Sprawozdanie z realizacji zadania

Algorytmy ewolucyjne

Tymoteusz Kwieciński

320637

Maj/Czerwiec 2024

1 Cel zadania

Zadanie to było podzielone na 3 części, które stopniowo pozwalały zapoznać się nam z algorytmami ewolucyjnymi - niezwykle interesującą klasą algorytmów bezgradientowych, które za pomocą mechanizmów inspirowanych biologicznym rozmnażaniem są w stanie rozwiązywać skomplikowane problemy.

1.1 Zadanie 1

Celem pierwszego zadania było zapoznanie się z algorytmami ewolucyjnymi na prostym przypadku optymalizacji - poszukiwania minimum zadanej funkcji. Algorytm miał optymalizować jedną z dwóch zadanych funkcji:

- Funkcję kwadratową: $x^2 + y^2 + 2z^2$
- 5-cio wymiarową funkcję Rastrigina

Obie funkcje mają globalne minimum w 0. Algorytm genetyczny miał wykorzystywać mutację gaussowską oraz krzyżowanie jednopunktowe.

1.2 Zadanie 2

Celem drugiego zadania było rozwiązanie problemu podobnego do *stock-stock* problem. Dla koła o zadanym promieniu r oraz zadanego zbioru prostokątów o zadanych wymiarach oraz jego wartości, naszym zadaniem jest ułożenie prostokątów w kole w taki sposób, aby zmaksymalizować sumę ich wartości. Prostokąty nie powinny nachodzić na siebie, a także każdy powinien być wewnątrz koła. Każdy prostokąt można wykorzystywać dowolnie wiele razy.

W mojej implementacji nie wykorzystywałem opcji obrotu prostokątów, ale i tak udało mi osiągnąć zamierzone rezultaty.

2 Szczegóły implementacji

2.1 Zadanie 1

Każdy osobnik (*individual*) był reprezentowany za pomocą jednowymiarowej listy, która odpowiadała kolejnym zmiennym zadanej funkcji. Korzystając z [1] parametry każdego osobnika z prawdopodobieństwem 0.2 zostały modyfikowane - do każdej zmiennej osobnika dodawany zostawał szum gaussowski o odchyleniu standardowym ustalonym jako hiperparametr zależny również od szerokości przedziału.

2.1.1 Krzyżowanie

Krzyżowanie polegało na wyborze jednej wybranej liczby naturalnej, nie większej niż długość listy reprezentującej osobnika, a następnie modyfikacja dwóch wejściowych osobników, w taki sposób, aby pierwsza część osobnika pierwszego trafiła do pierwszej części nowego osobnika pierwszego, zaś druga część osobnika pierwszego trafiła do drugiej części osobnika drugiego. Krzyżowanie następowało z prawdopodobieństwem 0.7

2.1.2 Selekcja

Selekcja po każdej iteracji była przeprowadzana za prawdopodobieństwem będącą wynikiem funkcji softmax z parametrem temperatury, której wejściem były wyniki zadanej funkcji dla zadania. To znaczy, osobniki z niższą wartością funkcji celu, miały większą szansę dostać się do następnego etapu.

W implementacji uwzględniłem również tak zwaną *elitę* - część najlepszych osobników była na pewno wybierana do następnej iteracji algorytmu.

2.2 Zadanie 2

Każdy osobnik był reprezentowany jako klasa w `Pythonie`. Każdy osobnik zawierał listę prostokątów, reprezentowanych jako pozycja jego środka w układzie współrzędnym, wysokość oraz szerokość, a także koło o ustalonym promieniu. Zdecydowałem się na jak najprostszą implementację, tak aby skupić się lepiej na działaniu algorytmów ewolucyjnych, a nie zastanawiać się nad elementami kodu.

2.2.1 Generowanie prostokątów

Każdy nowy prostokąt był generowany losowo, przez wylosowanie pozycji jego środka. Następnie przeprowadzałem weryfikację, czy prostokąt znajduje się wewnątrz koła, a także iterując po wszystkich pozostałych prostokątach, szukałem ewentualnej kolizji. Jeżeli prostokąt postawiony był nieprawidłowo, to przez określoną liczbę iteracji (1000) próbowałem wstawić go ponownie. Po tym czasie, aby ograniczyć czas działania programu, przechodziłem do kolejnych instrukcji.

