Sprawozdanie 3
Algorytmy Ewolucyjne
MIOwAD

Mateusz Nizwantowski
01161932@pw.edu.pl
313839

11 czerwca 2024

Spis treści

1 Wstęp

W ramach przedmiotu Metody Inżynierii Obliczeniowej w Analizie Danych poznaliśmy temat algorytmów ewolucyjnych. Dla wielu z nas było to pierwsze spotkanie z tą tematyką, jednak nie dla mnie. W ramach przedmiotu Warsztaty Badawcze 1 zgłębiałem tematy związane z metodami optymalizacji dyskretnej i jednym z poznanych sposobów były właśnie algorytmy ewolucyjne. Nie mniej jednak na MIO celem jest rozwiązanie kilku problemów przy użyciu właśnie algorytmów ewolucyjnych. Podobnie jak w przypadku sieci neuronowych i sieci Kohonena co tydzień mieliśmy do wykonania etapy, które budowały na wiedzy z poprzednich tygodni, tym razem były to tylko 3 etapy z czego jeden był dwutygodniowy.
Na końcu naszym finalnym zadaniem jest opisanie naszych doświadczeń w postaci raportu. Przedstawienie co udało nam się osiągnąć, jakie problemy napotkaliśmy, czego się nauczyliśmy i w jaki sposób planujemy rozwijać wiedzę zdobytą podczas tego etapu. Cały kod, który napisałem w ramach tego projektu znajduje się w repozytorium na GitHub.

2 Teoria

Algorytmy ewolucyjne (AE) są klasą algorytmów heurystycznych inspirowanych procesami biologicznej ewolucji. Zostały one zaprojektowane do rozwiązywania problemów optymalizacyjnych, w których tradycyjne metody mogą zawodzić lub być nieefektywne. AE są stosowane w wielu dziedzinach, takich jak inżynieria, ekonomia, informatyka i biologia.

2.1 Podstawowe Koncepcje

AE składają się z populacji rozwiązań, które ewoluują w kierunku optymalnych lub sub-optymalnych rozwiązań problemu. Główne komponenty AE to:

- **Populacja:** Zbiór możliwych rozwiązań problemu.
- **Osobniki:** Pojedyncze rozwiązania w populacji.
- **Selekcja:** Proces wyboru najlepszych rozwiązań do reprodukcji.
- **Krzyżowanie:** Proces tworzenia nowych rozwiązań poprzez łączenie cech dwóch lub więcej rozwiązań.
- **Mutacja:** Proces wprowadzania losowych zmian do rozwiązań.

2.2 Operatory Ewolucyjne

2.2.1 Selekcja

Selekcja ma na celu wybranie najlepszych osobników do kolejnych pokoleń. Popularne metody selekcji to:

- **Selekcja ruletkowa:** Osobniki są wybierane z prawdopodobieństwem proporcjonalnym do ich "jakości".
- **Selekcja turniejowa:** Losowe grupy osobników są wybierane do turniejów, a zwycięzcy turniejów przechodzą do kolejnego pokolenia.
- **Selekcja rankingowa:** Osobniki są sortowane według dopasowania, a prawdopodobieństwo selekcji zależy od rangi.

2.2.2 Krzyżowanie

Krzyżowanie polega na łączeniu dwóch lub więcej rodziców w celu stworzenia potomstwa. Typowe metody krzyżowania to:

- **Krzyżowanie jednopunktowe:** Punkt krzyżowania jest losowo wybierany, a fragmenty rodziców są wymieniane.
- **Krzyżowanie dwupunktowe:** Dwa punkty krzyżowania są losowo wybierane, a segmenty pomiędzy nimi są wymieniane.
- **Krzyżowanie jednorodne:** Każdy gen potomka jest losowo wybierany od jednego z rodziców.

2.2.3 Mutacja

Mutacja wprowadza losowe zmiany w genotypie osobników, aby utrzymać różnorodność genetyczną w populacji. Typowe rodzaje mutacji to:

- **Mutacja punktowa:** Pojedynczy gen jest losowo zmieniany.
- **Mutacja inwersyjna:** Fragment chromosomu jest odwracany.
- **Mutacja przesunięcia:** Geny są przesuwane wzdłuż chromosomu.

