Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Podstawy Sztucznej Inteligencji

Sprawozdanie z projektu SK.AE.7

Zdający:

Maria Konieczka
Alicja Poturała
Jakub Sikora

Prowadzący:

dr inż. Stanisław Kozdrowski

Spis treści

1 Treść zadania

Masz 10 kart ponumerowanych od 1 do 10. Znajdź przy użyciu algorytmu ewolucyjnego sposób na podział kart na dwie kupki w taki sposób że suma kart na pierwszej kupce jest jak najbliższa wartości A, a suma kart na drugiej kupce jest jak najbliższa wartości B.

2 Pełen opis funkcjonalny

2.1 Pomysł ogólny

Wykorzystany został algorytm genetyczny. Genotyp każdego osobnika opisany jest przez d bitów, gdzie d oznacza liczbę kart. Jeśli na k-tej pozycji znajduje się 1, oznacza to, że k-ta karta należy do kupki A, w przeciwnym wypadku należy do kupki B.

2.2 Funkcja celu

Funkcja celu określana jest przez sumę kwadratów błędów:

$$f(a) = (A - a)^2 + (B - (sum - a))^2$$

gdzie a oznacza sumę wartości kart na kupce A danego osobnika, a sum określa sumę wszystkich kart w talii, (sum-a) określa więc sumę wartości kart na kupce B danego osobnika. A i B oznaczają zadane wartości sum na kupkach. Naszym celem jest minimalizacja tej funkcji. Jak można zauważyć funkcja celu jest funkcją kwadratową jednej zmiennej. Wynika z tego, że na każdym przedziale posiada jedno minimum.

2.3 Strategia wyboru kolejnego pokolenia

Kolejnym zagadnieniem jest przyjęcie odpowiedniej strategii przy wyborze następnego pokolenia. Z racji tego, że funkcja celu posiada jedno minimum, nie trzeba eksplorować dziedziny w poszukiwaniu optymalnego rozwiązania. Postawiliśmy na wysoką presje selekcyjną i przyjęliśmy strategię elitarną. Do kolejnego pokolenia brane jest μ najlepszych osobników (o najmniejszej wartości funkcji celu).

2.4 Krzyżowanie

Z danej populacji wybieranych jest λ par osobników. Osobnicy są dobierani w pary: najpierw losowany jest jeden osobnik, a następnie bez zwracania pierwszego

osobnika do puli losowany jest kolejny. Następuje krzyżowanie. Najpierw wybierane są miejsca krzyżowań. Losowane są one bez zwracania spośród liczb od zera do d-2. Liczba punktów krzyżowań zawarta jest w pliku konfiguracyjnym. Mając ustalone miejsca krzyżowania, tworzony jest osobnik potomny (z jednej pary tworzony jest jeden osobnik potomny).

2.5 Mutacje

Zakładamy, że prawdopodobieństwo mutacji w genotypie wynosi $\frac{1}{10}$ - średnio co 10 genotyp posiada zmutowany gen. Można zauważyć, że prawdopodobieństwo mutacji konkretnego genu zależy od długości genotypu.

3 Struktura programu

Aplikacja podzielona została na dwie części: serwer algorytmu oraz część graficzną. Serwer algorytmu napisany zostanie w Pythonie i będzie wystawiała API do komunikacji z serwerem frontendowym. Do zaimplementowania komunikacji wykorzystamy bibliotekę Flask.

4 Serwer algorytmu

4.1 Opis klas

Serwer algorytmu korzysta z napisanych klas:
- CardsHandler,
- Judge,
- Phenotype,
- Population.

CardsHandler
Jest to obiekt, który zawiera wartości a i b, sędziego, a także parametry pliku konfiguracyjnego. Przy tworzeniu tego obiektu te parametry odczytywane są z pliku typu .json. Klasa ta zawiera tylko jedną funkcję *evaluate*(), która koordynuje wykonaniem całego algorytmu. W funkcji tej kolejno: - tworzona jest pierwsza losowa populacja - następnie w pętli na podstawie aktualnej populacji tworzona jest kolejna populacja, pętla wykonywana jest tyle razy ile wynosi parametr *epochs* zapisany w pliku konfiguracyjnym. - po skończeniu pętli, odczytywany jest najlepszy genotyp. - tworzone są listy kart A i B na podstawie najlepszego gentypu. - zwracany jest słownik zawierający te listy kart.

Judge

Jest obiektem tworzonym przez Cardshandlera przy jego powstaniu. Posiada on tablicę kart, długość i sumę wszystkich kart. Ma tylko jedną funkcję *goal_eval*() wykorzystywaną jednorazowo dla każdego osobnika przy wyliczeniu błędu. Do funkcji przekazywane są wartości a, b (wartości zadane na kupkach) oraz osobnik, na rzecz którego liczony jest błąd. Po obliczeniu tego błędu sędzia wywołuje funkcję osobnika, która ustawia wyliczony błąd jako wartość funkcji celu danego osobnika.

Phenotype

Jest to obiekt, który odwzorowuje jednego osobnika. Zawiera w sobie tablicę genów, ich liczbę, a także wartość funkcji kosztów. Osobniki tej klasy powstają przy wywołaniu funkcji tworzącej pierwszą populację, a także w wyniku krzyżowania dwóch osobników. Do generowania losowego osobnika służy funkcja *generate_random_genes*(), która za pomocą funkcji random() losuje, a następnie przypisuje wartości 0 lub 1 niezależnie dla każdego genu. Do tworzenia nowego osobnika z osobników już istniejących wykorzystywana jest funkcja *crossover*(). Funkcja ta wywoływana jest na osobniku "matce"przy generacji nowego pokolenia. Do tej funkcji przekazywane są jako argumenty: ójciec"(drugi osobnik) oraz liczba krzyżowań. W funkcji *crossover*() kolejno wykonywane jest:
- tworzenie nowego osobnika,
- losowanie bez zwracania miejsc krzyżowań,
- lista tych miejsc jest sortowana
- ustalany jest początkowy parent_id (True oznacza "matkę", False ójca")
- wykonywana jest pętla iterująca po genach: danemu genowi przypisywany jest gen tego rodzica, któremu aktualnie odpowiada wartość zmiennej parent_id, następnie sprawdzane jest czy nie należy zmienić wartości zmiennej parent_id.
- na koniec wywoływana jest funkcja *mutate*()

Funkcja *mutate*() najpierw wyznacza prawdopodobieństwo z jakim mutowany jest pojedynczy gen, a następnie iteruje po całej tablicy genów i z wyliczonym prawdopodobieństwem zmienia wartość genu.
Ostatnią funkcją w tej klasie jest funkcja *set_fitness*(), która ustawia atrybut fitness (wartość funkcji celu) na wyliczony przez Judge błąd. Jest to wykonywane tylko raz po stworzeniu osobnika.

