcja turniejowa) za pomocą metody tournament\_selection, utworzenie potomków za pomocą metody crossover (krzyżowanie jednopunktowe), mutacja potomków za pomocą metody mutation (perturbowanie chromosomów z prawdopodobieństwem pm), wyliczenie nowych wartości funkcji dopasowania przy pomocy metody eval\_fitnesses, wybranie nowej populacji do następnej generacji za pomocą metody choose\_population oraz aktualizacja danych przechowywanych pod atrybutem data.

## Eksperymenty

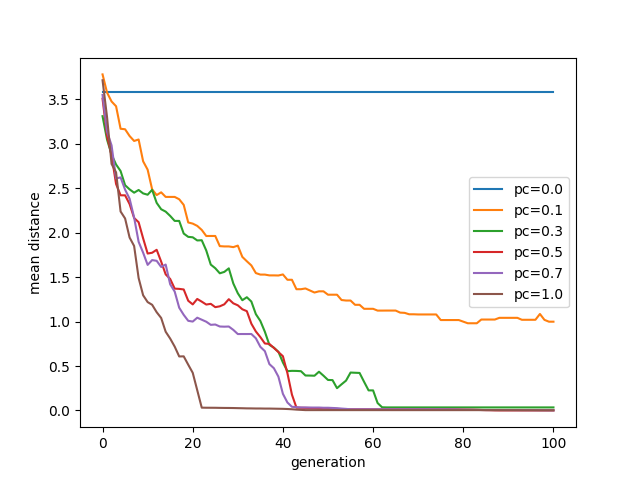
Ze względu na to że w moim programie funkcja dopasowanie jest znormalizowana (zsumowane wartości funkcji dopasowania dla całej populacji zawsze dają 1) do porównywania skuteczności algorytmu używam średniego dystansu populacji od minimum globalnego (mean\_dist). Decyzję tę popieram lepszą wizualizacją zależności skuteczności algorytmu od zmienianych parametrów.

Wyniki eksperymentów są przedstawione jako wykres oraz tabela. Wykres przedstawia zależność średniej odległości populacji do minimum globalnego od generacji. W tabeli zamieszczone są informacje o najlepszych indywiduach pod koniec działania algorytmu gdzie champ\_x1 i champ\_x2 to odpowiednio kolejne współrzędne najlepszego osobnika a MSE to średni bład kwadratowy obliczony dla tego osobnika.

Eksperyment 1 był prowadzony dla funkcji Rastrigina, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od prawdopodobieństwa krzyżowania (pc). Wartości pozostałych parametrów były następujące: liczebności populacji = 20, punkt środkowy do tworzenia początkowej populacji = (2.5, 2.5), odchylenie przy tworzeniu początkowej populacji = 1, prawdopodobieństwo mutacji = 0.5, odchylenie mutacji = 1, ilość generacji = 100.

Algorytm testowany był dla wartości pc równych: 0., 0.1, 0.3, 0.5, 0.7, 1.

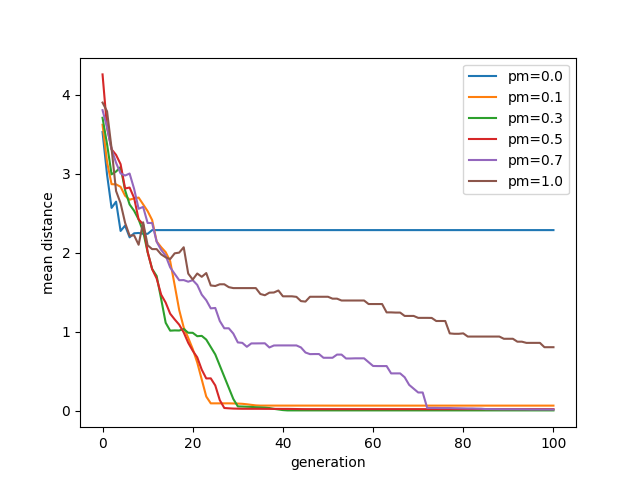
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **pc** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 0 | 1,78 | 2,03 | 15,56 | 2,70 |
| 0,1 | 0,991 | 0,958 | 2,26 | 1,38 |
| 0,3 | -0,0111 | 0,0208 | 0,110 | 0,0236 |
| 0,5 | 0,00219 | 0,000575 | 0,00102 | 0,00226 |
| 0,7 | -0,00264 | -0,00466 | 0,00570 | 0,00536 |
| 1 | 0,000941 | -0,00341 | 0,00248 | 0,00354 |



Eksperyment 2 był prowadzony dla funkcji Rastrigina, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od prawdopodobieństwa mutacji (pc). Wartości pozostałych parametrów były następujące: liczebności populacji = 20, punkt środkowy do tworzenia początkowej populacji = (2.5, 2.5), odchylenie przy tworzeniu początkowej populacji = 1, prawdopodobieństwo krzyżowania = 0.5, odchylenie mutacji = 1, ilość generacji = 100.