2.3 Pseudokod

Algorytm genetyczny jest najpopularniejszym rodzajem AE. Pseudokod AG można następująco:

1. **Inicjalizacja:** Losowa generacja początkowej populacji.
2. **Stop:** Sprawdzenie warunku zakończenia (np. osiągnięcie maksymalnej liczby pokoleń lub satysfakcjonującego rozwiązania).
3. **Krzyżowanie:** Tworzenie nowego potomstwa przez krzyżowanie wybranych osobników.
4. **Mutacja:** Wprowadzanie losowych zmian do potomstwa.
5. **Ewaluacja:** Ewaluacja każdego osobnika w populacji.
6. **Selekcja:** Stworzenie nowej populacji na podstawie obecnej i nowego potomstwa.

3 Eksperymenty

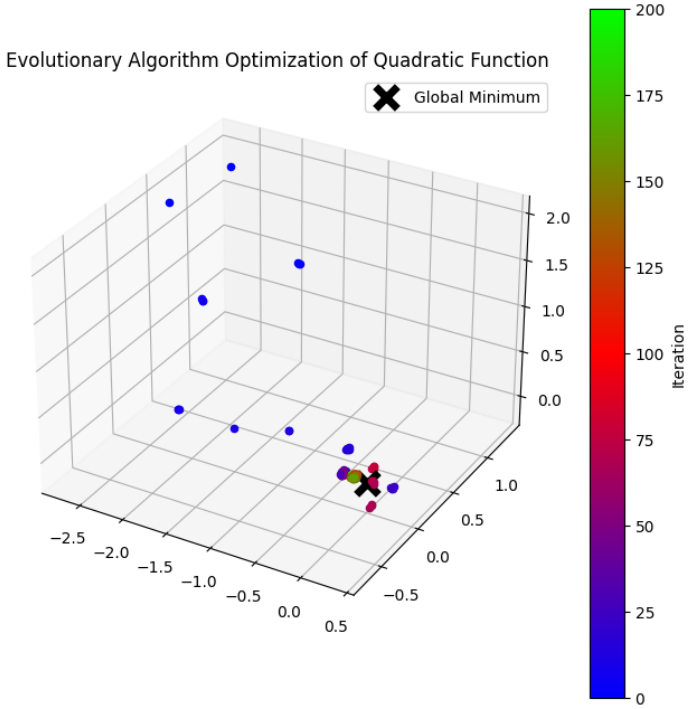
3.1 AE1

Pierwsze zadanie nie było obowiązkowe, w takim znaczeniu, że nie dostawało się za nie żadnych punktów. Ja na szczęście zrobiłem je jak gdyby było oceniane, i z perspektywy czasu uważam, że była to bardzo dobra decyzja. Była to świetna okazja na przypomnienie sobie framework'u algorytmów ewolucyjnych, a dodatkowo pod koniec semestru robi się "gorąco" i każde zadanie, które można zrobić wcześniej to dobry pomysł. I tak należało AE1 wykonać aby móc opisać je w tym raporcie więc nic uważam, żebym zmarnował czas, wręcz przeciwnie. Mam wrażenie, że przez to, że wykonałem te zadanie na początku i napisałem algorytm genetyczny rozwiązujący stosunkowo łatwy problem przedstawiony w AE1 to następne zadania były dla mnie znacząco prostsze gdyż wiedziałem jak do nich się zabrać. A więc co w tym etapie należało wykonać?

Polecenie do tego zadania było wyjątkowo krótkie. Należało napisać podstawowy algorytm genetyczny z mutacją gaussowską i krzyżowaniem jednopunktowym, a następnie przy jego pomocy zoptymalizować 3 wymiarową funkcję kwadratową i 5 wymiarową funkcję Rastrigina.