2.2.2 Mutacje

Zaimplementowałem 4 sposoby mutacji osobników:

- *move+skip* - do wybranego osobnika dodawany był szum gausowski, o określonej wariancji. Jeżeli po dodaniu takiego szumu prostokąt kolidował z innym prostokątem, dodanie szumu było anulowane.
- *move+remove* - do wybranego osobnika dodawany był szum gausowski, o określonej wariancji. Jeżeli po dodaniu takiego szumu prostokąt kolidował z innym prostokątem, modyfikowany prostokąt był usuwany.
- *replace* - wybrany prostokąt był usuwany, a następnie stosowałem opisaną wyżej procedure dodawania prostokąta, aby zastąpić ten usunięty
- *slide* - wybrany prostokąt był przesuwany maksymalnie w górę i w prawo, tak aby maksymalnie zminimalizowac niewykorzystaną przestrzeń

W każdej iteracji mutacji modyfikowany był każdy z prostokątów w osobniku, kolejność prostokątów do modyfikacji była wybierana losowo. Każdy osobnik był mutowany *inplace*, to znaczy osobnik przed mutacją nie był zachowany. Aby maksymalnie zwiększyć liczbę prostokątów w każdym osobniku, powyższa mutacja towarzyszy również mutacji polegającej na dodaniu 10 nowych kwadratów.

Każdy osobnik z prawdopodobieństwem 0.7 jest mutowany na jeden z 4 sposóbów opisanych powyżej lub z prawdopodobieństwem 0.3 dodawane są do niego prostokąty.

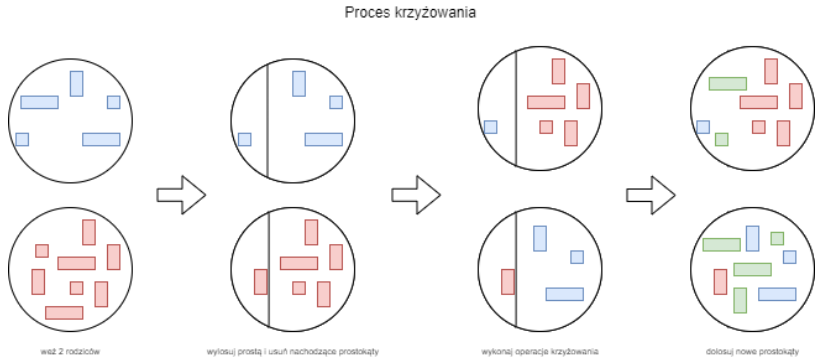
2.2.3 Krzyżowanie

Jako metodę krzyżowania zastosowałem analogię do *single-point-crossover*. To znaczy, dokładny algorytm krzyżowania to:

1. Losowo wybierz sieczną w kole - poziomą lub pionową prostą dzielącą koło na dwie części
2. Utwórz dwa nowe puste osobniki
3. Skopiuj wszystkie prostokąty z pierwszego osobnika do pierwszego nowego osobnika, które są na lewo (lub do góry) od prostej i jej nie przecinają
4. Skopiuj wszystkie prostokąty z drugiego osobnika do drugiego nowego osobnika, które są na lewo (lub do góry) od prostej i jej nie przecinają
5. Skopiuj wszystkie prostokąty z pierwszego osobnika do pierwszego nowego osobnika, które są na prawo (lub na dół) od prostej i jej nie przecinają
6. Skopiuj wszystkie prostokąty z drugiego osobnika do drugiego nowego osobnika, które są na prawo (lub na dół) od prostej i jej nie przecinają

7. Dodaj nowe prostokąty w każdym z nowych osobników, aby uzupełnić pionową lub poziomą linię, gdzie nie skopiowały sie prostokąty

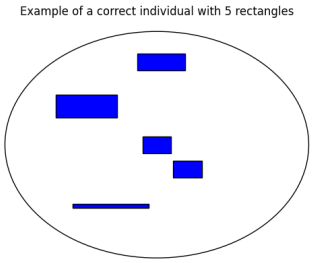
Krzyżowanie dodaje dwa nowe osobniki do populacji, nie zamieniając tych poprzednich. grafika przedstawiająca proces krzyżowania znajduje się na 1.



Rysunek 1: Graficzny opis sposobu krzyżowania

2.2.4 Selekcja

Selekcja po każdej iteracji była przeprowadzana za prawdopodobieństwem będącą wynikiem funkcji *softmax* z parametrem temperatury, której wejściem były wyniki zadanej funkcji dla zadania. To znaczy, osobniki z wyższą wartością funkcji celu, miały większą szansę dostać się do następnego etapu.
W implementacji uwzględniłem również tak zwaną *elitę* - część najlepszych osobników była na pewno wybierana do następnej iteracji algorytmu. Domyślnie ustawiłem liczbę osobników elitarnych jako 10% wszystkich osobników, aby zachować najlepsze wyniki.



Rysunek 2: Przykładowy osobnik wykorzystywany w zadaniu

2.3 Zadanie 3

Każdy osobnik był reprezentowany jako jedna instancja klasy `MLP`, którą zaimplementowałem w poprzednim ćwiczeniu dotyczącym sieci neuronowych. Wartość każdego osobnika reprezentowana była jako funkcja straty po danych treningowych - MSE w przypadku problemu regresji oraz $cross-entropyloss$ w przypadku problemu klasyfikacji.