Population

Jest to klasa zawierająca: parametry odczytane z pliku konfiguracyjnego, zadane wartości a i b, sędziego oraz listę osobników. Głównym zadaniem tej klasy jest tworzenie nowych pokoleń. Funkcja *generate_random_population*() wywoływana jest, aby stworzyć populację początkową. Tworzonych jest tyle osobników ile wynosi wartość parametru μ w pliku konfiguracyjnym: tworzony jest nowy osobnik,

osobnik losuje sobie geny, wywoływany jest sędzia, który ustawia wartość funkcji celu osobnikowi, na koniec tak stworzony osobnik dodawany jest do tworzonej populacji. Kolejną istotną funkcją jest *create_next_epoch*(). Wywoływana jest przez obiekt klasy CardsHandler na danej populacji. Z tej populacji losowane są pary osobników (tych par jest tyle ile wynosi parametr λ w pliku konfiguracyjnym). Po wylosowaniu pary osobników na osobniku "matka"wykonywana jest funkcja crossover, która zwraca osobnik "dziecko". Następnie wywoływany jest sędzia przypisujący nowemu osobnikowi odpowiednią wartość funkcji celu, po czym ten osobnik dodawany jest do listy dzieci. Po stworzeniu listy dzieci tworzona jest lista wszystkich osobników (poprzednie osobniki plus nowostworzone). Są one sortowane malejąco po wartości funkcji celu, a następnie tworzona jest nowy obiekt populacji. Przypisujemy do niego μ najlepszych osobników, a następnie ten obiekt jest zwracany do obiektu CardsHandlera. Do przypisywania nowych osobników populacji służy funkcja *push_phenotype*(). Funkcja ta dodaje do listy osobników danej populacji przekazywanego do funkcji osobnika. Ostatnią funkcją tej klasy jest funkcja *get_the_best_error*(), która to sortuje osobników z danej populacji i zwraca genotyp najlepszego osobnika. Wykorzysywane jest to do odczytania rezultatu algorytmu - z populacji końcowej odczytywany jest najlepsze rozwiązanie.

4.2 Opis pliku main.py

W pliku tym tworzymy obiekt serwera, włączamy mechanizm Cross-Origin-Resource-Sharing umożliwiający współpracę pomiędzy serwerami, serwer uruchamiany jest na wskazanym porcie (np. 5000). Dla żądania wykonania algorytmu określona jest funkcja *find_distribution_handler*(). W funkcji tej odczytywane są wartości zadane a i b, a następnie tworzony jest obiekt CardsHandler, wywoływana jest na nim funkcja *evaluate*(). Tworzony jest słownik, który zawiera listy kart A, B oraz informację o bezbłędnym wykonaniu żądania. Z funkcji zwracany jest obiekt typu Response, który zawiera ten słownik.

4.3 Serwer frontendowy

Zalążek serwera frontendowego został wygenerowany automatycznie za pomocą narzędzia `create-react-app`, które tworzy podstawową stronę w frameworku React oraz tworzy hostujący serwer w języku Javascript. Aplikacja uruchamiana jest z pliku `src/index.js` a następnie generowane są komponenty z folderu `src/components`. Najważniejszym komponentem jest kontener `AlgorithPage` który jest najbardziej widoczny dla użytkownika ponieważ zajmuje się umieszczaniem elementów na ekranie. Komponenty komunikują się ze sobą za pomocą akcji frameworka Redux. Komponent generuje akcje, która następnie jest obsługiwana przez funkcję nazywaną reducerem. Funkcja ta na podstawie akcji modyfikuje globalny

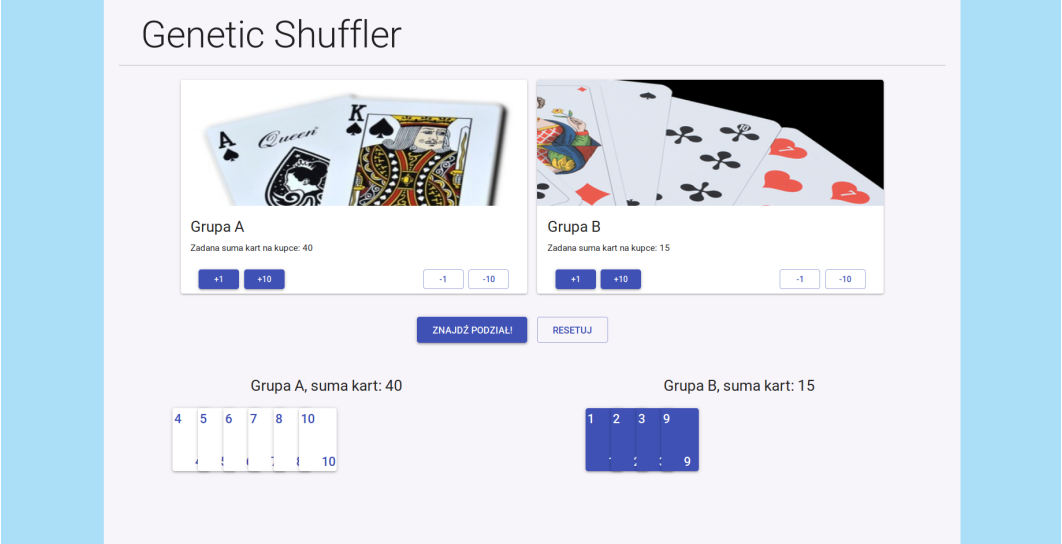
stan aplikacji z którego czytają komponenty. Wymusza to komunikację pomiędzy komponentami tylko w jednym kierunku co znacznie ją uproszcza. Reducer został złączony z dwóch podfunkcji: `valuesReducer` oraz `geneticReducer`. Pierwszy obsługuje argumenty wprowadzone przez użytkownika a drugi odpowiedź serwera.