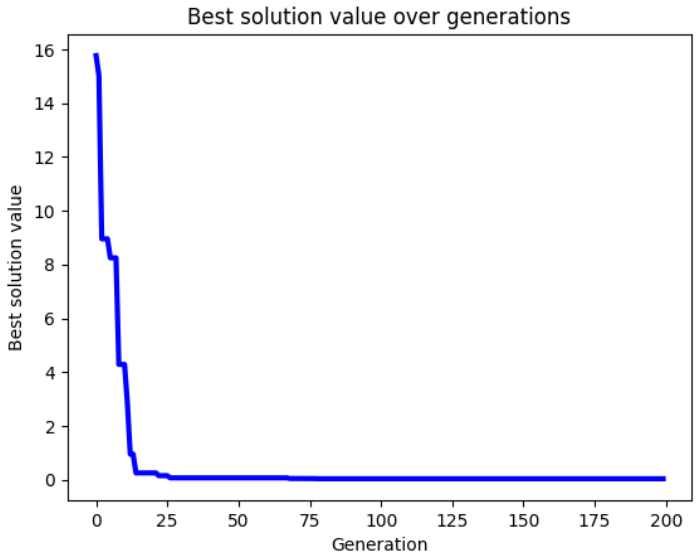
Rozwiązanie tego problemu zajęło 120 linijek kodu, sformatowanego zgodnie z zasadami PIP8. Samo pisanie szło bardzo szybko. Zaimplementowałem losową inicjalizację która polegała na losowaniu jednostajnie z przedziału $[-10, 10]^5$ lub $[-10, 10]^3$ w zależności od wymiaru rozwiązywanego problemu. Mutacja, którą zaimplementowałem to dodanie wektora losowanego z standardowego rozkładu gaussowskiego do rozwiązania. Krzyżowanie jednopunktowe, które zaimplementowałem polegało na wylosowaniu liczby całkowitej z przedziału $[1, wymiar_rozwizania - 2]$ a następnie wzięcie współrzędnych do wylosowanej liczby z rozwiązania pierwszego a reszty z rozwiązania drugiego i analogicznie symetrycznie. Selekcja odbywała się na zasadzie ruletkowej z dodatkową implementacją konceptu *hall of fame*.

Wyniki, które otrzymałem są relatywnie dobre. Nie mniej jednak istnieje nadal pole do popisu. Dla przykładu 3 wymiarowego zwizualizowałem jak wyglądały najlepsze rozwiązania w kolejnych iteracjach.



Rysunek 1: Ewolucja najlepszego rozwiązania w czasie

Najlepszą wartość jaką udało mi się osiągnąć po 200 iteracjach dla zadania optymalizacji funkcji kwadratowej to 0.037 dla rozwiązania [-0.086, -0.075, 0.110], problem jaki napotkałem to fakt, że mutacja, która polegała na dodaniu wektora wylosowanego z rozkładu normalnego miała statyczną wariancję. Na początku dobrze eksplorowałem przestrzeń rozwiązań jednak po czasie gdy znalazłem się w optymalnym punkcie to miałem problem ze zbiegnięciem do minimum. Tak wygląda dla tego problemu wartość minimum w czasie:



Rysunek 2: Wartość minimum w czasie dla funkcji kwadratowej

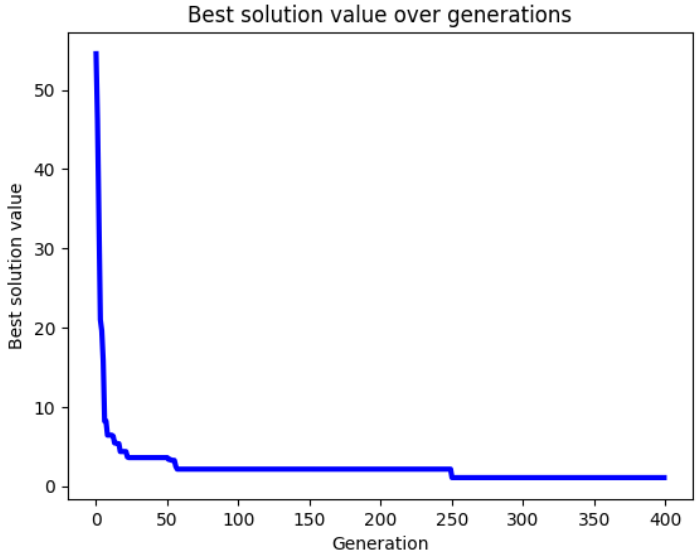
Następny przykład z funkcją Rastrigina był znacznie cięższy do optymalizacji niż funkcja kwadratowa. Funkcja Rastrigina jest często używana jako funkcja testowa w optymalizacji i w ogólnym przypadku możemy ją zapisać następująco:

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Dokładniej w poleceniu jest zaznaczone, że mamy rozważyć przypadek pięciowymiarowy ($n = 5$):

$$f(\mathbf{x}) = 50 + \sum_{i=1}^{5} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Niestety, tego przykładu nie da się zwizualizować tak samo jak poprzedniego ze względu na zbyt duży wymiar. Wynik jaki udało mi się osiągnąć w tym problemie po 400 generacjach to 1.06 dla rozwiązania [0.000, 0.043, 0.058, 0.015, 0.001]. Problemem podobnie jak poprzednio jest statyczna wartość wariancji w mutacji. Wykres minimum w czasie jest zbliżony do tego dla funkcji kwadratowej i wygląda następująco:



Rysunek 3: Wartość minimum w czasie dla funkcji Rastrigina

3.2 AE2

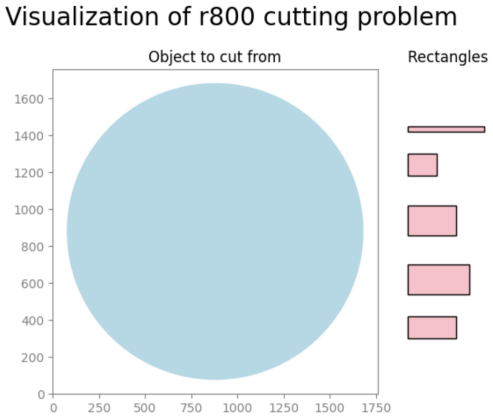
3.2.1 Wstęp

To zadanie było jednym z najtrudniejszych wyzwań z jakim przyszło mi się zmagać na tym przedmiocie. Z racji skomplikowania na ten etap przeznaczone były dwa tygodnie. Podczas pierwszego mieliśmy przemyśleć i spisać naszą koncepcję rozwiązania zadanie: jak ma wyglądać mutacja, krzyżowanie i jak chcemy reprezentować. Drugi tydzień miał być przeznaczony na zaimplementowanie wskazanego wcześniej rozwiązania. A co było do zrobienia?

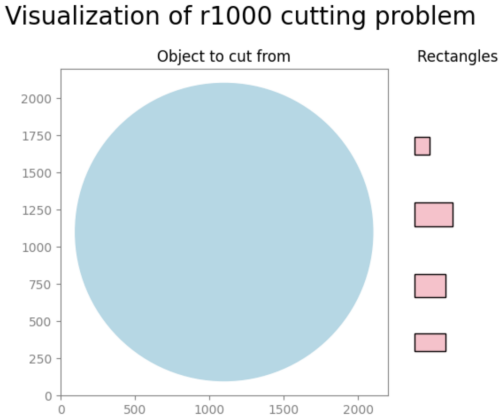
Problem, który musieliśmy rozwiązać w literaturze znany jest pod nazwą cutting stock problem. Mamy dane koło o promieniu r oraz zbiór dostępnych prostokątów zadanych przez trzy liczby: wysokość, szerokość i wartość. Celem jest ułożenie prostokątów w kole tak, aby zmaksymalizować sumę ich wartości, spełniając warunki:

1. boki wszystkich prostokątów muszą być równoległe do osi układu (prostokąt można obrócić o 90°),
2. prostokąty nie mogą na siebie nachodzić ale mogą stykać się bokami,
3. każdy prostokąt można wstawić dowolnie wiele razy.