Inicjalizacja wag sieci została również wykorzystana z poprzedniego ćwiczenia - w eksperymentach wykorzystywałem funkcję ReLU jako funkcje aktywacji w warstwach ukrytych oraz rozkład gaussowski z wariancją równą 1 i średnią 0 w przypadku ostatniej warstwy - liniowej lub softmax.

2.3.1 Mutacje

Mutacja była zaimplementowana jako dodanie gausowskiego szumu do wag oraz biasu sieci. Wynikiem mutacji był nowy osobnik, nie modyfikowała ona starego, dzięki czemu najlepsze osobniki miały szansę przetrwać do późniejszych generacji.

2.3.2 Krzyżowanie

Jako metodę krzyżowania zastosowałem algorytm porobny do *single-point-crossover*. Ponieważ wszystkie osobniki miały taką samą architekturę, to na każdej warstwie sieci wykonywałem *single-point-crossover* dla wag oraz biasu. Krzyżowanie następowało wzdłuż wymiaru wyjściowego. Punkt przełamania był losowany z rozkładu jednostajnego.

Krzyżowanie dodaje dwa nowe osobniki do populacji, nie zamieniając tych poprzednich, co umożliwia zachowanie najlepszych wyników rodziców.

2.3.3 Selekcja

Podobnie jak w poprzednich podzadaniach zastosowałem selekcje na podstawie wyników poszczególnych osobników, dla odmiany tym razem zastosowałem jednak losowanie ze zwracamiem.

Selekcja po każdej iteracji była przeprowadzana za prawdopodobieństwem będącą wynikiem funkcji *softmax* z parametrem temperatury, której wejściem były wyniki zadanej funkcji dla zadania. To znaczy, osobniki z niższą wartością funkcji celu, miały większą szansę dostać się do następnego etapu. Podobnie jak poprzednio, w implementacji uwzględniłem również tak zwaną *elitę* - część najlepszych osobników była na pewno wybierana do następnej iteracji algorytmu. Domyślnie, podobnie jak w poprzednich zadaniach ustawiłem liczbę osobników elitarnych jako 10% wszystkich osobników, aby zachować najlepsze wyniki.

3 Wyniki eksperymentów

3.1 Zadanie 1

Okazuje się, że algorytm ewolucyjny zadziałał bardzo dobrze. Zbiegał bardzo szybko - nawet po kilku iteracjach. Dla obu funkcji otrzymywał bardzo dobre rezultaty i był dowolnie blisko rzeczywistego minimum zadanej funkcji. Wyniki eksperymentów można zaobserwować na 3. Do eksperymentów wykorzystałem wartość $\sigma_0 = 0.1$, w przypadku funkcji kwadratowej przedział z którego losowane były początkowe wartości osobników to $[-10, 10]$, zaś w przypadku funkcji Rastrigina to $[-5.12, 5.12]$. Losowanie było przeprowadzane z rozkładu jednostajnego na zadanym przedziale.

Dodatkowo, w eksperymencie w każdej iteracji wybierane było dokładnie n osobników, a dla jednoznacznego porównania wyników eksperymenty kończyły się po 100 iteracjach. Liczba *elitarnych* osobników wynosiła $10\%n$.

liczba osobników	średni wynik osobnika	wynik najlepszego osobnika
10000	16.3 ± 0.5	$1.1 \pm 2.0 \cdot 10^{-7}$
1000	16.1 ± 1.7	$6.6 \pm 3.5 \cdot 10^{-6}$
100	12.9 ± 13.5	$5.4 \pm 8.9 \cdot 10^{-4}$
10	16.7 ± 16.7	$4.8 \pm 5.1 \cdot 10^{-1}$

Tabela 1: Porównanie wyników algorytmu dla różnych wartości rozmiarów populacji przy zadaniu funkcji kwadratowej. Najlepszy wynik osiągany był dla bardzo dużej liczby osobników, ale średnie wyniki są bardzo zbliżone dla uruchomień algorytmu.

Zmiana temperatury w funkcji softmax nie miała prawie żadnych wpływów na wyniki algorytmu, w związku z tym porównania nie umieszczam w raporcie.

3.2 Zadanie 2

Eksperymenty, które wykonałem w ramach tego zadania można podzielić na dwie części - eksperymenty na różnych zbiorach danych oraz eksperymenty sprawdzające wpływ hiperparametrów na zachowanie algorytmów.

Wszystkie eksperymenty wykonywałem powtarzając uruchomienie algorytmu trzykrotnie dla uśrednienia wyników.