5 Opis interfejsu użytkownika

Interfejs zajmuje się tylko interakcją z użytkownikiem i prezentacją danych. Uruchamiany jest jako strona insternetowa w przeglądarce.

Użytkownik zadać sumę wartości kart na każdej z kupek za pomocą zestawu przycisków +1, +10, -1, -10. Ponadto są 2 przyciski: jeden do czyszczenia strony (wyzerowania wartości kupek i pozbycia się kart) oraz do wysłania żądania dla zadanych wartości sum.

Wynik prezentowany jest zarówno za pomocą zestawu kart z odpowiednimi wartościami oraz uzyskaną przez nie sumą.



6 Pliki konfiguracyjne

Parametrami programu są:
- wartości kart
- λ - liczebność pokolenia
- μ - liczba generowanych osobników potomnych w każdej epoce
- liczba krzyżowań - ile miejsc krzyżowań przy tworzeniu nowego osobnika
- liczba epok - liczba iteracji algorytmu

Parametry te są zapisane w jednym pliku typu config.json. Plik ten zawiera obiekt, który jest nieuporządkowanym zbiorem par nazwa/wartość.

Listing 1: Struktura pliku config.json

```
{
        "cards": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        "mi": 50,
        "lambda": 30,
        "crosses": 3,
        "epochs": 100
}
```

Plik konfiguracyjny jest wczytywany przy tworzeniu CardsHandlera, czyli po przyjęciu każdego nowego zapytania od użytkownika. Pozwala to na zmianę pliku konfiguracyjnego przy włączonym serwerze.

7 Uruchamianie programu

7.1 Wymagania

Aby poprawnie uruchomić serwer algorytmu należy:
- zainstalowany python w wersji 3 (dokładna specyfikacja w pliku */algorithm_server/Pipfile*)
- zainstalowany pipenv; na Ubuntu można to zrobić przy pomocy poleceń:

```
sudo apt install python-pip
pip install pipenv
```

- środowisko Node.js - menedżer pakietów npm

7.2 Uruchomienie serwera algorytmu

W wierszu poleceń z poziomu katalogu */algorithm_server/* wywołujemy kolejno polecenia:

```
pipenv install
pipenv shell
python main.py
```

7.3 Uruchomienie serwera interfejsu użytkownika

Serwer klienta uruchamiany jest za pomocą programu `npm` który jest menedżerem pakietów środowiska `Node.js`. Aby uruchomić serwer należy wywołać w

katalogu */frontend_server* po kolei polecenia:

```
npm install
npm start
```

Polecenie `npm start` dokona procesu transpilacji nowego standardu języka Javascript zwanego JSX na tradycyjny Javascript, dlatego też uruchomienie serwera może zająć chwilę.

7.4 Uruchomienie

W celu łatwego korzystania z programu serwery można uruchomić zdalnie, a z samego programu można korzystać wpisując w przeglądarce adres:

$$http://mion.elka.pw.edu.pl/\sim jsikora$$

Jeżeli serwery uruchamiane są lokalnie, to w przeglądarce należy wpisać:

$$localhost : 3000$$

8 Przeprowadzone testy i wnioski

8.1 Testy jednostkowe

Zostały stworzone 4 pliki testujące po jednym na każdą klasę. Sprawdzane są m.in: poprawność obliczeń, tworzonych obiektów, zwracanych typów, reakcji na wywołanie funkcji z niepoprawnymi argumentami. Wszystkie testy pokazały poprawne działanie pojedynczych elementów programu.