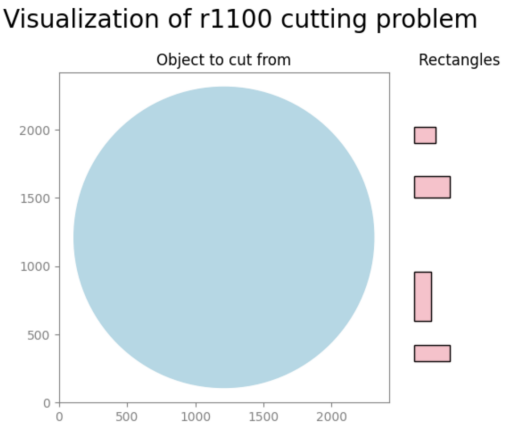
Jest to klasyczny cutting stock problem, który nie posiada żadnych dodatkowych ograniczeń np. że każdy z prostokątów trzeba użyć co najmniej n razy. Następnie po zaimplementowaniu rozwiązania należało je przetestować na dostarczonych problemach. Do każdego z nich wyznaczony jest pułap sumy, którą należy uzyskać aby otrzymać punkty. Problemy wyglądały następująco:



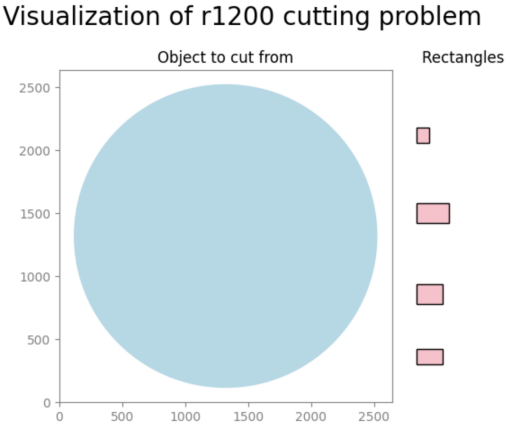
(a) Dla problemu r800 pułap wynosi 30k



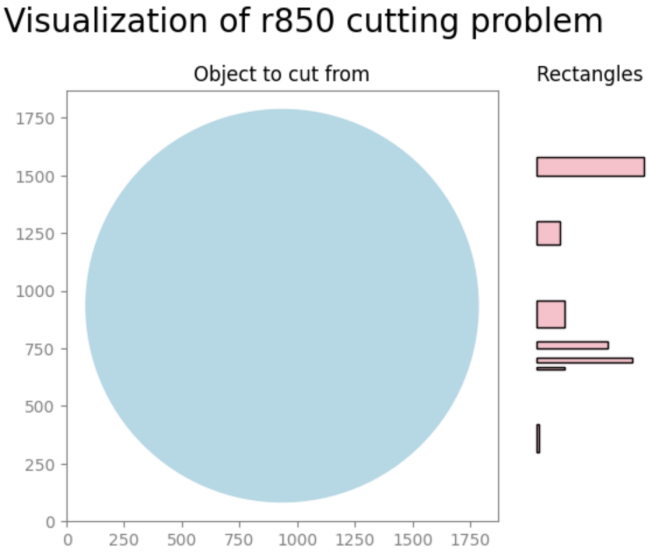
(b) Dla problemu r1000 pułap wynosi 17.5k



(c) Dla problemu r1100 pułap wynosi 25k



(d) Dla problemu r1200 pułap wynosi 30k



Rysunek 4: Problem r850 nie posiada pułapu ale trzeba zaprezentować dla niego rozwiązanie

3.2.2 Pierwotny pomysł na rozwiązanie

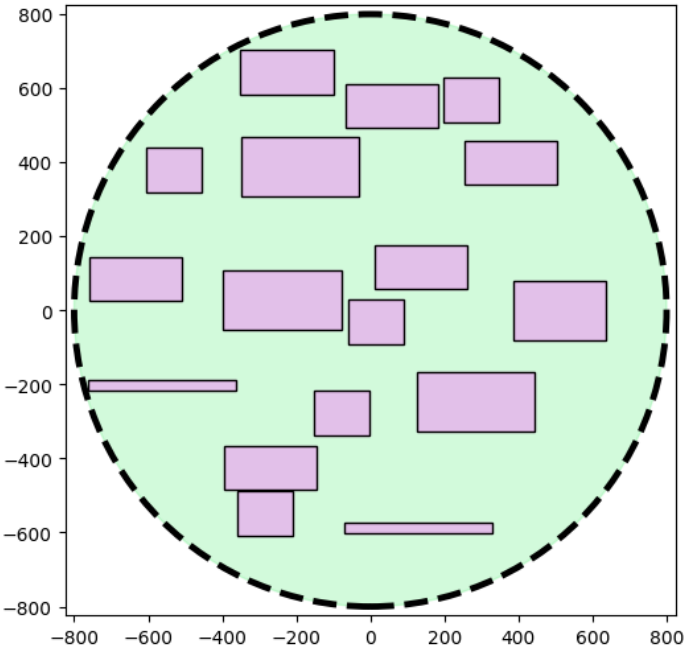
Pierwotnie mój pomysł wyglądał następująco:

Za potencjalne rozwiązanie przyjąłem listę która przechowuje prostokąty. Prostokąt to lista długości 5 której elementy znaczą kolejno: położenie_x lewego dolnego rogu, położenie_y lewego dolnego rogu, szerokość, wysokość, wartość.

- **Inicjalizacja**: n prób wstawienia losowego prostokąta w losowe miejsce (o ile to możliwe, n jest parametrem)
- **Mutacja**: wybranie prostokąta z koła i przesunięcie go w prawo i w górę najdalej jak się da lub próba dodania jakiegoś prostokąta
- **Krzyżowanie**: wybranie poziomej lub pionowej linii i "krzyżowanie" połówek z usunięciem nachodzących na siebie prostokątów oraz uzupełnieniem pustych miejsc powstałych w ten sposób
- **Warunek stopu**: wykonanie określoną z góry ilość iteracji lub spełnienie określonego pułapu
- **Selekcja**: do kolejnej iteracji przechodzą osobniki na zasadzie wyboru ruletkowego, a na końcu wybierany jest najlepszy z osobników

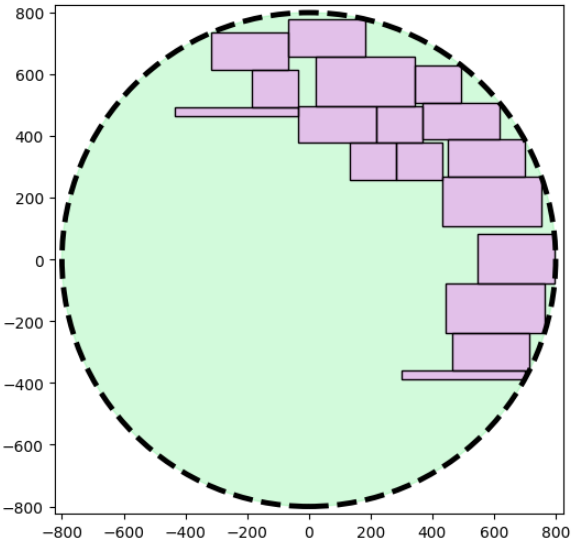
3.2.3 Realizacja

Przystąpiłem do realizacji tego pomysłu, najpierw napisałem inicjalizacje: tak wygląda przykładowo wygenerowany osobnik:



Rysunek 5: Przykładowy osobnik po inicjalizacji

Następnie napisałem mutacje która losowo wybiera prostokąt i przesuwa go w prawo lub w górę, i przetestowałem ją na wykonując ją 10000 razy na osobniku znajdującym się na obrazku powyżej:

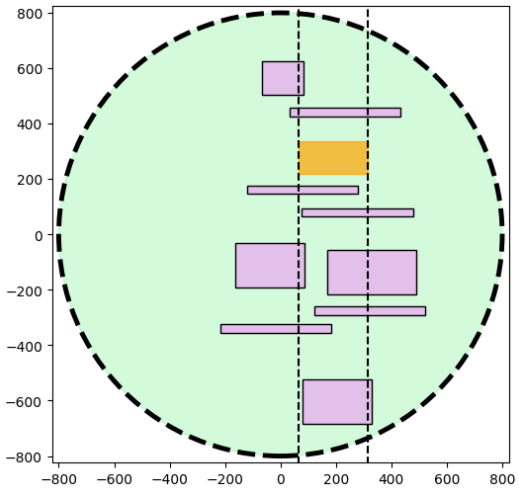


Rysunek 6: Osobnik po wykonaniu na nim 10000 mutacji

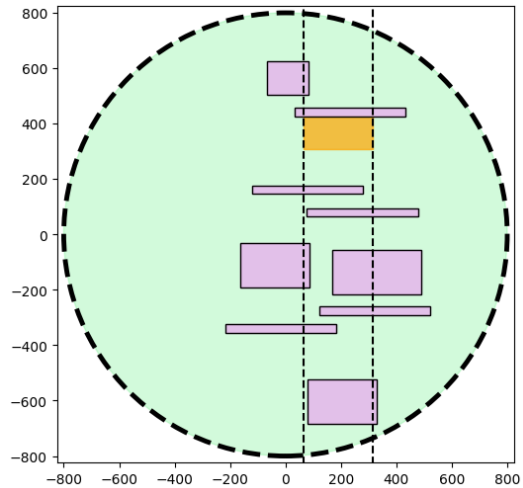
Wyniki podczas gdy zadowalające, pomiędzy prostokątami istnieją szczeliny, których minimalizacja pozwoliła by mi na osiągnięcie lepszego wyniku finalnego. Zacząłem się więc zastanawiać co tworzy szczeliny. Jest to losowanie na początku dużej ilości prostokątów co powoduje że zaczynają się one zakleszczać i blokować. Dlatego zmodyfikowałem mój pomysł i teraz generuję na początku tylko jeden prostokąt i następnie mutuję go przestawiając go najbardziej na lewo i w górę jak się da a następnie powtarzam tą operacje. Aby dodatkowo zoptymalizować ten proces na początku wybieram najbardziej opłacalne (z najlepszą ceną za jednostkę powierzchni)

3.2.4 Jak działa mutacja

Wybieram losowo jeden z prostokątów (jest on zaznaczony na rysunkach na żółto) i wyznaczam wszystkie prostokąty z którymi mógł by on potencjalnie kolidować przy przesunięciu go pionowo/poziomo (na rysunkach pokazana jest opcja z przesunięciem pionowym). Następnie znajduję granicę do której mogę przesunąć prostokąt w górę (może to być inny prostokąt lub kraniec koła) i przesuwam go tam, co zostało pokazane na rysunku. Z tym krokiem było trochę problemów gdyż ze względu na niedokładność reprezentacji liczb w pamięci komputera, zdarzało się, że przesunięty prostokąt nieznacznie zaczął nachodzić na inny prostokąt i wtedy wszystko się psuło. Należało wziąć to pod uwagę i dołączyć małe marginesy aby temu zapobiec.



(a) Sytuacja przed przesunięciem



(b) Sytuacja po przesunięciu

3.2.5 Finalny pomysł

Za potencjalne rozwiązanie przyjąłem listę która przechowuje prostokąty. Prostokąt to lista długości 5 której elementy znaczą kolejno: położenie_x lewego dolnego rogu, położenie_y lewego dolnego rogu, szerokość, wysokość, wartość.

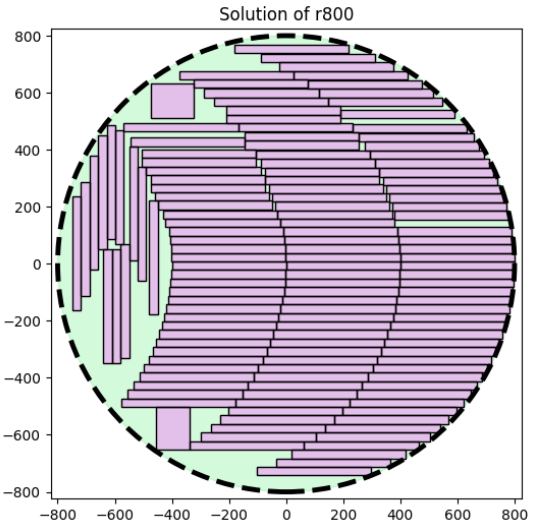
- **Inicjalizacja**: utworzenie n struktur, jednakże na tym etapie nie będę zapełniał ich prostokątami
- **Mutacja**: losowe wstawienie prostokąta o ile to możliwe i przesunięcie go w prawo i w górę, metodą opisaną powyżej
- **Krzyżowanie**: brak krzyżowania
- **Warunek stopu**: wykonanie określoną z góry ilość iteracji
- **Selekcja**: do kolejnej iteracji przechodzą wszystkie osobniki, a na końcu wybierany jest najlepszy z osobników

Ten pomysł na rozwiązanie jest znacznie prostszy niż klasyczny algorytm ewolucyjny (jeżeli ma się już zaimplementowanie przesuwanie), liczy się bardzo szybko i nie ma problemów z osiągnięciem określonych pułapów. Nazwał bym go jednak algorytmem para ewolucyjnym, gdyż czerpie z teorii algorytmów ewolucyjnych to nie implementuje krzyżowania czy selekcji.