3.2.1 Zbiory danych

W tej części eksperymentów przetestowałem działanie sieci z domyślnymi hiperparametrami. Chciałem zweryfikować, czy algorytm jest w stanie osiągnąć podane wyniki. Zastosowane hiperparametry to między innymi:

hiperparametr	wartość
ilość osobników	100
temperatura	1000
rodzaj mutacji	slide
część mutowanych osobnikó w każdej epoce	0.2
część osobników poddawanych krzyżowaniu w każdej epoce	0.7

Tabela 2: Wyniki algorytmu dla różnych zbiorów danych. Jak widać w przypadku każdego zbioru udało się uzyskać satysfakcjonujący wynik.

zbiór danych	średni wynik osobnika	wynik najlepszego osobnika	wymagany wynik
800	45253±647	44424±429	30000
850	342513±3568	340887±5585	-
1000	20100±810	19596±836	17500
1100	25820±544	24713±549	25000
1200	30487±490	30162±447	25000

Tabela 3: Wyniki algorytmu dla różnych zbiorów danych. Jak widać w przypadku każdego zbioru udało się uzyskać satysfakcjonujący wynik.

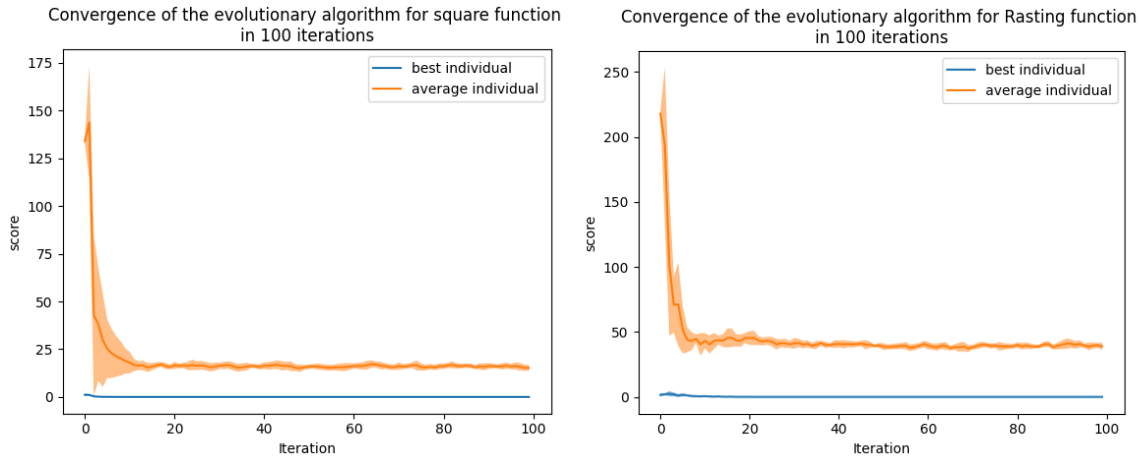
Jak widać na 3 i 5 najtrudniejszy okazał się zbiór 1100, gdzie algorytm ledwo przekroczył granicę minimalnego wyniku. Warto zauważyć, że o ile pod koniec algorytmu szybkość wzrostu wyników nie jest już tak duża jak na początku, prawdopodobnie z powodu zmniejszonej różnorodności osobników, to wciąż wyniki populacji stale rosną.

3.2.2 Hiperparametry

Aby sprawdzić wpływ różnych hiperparametrów na działanie algorytmu przeprowadziłem kilka eksperymentów. Wszystkie eksperymenty zostały przeprowadzone używając hiperparametrów takich jak w poprzednim zadaniu i na 30 generacjach algorytmu.

Rozmiar populacji Jasno widać, że większy rozmiar populacji oznacza lepsze wyniki algorytmu.

Rodzaj mutacji Najlepszy rodzaj mutacji został zastosowany do treningów - *slide*. Okazuje się, że to proste podejście pozwala uzyskać znacznie większe wyniki niż pozostałe sposoby, które są bardzo zbliżone działaniem



Rysunek 3: Zbieżność algorytmów dla funkcji kwadratowej i Rastinga dla populacji z $n = 1000$ osobnikami

rozmiar populacji	średni wynik	najlepszy wynik
10 osobników	35105±285	35860±393
50 osobników	36165±1690	37527±1329
100 osobników	36983±1140	38280±1129

Tabela 4: Wpływ rozmiaru populacji na wynik

rodzaj mutacji	średni wynik	najlepszy wynik
replace	30923±863	32420±871
move+skip	30571±496	31780±642
move+remove	29308±352	30527±450
slide	36983±1140	38280±1129