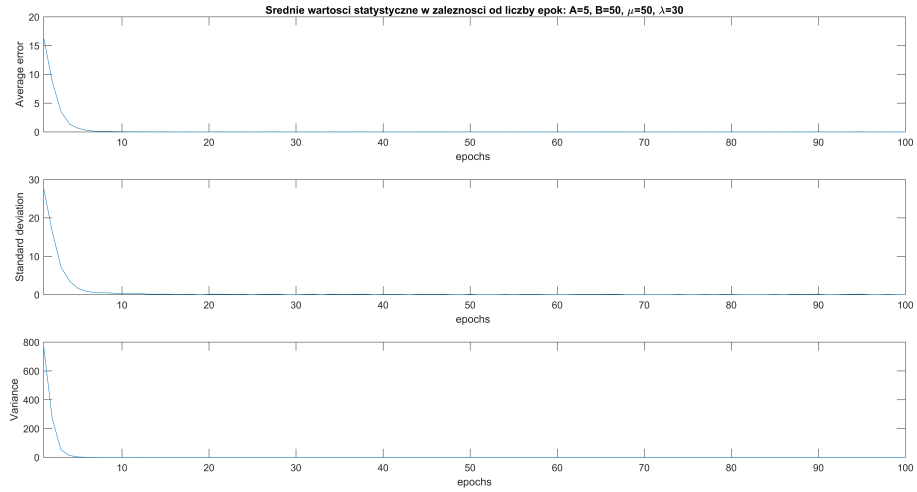
8.2 Testowanie parametru liczby epok

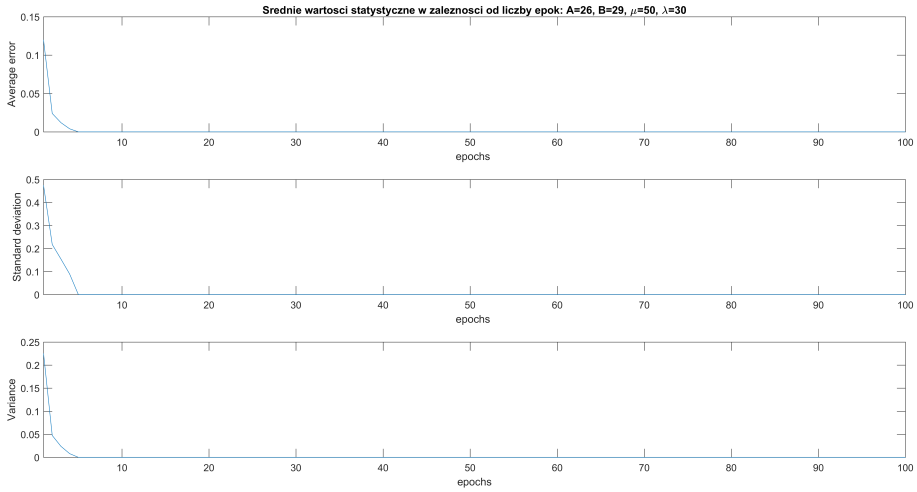
Przeprowadzony został test na średnie wartości statystyczne w zależności od liczby epok. Wyliczone zostały wartości: średnia, odchylenie standardowe i wariancja. Dla każdej epoki przeprowadzone było 1000 zapytań z jednakowymi parametrami A i B. Test był przeprowadzony dla różnych A i B oraz różnych wartości μ i λ.

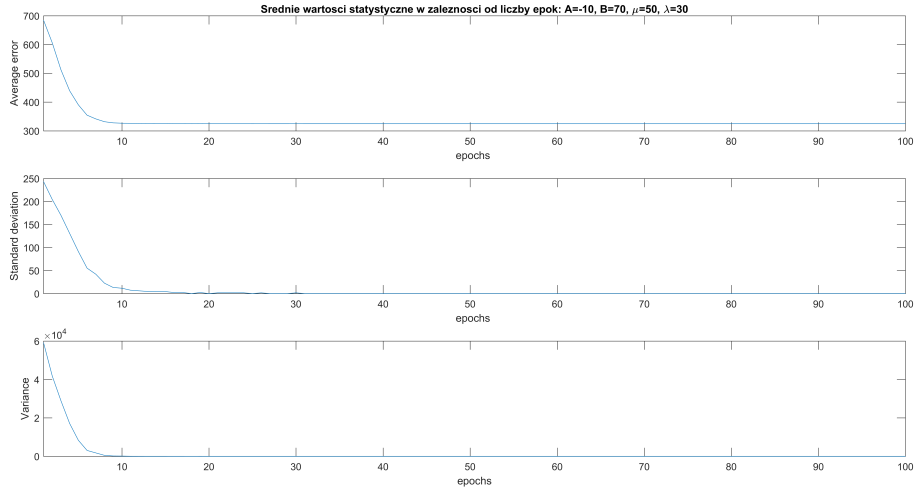
8.3 Duże wartości μ i λ

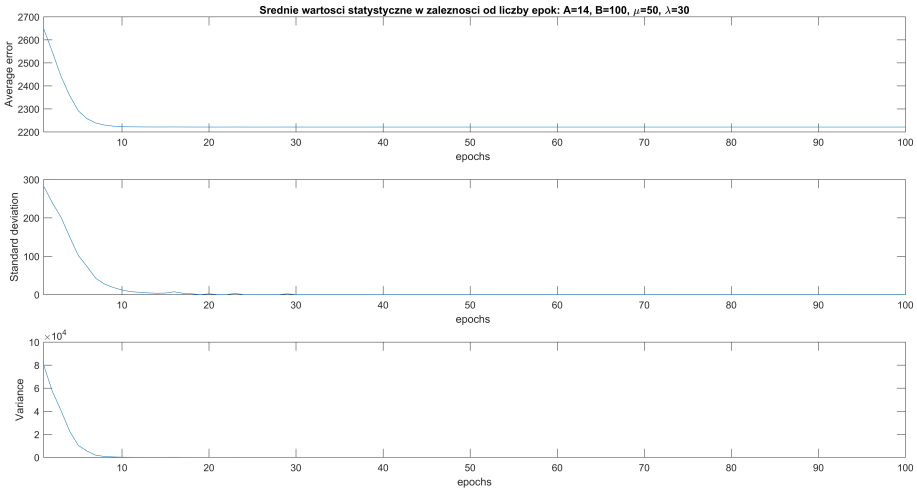
Przyjęte zostały wartości: $\mu = 50$ i $\lambda = 30$. Przeprowadzono eksperyment dla wartości zadanych A i B:
- A = 5, B = 50