Algorytm testowany był dla wartości pm równych: 0., 0.1, 0.3, 0.5, 0.7, 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **pm** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 0 | 2,06 | 0,989 | 6,01 | 2,29 |
| 0,1 | 0,0455 | -0,0457 | 0,819 | 0,0645 |
| 0,3 | 0,00387 | -0,000789 | 0,00310 | 0,00395 |
| 0,5 | 0,0171 | -0,00472 | 0,0621 | 0,0177 |
| 0,7 | -0,00866 | 0,000375 | 0,0149 | 0,00867 |
| 1 | 1,03 | -0,029 | 1,44 | 1,03 |



Eksperyment 3 był prowadzony dla funkcji Rastrigina, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od liczebności populacji (size). Wartości pozostałych parametrów były następujące: punkt środkowy do tworzenia początkowej populacji = (2.5, 2.5), odchylenie przy tworzeniu początkowej populacji = 1, prawdopodobieństwo krzyżowania = 0.5, prawdopodobieństwo mutacji = 0.5, odchylenie mutacji = 1, ilość generacji = 100.

Algorytm testowany był dla wartości size równych: 10, 20, 40, 60, 80, 100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **size** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 10 | 0,0121 | -0,0108 | 0,0522 | 0,0162 |
| 20 | 0,00238 | 0,00350 | 0,00356 | 0,00423 |
| 40 | 0,000709 | -0,000720 | 0,000202 | 0,00101 |
| 60 | 0,00182 | 0,000650 | 0,000740 | 0,00193 |
| 80 | -0,00535 | 0,000820 | 0,00582 | 0,00541 |
| 100 | -0,000457 | 0,00389 | 0,00304 | 0,00392 |

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **pc** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 0 | 25,8 | 26,5 | 0,543 | 37,0 |
| 0,1 | 9,42 | -4,32 | 0,0309 | 10,4 |
| 0,3 | -3,19 | -4,47 | 0,00868 | 5,49 |
| 0,5 | -3,12 | 4,43 | 0,00765 | 5,42 |
| 0,7 | 0,00798 | -0,0119 | 0,0000674 | 0,0143 |
| 1 | 0,0279 | 0,000699 | 0,000390 | 0,0279 |

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznieEksperyment 4 był prowadzony dla funkcji Griewanka, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od prawdopodobieństwa krzyżowania (pm). Wartości pozostałych parametrów były następujące: liczebności populacji = 40, punkt środkowy do tworzenia początkowej populacji = (25, 25), odchylenie przy tworzeniu początkowej populacji = 5, prawdopodobieństwo mutacji = 0.5, odchylenie mutacji = 10, ilość generacji = 150.

Algorytm testowany był dla wartości pc równych: 0., 0.1, 0.3, 0.5, 0.7, 1.

Eksperyment 5 był prowadzony dla funkcji Griewanka, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od prawdopodobieństwa mutacji (pc). Wartości pozostałych parametrów były następujące: liczebności populacji = 40, punkt środkowy do tworzenia początkowej populacji = (25, 25), odchylenie przy tworzeniu początkowej populacji = 5, prawdopodobieństwo mutacji = 0.5, odchylenie mutacji = 10, ilość generacji = 150.

Algorytm testowany był dla wartości pm równych: 0., 0.1, 0.3, 0.5, 0.7, 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **pm** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 0 | 19,1 | 18,1 | 0,241 | 26,3 |
| 0,1 | -0,0569 | -0,0157 | 0,00168 | 0,0590 |
| 0,3 | 0,0208 | -0,0492 | 0,000823 | 0,0534 |
| 0,5 | 0,00339 | 0,0120 | 0,0000420 | 0,0125 |
| 0,7 | 3,13 | -4,46 | 0,00753 | 5,45 |
| 1 | -6,29 | -0,122 | 0,0136 | 6,29 |

Obraz zawierający tekst, diagram, linia, Wykres

Opis wygenerowany automatycznie

Eksperyment 6 był prowadzony dla funkcji Griewanka, a jego zadaniem było sprawdzenie skuteczności algorytmu w zależności od liczebności populacji (size). Wartości pozostałych parametrów były następujące: punkt środkowy do tworzenia początkowej populacji = (25, 25), odchylenie przy tworzeniu początkowej populacji = 5, prawdopodobieństwo krzyżowania = 0.5, prawdopodobieństwo mutacji = 0.5, odchylenie mutacji = 10, ilość generacji = 150.