Tabela 5: Wpływ rodzaju mutacji na wynik

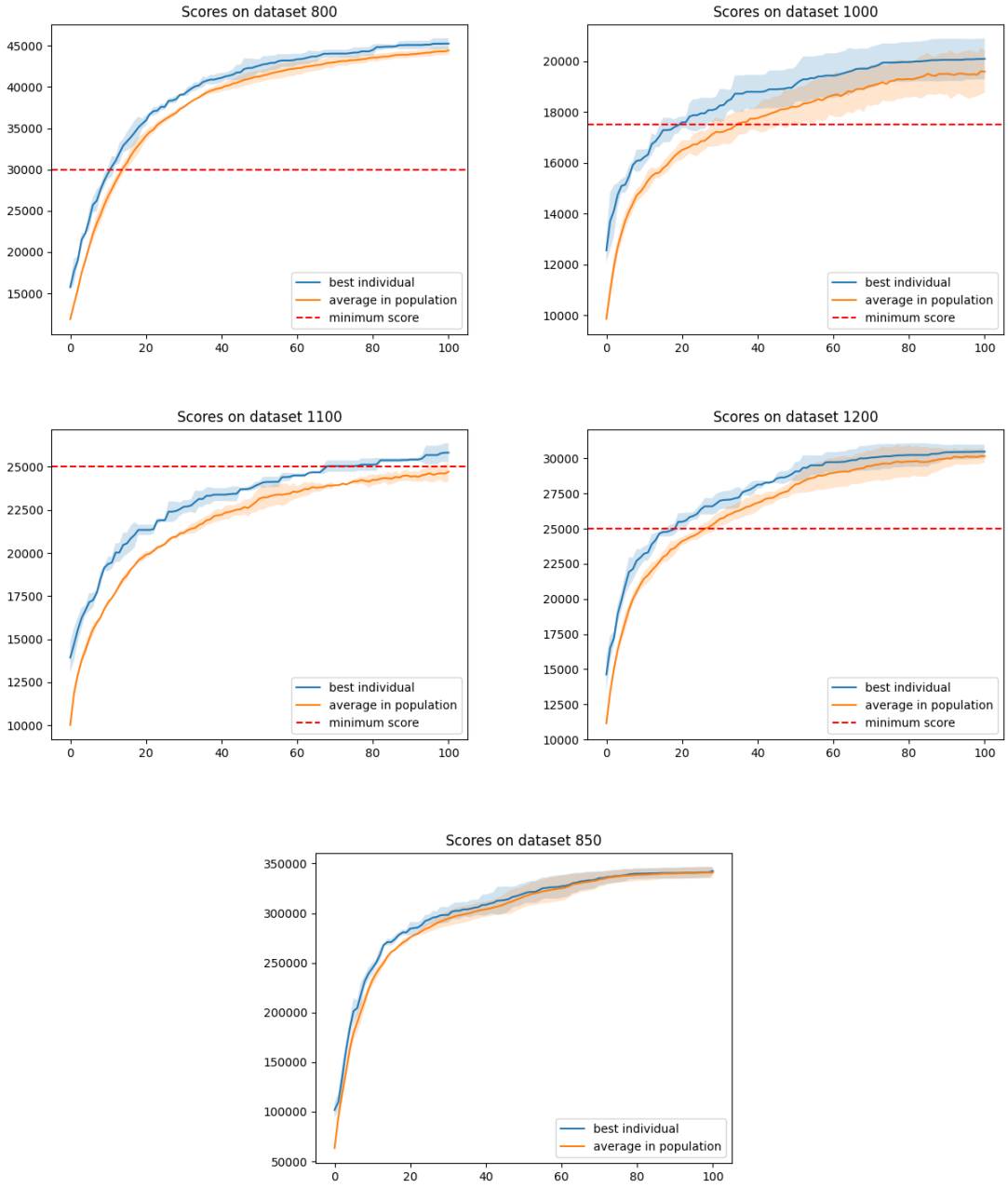
Prawdopodobieństwo wykonania operacji crossover Najlepsze wyniki są osiągane na prawdopodobieństwie ustalonym jako domyślnie - 0.7.

Prawdopodobieństwo wykonania operacji mutate Jak się okazuje najlepsze wyniki zostały osiągnięte dla domyślnej wartości prawdopodobieństwa wykonania mutacji - 0.2, najgorsze wyniki są dla braku mutacji - wówczas różnorodność osobników jest znacznie mniejsza. W przypadku pozostałych wartości, wyniki są podobne.

Temperatura selekcji Im wyższa temperatura w funkcji softmax tym bardziej wyrównane prawdopodobieństwa wyboru poszczególnych osobników - to znaczy lepsze osobniki będą wybierane z mniejszym prawdopodobieństwem niż w przypadku mniejszej temperatury.

3.3 Zadanie 3

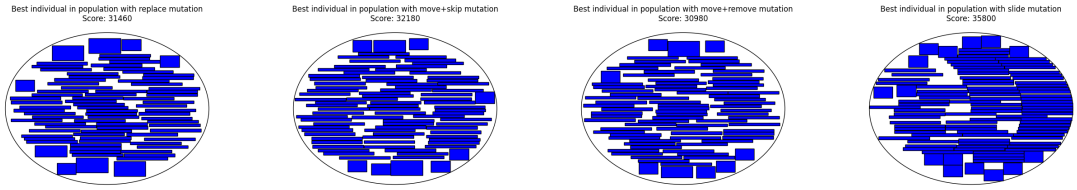
W zadaniu 3 właściwie jedynym hiperparametrem, który testowaliśmy to *magnitude* - czyli odchylenie standardowe szumu, który był dodawany do wag sieci w trakcie procesu ewolucyjnego. Na każdym zbiorze danych przetestowałem 3 różne wartości tego hiperparametru - $0.1, 0.01, 0.001$. Można o nim myśleć jako o parametrze *learning rate* dla sieci neuronowych optymalizowanych za pomocą propagacji wstecznej.