- A = 26, B = 29
- A = -10, B = 70
- A = 14, B = 100





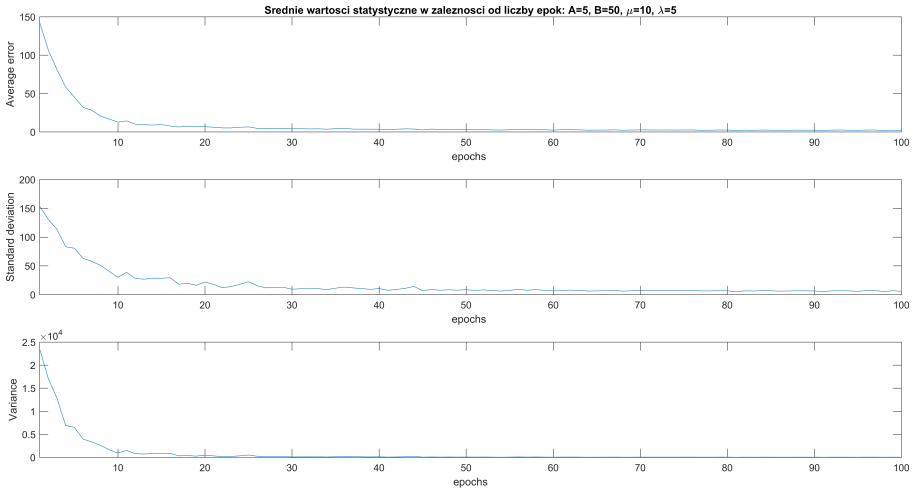


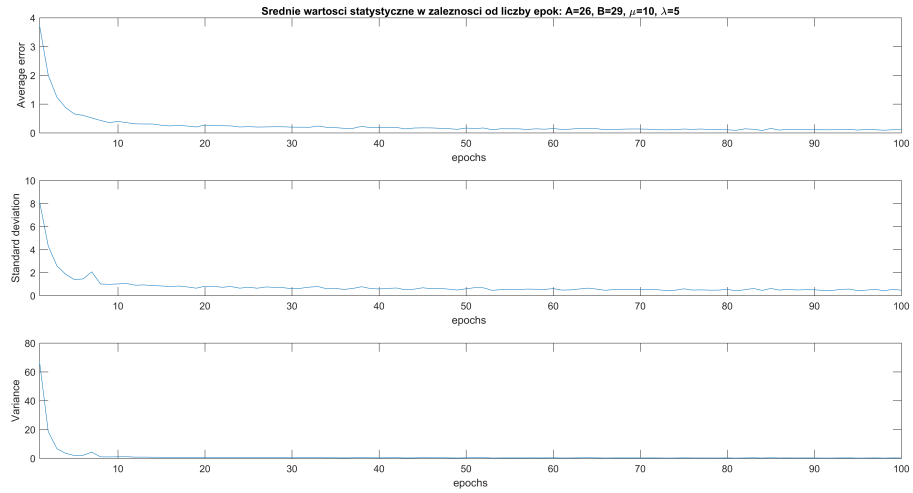


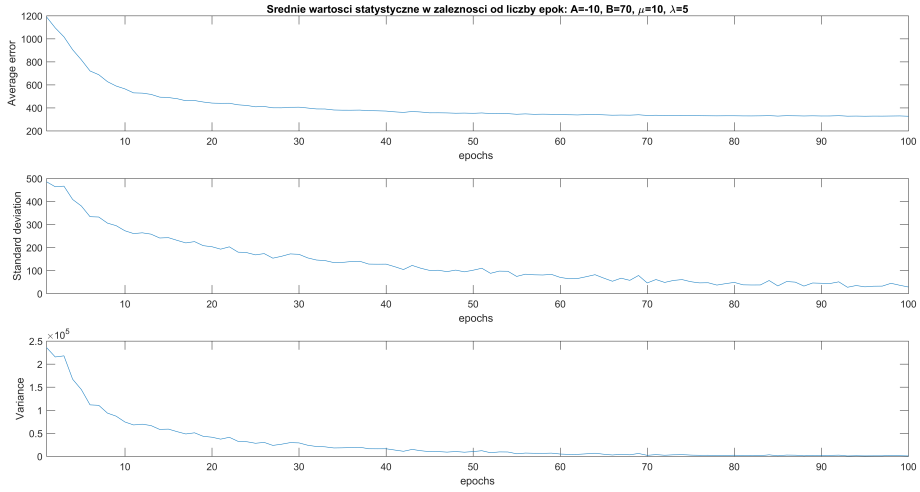
Można zauważyć, że dla dużych wartości μ i λ algorytm szybko znajduje rozwiazanie optymalne. Zajmuje mu tu w zależności od zadanych A i B od 5-20 epok. Największe zmiany średnich wartości widoczne są na początku wykresów (przy pierwszych iteracjach) - średnie wartości gwałtownie spadają. Średnia wartość błędu spada do minimalnej do osiągnięcia wartości w zależności od dostępnych kart oraz zadanych A i B - nie zawsze da się osiągnąć 0. Wariancja i odchylenie po pewnej liczbie iteracji zbiegają do zera.

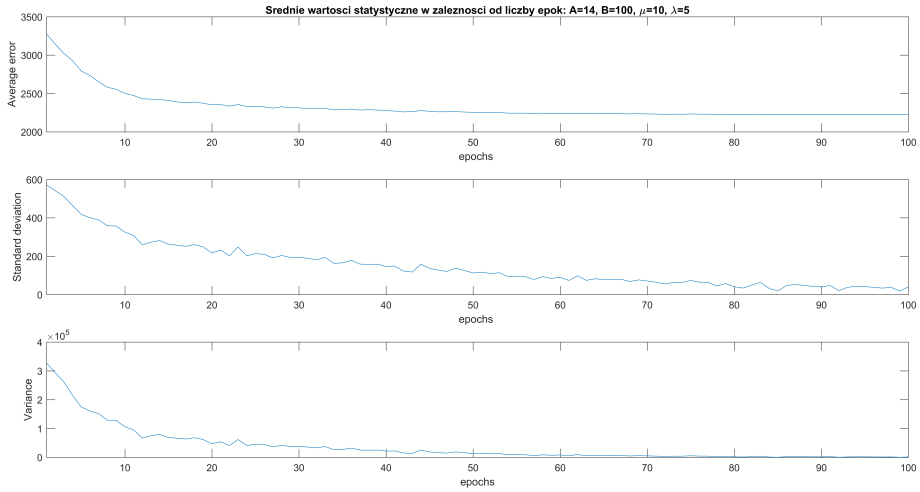
8.4 Małe wartości μ i λ

Przyjęte zostały wartości: $\mu = 10$ i $\lambda = 5$. Przeprowadzono eksperyment dla wartości zadanych A i B:
- A = 5, B = 50
- A = 26, B = 29
- A = -10, B = 70
- A = 14, B = 100