Algorytm testowany był dla wartości size równych: 10, 20, 40, 60, 80, 100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **size** | **champ\_x1** | **champ\_x2** | **champ\_cost** | **MSE** |
| 10 | -3,19 | 4,56 | 0,0127 | 5,57 |
| 20 | 3,20 | -4,45 | 0,00901 | 5,48 |
| 40 | 0,0562 | 0,0287 | 0,00178 | 0,0631 |
| 60 | -0,0457 | -0,00553 | 0,00105 | 0,0461 |
| 80 | 0,0143 | 0,00615 | 0,000112 | 0,0156 |
| 100 | 3,14 | 4,44 | 0,00740 | 5,44 |

## Obraz zawierający tekst, diagram, linia, zrzut ekranu Opis wygenerowany automatycznieWnioski

Piewszym nasuwającym się wnioskiem jest to że skuteczność algorytmu ewolucyjnego zależy od wielu parametrów których wartości na przyjąć. Takie parametry jak liczebność populacji, reprezentatywność początkowej populacji, odległość początkowej populacji od globalnego minimum, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji czy ilość generacji mają duży wpływ na jakość otrzymanego wyniku.

Na podstawie eksperymentów 1 i 4 możemy zauważyć jaki wpływ na skuteczność algorytmu ma parametr prawdopodobieństwa krzyżowania. W przypadku gdy parametr ten wynosi zero lub jest zbyt mały, algorytm nie jest w stanie w optymalnej liczbie generacji odnaleźć minimum globalnego. W przypadku zaimplementowanego przeze mnie algorytmu widać że wraz ze wzrostem parametru pc algorytm coraz szybciej odnajduje globalne minimum. Jest to spowodowane tym że im więcej osobników przejdzie do etapu krzyżowania tym większa będzie różnorodność osobników podlegających sukcesji.

Dzięki eksperymentom 2 i 5 widoczny jest wpływ prawdopodobieństwa mutacji na skuteczność algorytmu. W tym przypadku aby algorytm był w stanie odnaleźć minimum globalne konieczny jest dobór wartości tego parametru w przedziale 0.1 – 0.7. Dla zbyt małych wartości pm populacja zbyt szybko traci różnorodność ponieważ reprodukowane chromosomy nie podlegają perturbacji co utrudnia odnalezienie minimum. Z kolei dla zbyt dużej wartości pm nie jest możliwe aby korzystne chromosomy przetrwały ponieważ są zbyt często mutowane.

Na poniższych wykresach przedstawiono jak wygląda populacja w zależności od generacji dla skrajnych wartości parametru pm. (po lewej pm = 0, po prawej pm = 1)

Obraz zawierający zrzut ekranu, wzór, tekst, Wielobarwność

Opis wygenerowany automatycznieObraz zawierający zrzut ekranu, tekst, wzór, Wielobarwność

Opis wygenerowany automatycznie

Na wykresach utworzonych podczas eksperymentów 3 i 6 przedstawiona jest zależność skuteczności algorytmu od liczebności populacji. W przypadku funkcji Rastrigina nie widać znaczącego wpływu liczebności populacji na skuteczność algorytmu. Z kolei dla funkcji Griewanka można zauważyć że dla większych wartości liczebności populacji algorytm szybciej odnajduje minimum i a dla wartości size = 80 odnajduje minimum globalne. Dla wartości size = 10 i size = 20 algorytm wolno zbiega do minimum.

Dla każdego eksperymentu widoczny jest wpływ ilości generacji na skuteczność całego algorytmu. Widać że często algorytm potrzebuje nawet kilkudziesięciu iteracji żeby wydostać się z minimum globalnego. Można zauważyć że ilość generacji potrzebna algorytmowi do odnalezienia minimum globalnego zależy od funkcji i od odległości populacji początkowej od minimum globalnego. Dla funkcji Rastrigina przy zachowaniu optymalnych parametrów jest to około 50 generacji a dla funkcji griewanka około 80.

Poniżej przykładowa wizualizacja pokazująca końcowe populacje w eksperymencie 5. Widać na niej że dla zbyt małej wartości pc algorytm nie tworzy dostatecznej ilości potomków aby w optymalnej liczbie kroków odnaleźć minimum globalne.

Obraz zawierający zrzut ekranu, tekst, Wielobarwność, wzór

Opis wygenerowany automatycznie