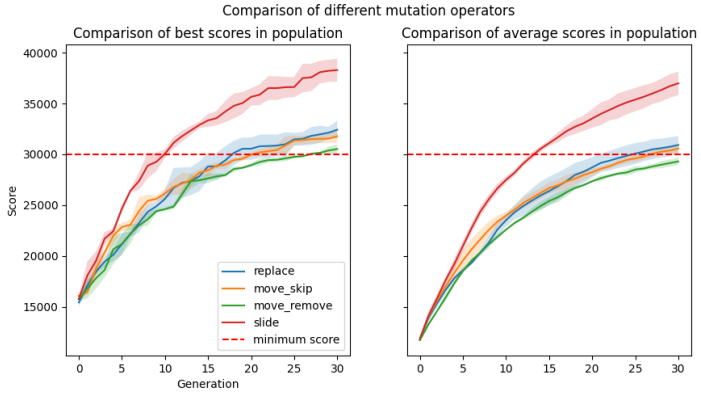
Rysunek 4: Zbieżność algorytmu dla różnych zbiorów

W przypadku tego zadania wykonywałem eksperymenty wykorzystując 100 osobników w każdej populacji przez 300 generacji. Dodatkowo wyniki uśredniałem wyniki wykorzystując 5 uruchomień algorytmu.

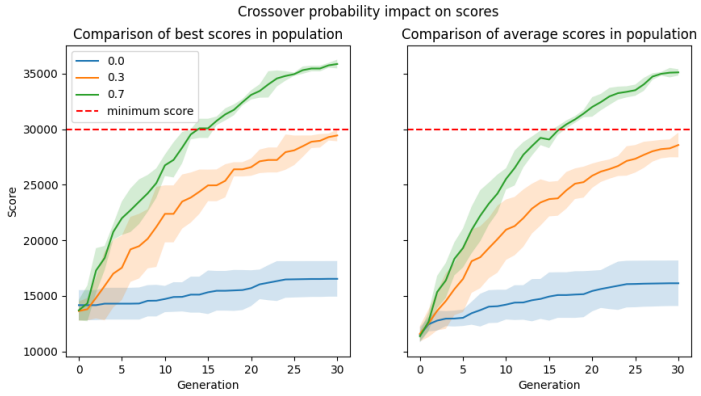
Niestety wyniki, które udało się osiągnąć, były znacznie gorsze niż te osiągane w zadaniu o sieciach neuronowych, ale z uwagi na złożoność problemu, i tak satysfakcjonujące.



Rysunek 5: Najlepsze osobniki w każdej z mutacji. Jak widać 3 pierwsze metody są podobne w wyniku, ostatnia *dosuwa* prostokąty do góry i na prawo, co dobrze wpływa na score



Rysunek 6: Wpływ rodzaju mutacji na zbieżność algorytmu



Rysunek 7: Wpływ prawdopodobieństwa wykonania operacji crossover na zbieżność algorytmu. Domyślnie stosowana wartość jest najlepsza. Brak krzyżowania osobników powoduje bardzo słabe wyniki algorytmu

3.3.1 Multimodal

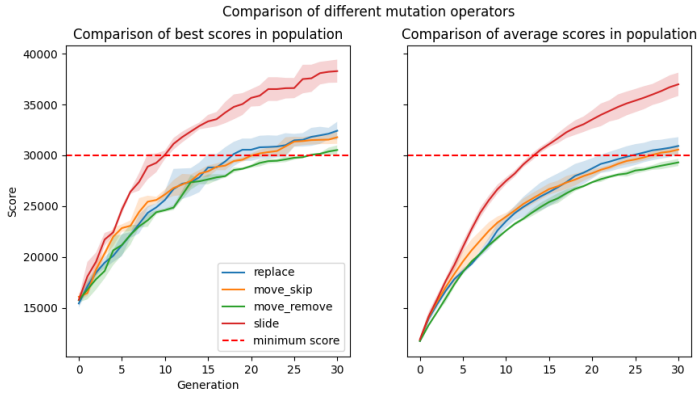
W przypadku tego zbioru danych mieliśmy wydzielone zbioru danych - testowy i treningowy. Trening oczywiście odbywał się na treningowym zbiorze, zaś krótka ewaluacja na testowym. Udało się osiągnąć zadowalające wyniki.