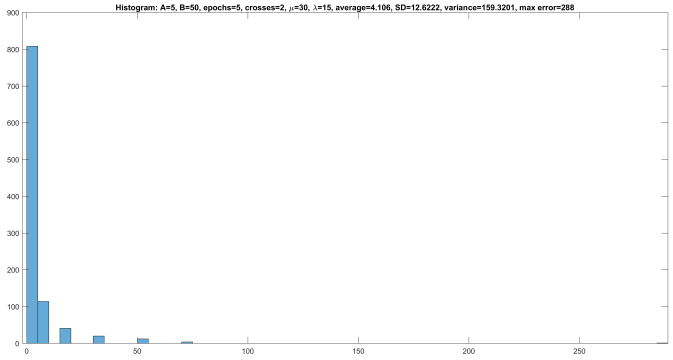


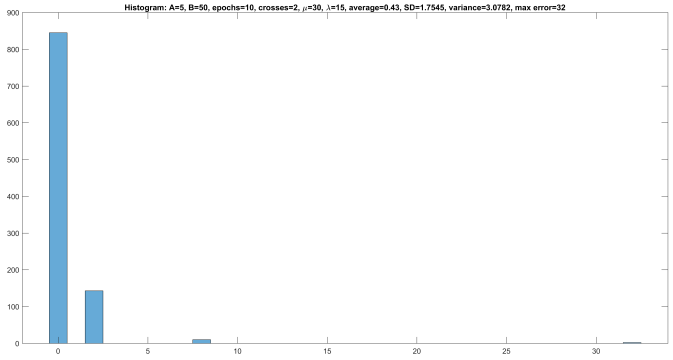






Dla małych wartości μ i λ algorytm radzi sobie zdecydowanie gorzej. Widać większą losowość, co wynika z tego, że mamy mało osobników, czyli mamy mniejszą szansę na uzyskanie czegoś dobrego - tworzymy mniej osobników w danej iteracji. Średnie wartości dużo wolniej spadają do optymalnych rezultatów. Nawet po stosunkowo dużej liczbie iteracji nie mamy wystarczającego prawdopodobieństwa, że osiągnięty rezultat jest najlepszy.

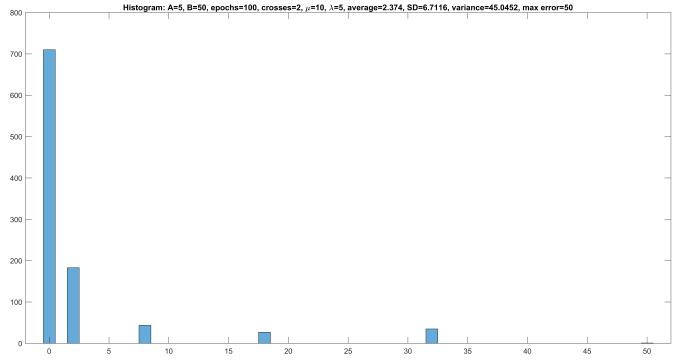


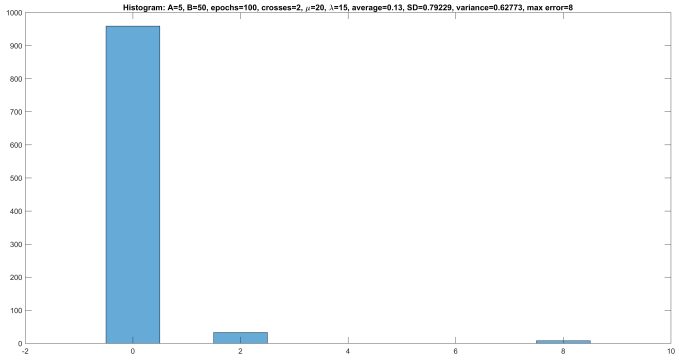


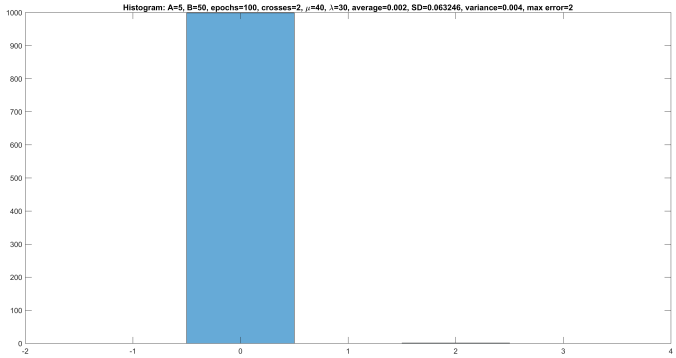
Przy niskiej wartości liczby iteracji ($epochs = 5$) można zauważyć, że jest stosunkowo wysokie odchylenie standardowe. Z powyższego histogramu wynika, że na 1000 zapytań przynajmniej jedno dało rezultat z błędem równym 288. Pomimo tego, że statystycznie 80% zapytań zwróciło optymalne dane, to odchylenie standardowe było zdecydowanie za duże. Już podwójne zwiększenie liczby epok ($epochs = 10$) spowodowało, że odchylenie standardowe zmalało ponad siedmiokrotnie - pomimo tego, że procent optymalnie rozwiazanych zapytań nie wzrósł drastycznie.

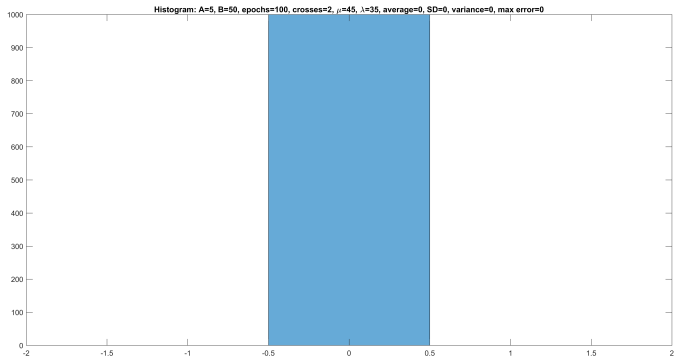
8.5 Testowanie μ i λ

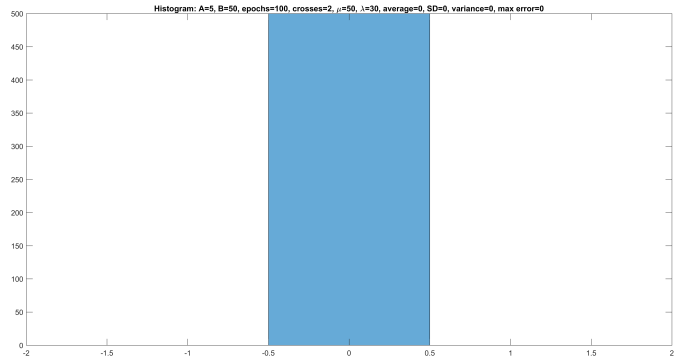
Wszystkie eksperymentry dotyczące badania parametrów μ i λ zostały badane dla A = 5, B = 50, epochs = 100, crosses = 2.