3.3.2 Iris

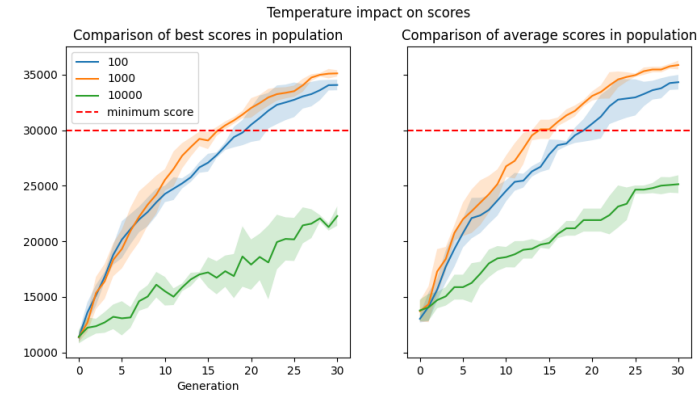
Zbiór Iris nie został domyślnie podzielony na treningowy i testowy. Przy każdym uruchomieniu dzieliłem go losowo na podzbiór treningowy i testowy, wyniki zbieżności zostały zbierane na zbiorze treningowym. W przypadku tego zbioru danych widać, że zbiega on prawidłowo, udaje się osiągnąć

Crossover probability	średni wynik	najlepszy wynik
0.0	16137±2046	16533±1600
0.3	28573±1107	29433±522
0.7	35105±285	35860±393

Tabela 6: Wpływ prawdopodobieństwa wykonania operacji crossover



Rysunek 8: Wpływ prawdopodobieństwa wykonania operacji mutate na zbieżność algorytmu. Jak widać najlepsze wyniki są osiągane dla prawdopodobieństwa 0.2. Najgorsze wyniki otrzymywane są gdy nie jest wykonywana żadna mutacja.



Rysunek 9: Wpływ temperatury na wyniki algorytmu. Jak widać 1000 wydaje się optymalną wartością temperatury. O ile na początkowym etapie mniejsza temperatura pozwala osiągać lepsze wyniki, to pod koniec ewolucji trochę lepiej sprawdza się temperatura 1000

dośc dobre wyniki, ale mimo to że zbiór jest bardzo prosty, to algorytm nie osiąga perfekcyjnego wyniku.

Ponieważ zbiór testowy w tym zbiorze danych okazał się bardzo mały, to wizualizacjia 12 jest przedstawiona na złączonych zbiorach danych.

3.4 MPG

Zbiór MPG nie został domyślnie podzielony na treningowy i testowy. Przy każdym uruchomieniu dzieliłem go losowo na podzbiór treningowy i testowy, wyniki zbieżności zostały zbierane na zbiorze treningowym. Jak widać na 13 najlepsze rezultaty były osiągane dla wartości 0.01.

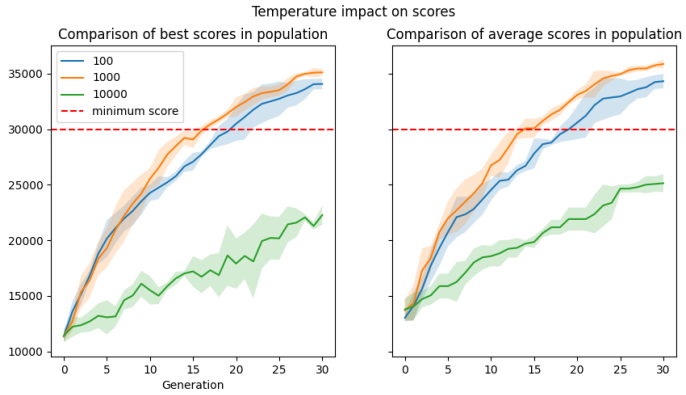
Zbiór testowy był tworzony za pomocą losowego podziału przy każdym uruchomieniu algorytmu.

prawdopodobieństwo mutacji	średni wynik	najlepszy wynik
0.0	26868±708	27360±806
0.2	35105±285	35860±393
0.4	32353±908	33147±526
0.6	32790±1089	33560±1207
0.8	32096±1943	33280±1702
1.0	31641±1707	32593±2080

Tabela 7: Wpływ prawdopodobieństwa wykonania operacji mutate

temperatura	średni wynik	najlepszy wynik
100	34313±653	34061±475
1000	35860±393	35105±285
10000	25133±802	22265±868

Tabela 8: Wpływ temperatury na wyniki algorytmu. Jak widać 1000 wydaje się optymalną wartością temperatury. O ile na początkowym etapie mniejsza temperatura pozwala osiągać lepsze wyniki, to pod koniec ewolucji trochę lepiej sprawdza się temperatura 1000



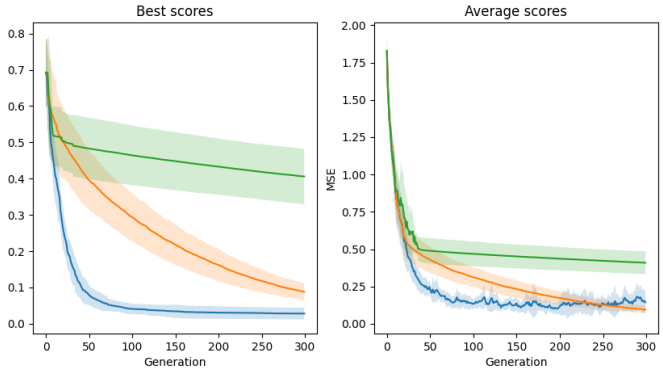
Rysunek 10: Wpływ temperatury na wyniki algorytmu. Jak widać 1000 wydaje się optymalną wartością temperatury. O ile na początkowym etapie mniejsza temperatura pozwala osiągać lepsze wyniki, to pod koniec ewolucji trochę lepiej sprawdza się temperatura 1000