Histogram: A=5, B=50, epochs=100, crosses=2, μ=10, λ=5, average=2.374, SD=6.7116, variance=45.0452, max error=50
800
700
600
500
400
300
200
100
0
0
5
10
15
20
25
30
35
40
45
50


Histogram: A=5, B=50, epochs=100, crosses=2, μ=20, λ=15, average=0.13, SD=0.79229, variance=0.62773, max error=8
1000
900
800
700
600
500
400
300
200
100
0
-2
0
2
4
6
8
10


Histogram: A=5, B=50, epochs=100, crosses=2, μ=40, λ=30, average=0.002, SD=0.063246, variance=0.004, max error=2
1000
900
800
700
600
500
400
300
200
100
0
-2
-1
0
1
2
3
4

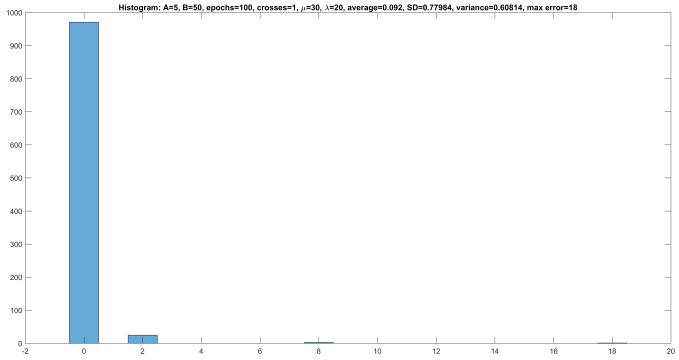


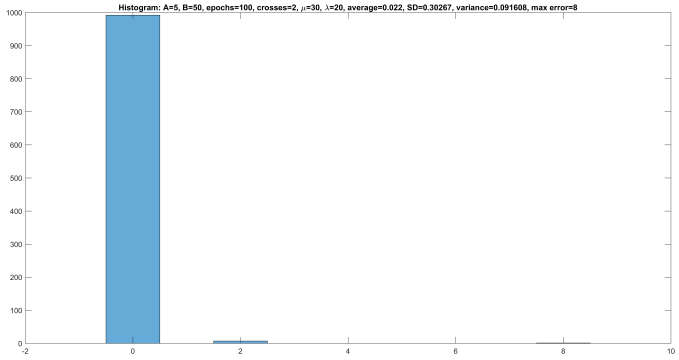


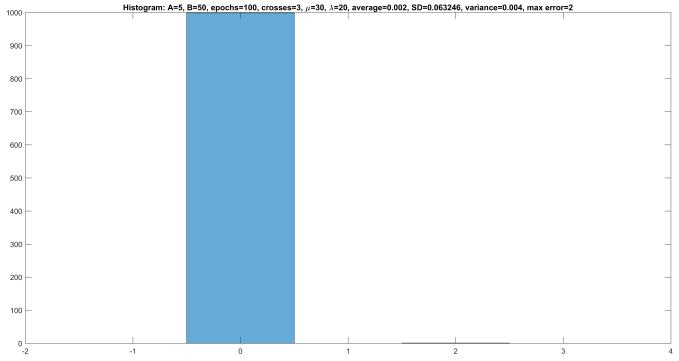
Można zauważyć, że zbyt mała liczba osobników zarówno potomnych jak i rodziców sprawia, że jest mniejsze prawdopodobieństwo uzyskania optymalnego rozwiązania. Przy $\mu = 10$ i $\lambda = 5$ jest bardzo duża losowość. Można zauważyć, że maksymalny błąd jaki wystąpił wynosi 50, co oznacza, że różnica pomiędzy zadanymi a osiągniętymi wartościami na obu kupkach wynosiła 5. Istnieje nieco ponad 70% szans, że znalezione rozwiązanie (dla takich zadanych wartości) będzie optymalne.
Wraz ze wzrostem wartości μ i λ rośnie prawdopodobieństwo znalezienia optymalnego rozwiązania, a także zmniejsza się maksymalny osiągnięty błąd. Dla odpowiednio dużych wartości (zarówno μ i λ) osiągnęliśmy taki sam rezultat dla każdego żądania.

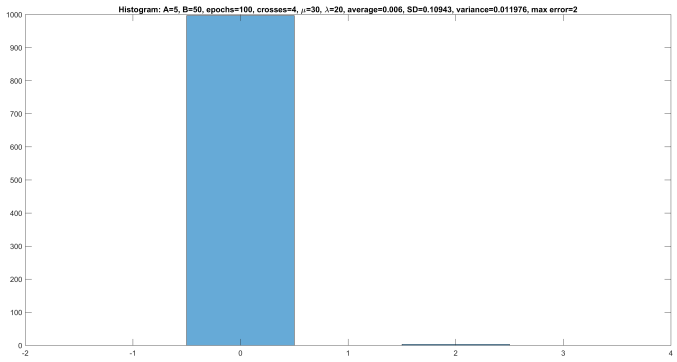
8.6 Testowanie liczby krzyżowań

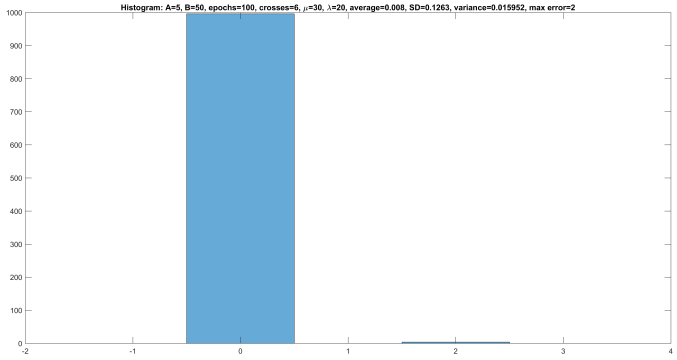
Ostatnim testowanym parametrem jest liczba krzyżowań przy tworzeniu nowego osobnika. Dla kart od 1 do 10 możliwe jest od 1 do 8 krzyżowań (zawsze zaczynamy od "matki").


Histogram: A=5, B=50, epochs=100, crosses=1, μ=30, λ=20, average=0.092, SD=0.77984, variance=0.60814, max error=18
1000
900
800
700
600
500
400
300
200
100
0
-2
0
2
4
6
8
10
12
14
16
18
20


Histogram: A=5, B=50, epochs=100, crosses=2, μ=30, λ=20, average=0.022, SD=0.30267, variance=0.091608, max error=8
1000
900
800
700
600
500
400
300
200
100
0
-2
0
2
4
6
8
10


Histogram: A=5, B=50, epochs=100, crosses=3, μ=30, λ=20, average=0.002, SD=0.063246, variance=0.004, max error=2
1000
900
800
700
600
500
400
300
200
100
0
-2
-1
0
1
2
3
4





Najlepsze rezultaty osiągnięte zostały dla 3 miejsc krzyżowań. Dla zbyt małej liczby krzyżowań osiągnięto stosunkowo duży średni błąd znalezionych rozwiązań. Nie ma stosunkowo dużej różnicy między 3, 4 a 6 miejscami krzyżowań. Opłaca się wziąć mniejszą liczbę krzyżowań, gyż redukuje to ilość operacji wykonywanych przez algorytm - mniejsza liczba losowań: zmniejszenie liczby krzyżowań o 1, zmniejsza liczbę losowań w danej iteracji o μ, dla wszystkich iteracji to $\mu * epochs$ (przykładowo dla $\mu = 20, epochs = 100$ mamy 2000 operacji mniej).

9 Dobrane parametry

Wybrane przez nas parametry dla algorytmu przynoszące zadawalające nas rezultaty wynoszą:

$$\mu = 50, \lambda = 30, crosses = 3, epochs = 100$$

10 Wykorzystywane narzędzia i biblioteki

Dla serwera algorytmu:

- PIPENV

Jest to narzędzie, które pozwala bez problemu zainstalować potrzebne biblioteki w wyizolowanych od siebie środowiskach. Zamiast samemu wyszukiwać w internecie odpowiednich pakietów, które są nam potrzebne, a następnie ręcznie dbać samemu o to, by je aktualizować, możemy wydać po prostu jedno polecenie, które zrobi to za nas - pipenv install. Pipenv pozwala też na tworzenie wirtualnych środowisk. Oznacza to między innymi, że możemy mieć dowolną ilość środowisk wirtualnych, każde z innymi wersjami pakietów przeznaczonych tylko dla danego projektu - polecenie pipenv shell.

Lista potrzebnych bibliotek zapisana jest w pliku Pipfile:

- FLASK

Flask to Pythonowy framework do tworzenia aplikacji internetowych, za pomocą którego stworzyliśmy serwer naszego programu.

- REQUESTS

Jest to biblioteka HTTP napisana w Pythonie, wykorzystywana do komunikacji: przyjmowania żądań i wysyłania odpowiedzi.

- NUMPY

Biblioteka do obliczeń matematycznych: losowanie, obliczenia średniej (wykorzystywana przy plikach testujących).

- SCIPY

Biblioteka do zapisu w pliku .mat (wykorzystywana przy plikach testujących)

Dla serwera klienckiego:
-NODEJS
Środowisko uruchomieniowe języka Javascript, wykorzystywane do serwowania strony klienckiej.

-NPM

Menedżer pakietów środowiska Node.js, pełni podobną rolę co pipenv.

Lista potrzebnych bibliotek jest zapisana w plikach package.json oraz package-lock.json. Najważniejszymi z nich są biblioteki frameworków React oraz Redux

oraz biblioteka do komunikacji z REST API wystawianymi przez serwer algorytmu o nazwie axios.

11 Metodyka testów i wyników testowania

Napisane zostały dwa skrypty do testowania. Każdy z nich zapisywał swoje rezultaty do plików .mat. Na podstawie tych plików zostały wygenerowane wykresy w środowisku *matlab*.
Porównywane były średnia, odchylenie standardowe, wariancja i maksymalny błąd otrzymanych rezultatów.

11.1 spammer.py

Plik ten przyjmował dwa parametry wejściowe A i B i nastepnie dla takich parametrów 1000 razy wysyłał żądanie do serwera algorytmu. Odpowiedzi były dodawane do wektora. Po zakończeniu eksperymentu wektor ten był zapisywany w pliku .mat. A następnie rysowany generowany był z niego histogram. Skrypt wypisywał także na ekran średnią, odchylenie standardowe i wariancję. Program spammer.py był uruchamiany z różnymi wartościami w pliku konfiguracyjnym.

11.2 statistic.py

Plik ten przymował dwa parametry wejściowe A i B. Dla tych danych uruchamiał algorytm 1000 razy dla każdej wartości ilości epok od 1 do 100. Program otwierał plik .json, zmieniał wartość liczby epok, a następnie uruchamiał 1000 razy algorytm. Dla każdej wartości epok obliczał średnią, odchylenie standardowe i wariancję. Do plików zapisywane były te wartości uzyskane dla każdej z epok.