Literatura

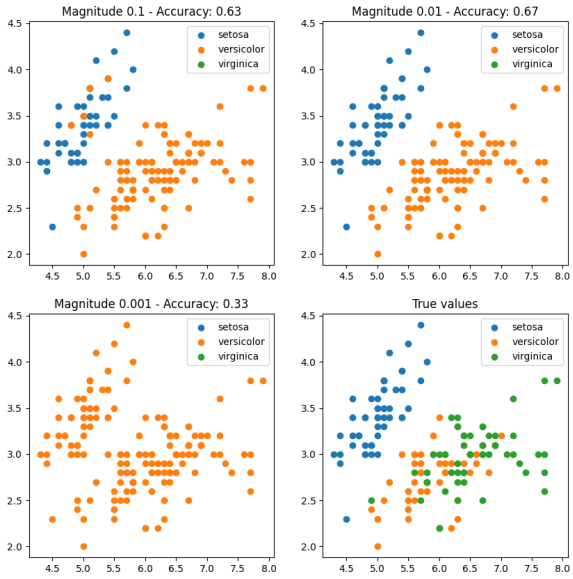
[1] Optimization by evolutionary computation.

4 Wnioski

Podsumowując wszystkie wyniki eksperymentów, pomyślnie udało mi się zaimplementować algorytm ewolucyjny dla 3 różnych problemów. Dzięki zrealizowanym ćwiczeniom, miałem okazje zapoznać się z podstawami uczenia ewolucyjnego, a także sprawdzić ich działanie i nauczyć się ich implementacji od podstaw. Okazuje się, że wielokrotnie parametry podawane jako domyślne, na przykład dotyczące części populacji wybieranej do mutacji, istotnie sprawowały się najlepiej.



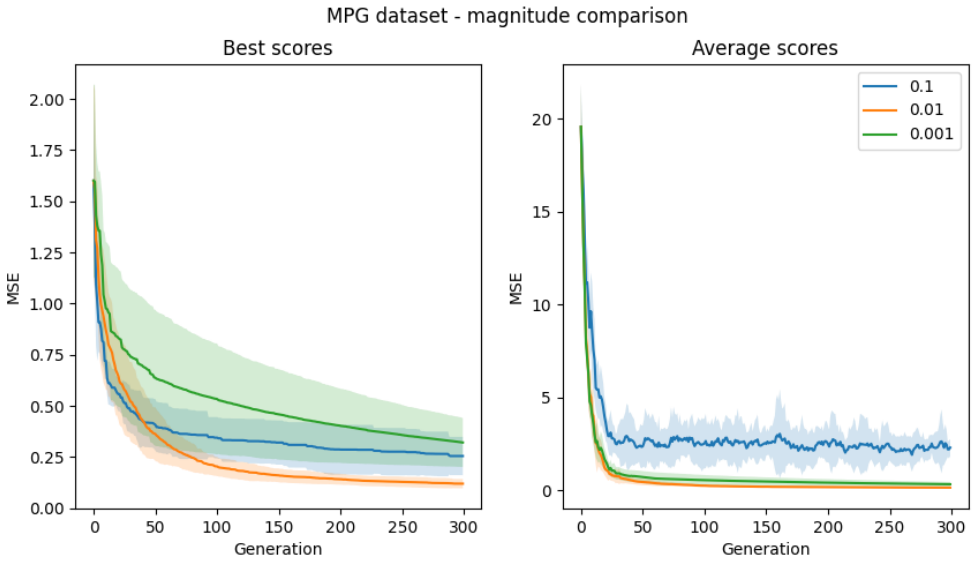
Rysunek 11: Porównanie zbieżności algorytmu dla różnych wartości *magnitude*. Jak widać większa wartośc tego hiperparametru zapewnia początkowo lepszą zbieżność, ale po wielu epokach wydaje się jakby mniejsza wartość *magnitude* mogłaby osiągnąć lepsze wyniki.



Rysunek 12: Wyniki klasyfikacja dla pierwszego uruchomienia algorytmu i najlepszego osobnika w populacji, na złączonym zbiorze treningowym i testowym

magnitude	wynik najlepszego osobnika
0.1	2.03±0.26
0.01	1.99±0.21
0.001	2.15±0.16

Tabela 9: Wyniki algorytmu dla zbioru MPG na zbiorze testowym



Rysunek 13: Wynik porównania zbieżności sieci dla zbioru MPG