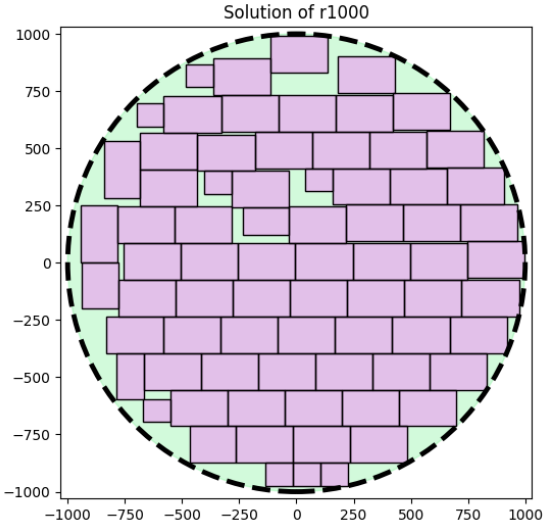
3.2.6 Wyniki

Wyniki tej metody są bardzo dobre, jak zawsze jest miejsce do poprawy jednak nie ma go dużo. Postanowiłem, że oprócz porównywać się do pułapu ustalonego przez prowadzących, będę również odwoływał się do maksymalnej możliwej do osiągnięcia wartości obliczanej w sposób następujący:

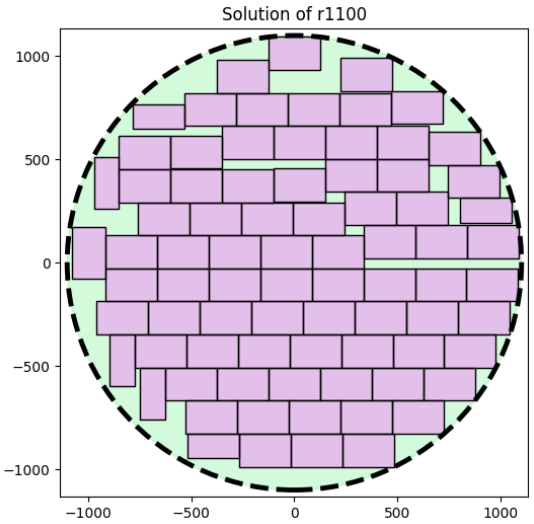
$$\text{Teoretyczne maximum} = 2\pi^2 \times \max_{p:\text{prostokąty}} \left(\frac{\text{wartość p}}{\text{pole p}}\right)$$



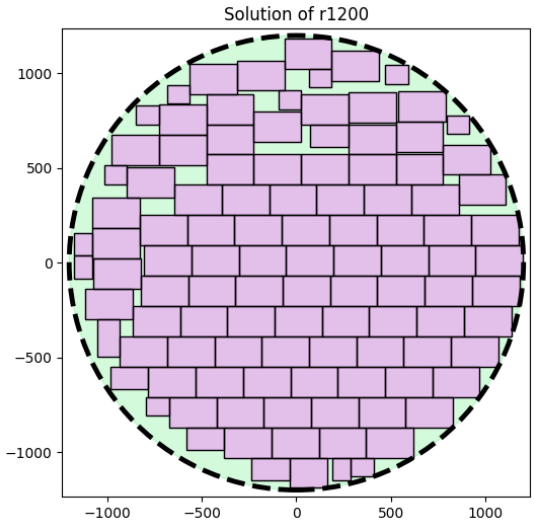
(c) Osiągnięty wynik: 54 480
Teoretyczne maximum: 67 021, osiągnięte 81.29%
Wymagana wartość: 30 000, osiągnięte 181.60%



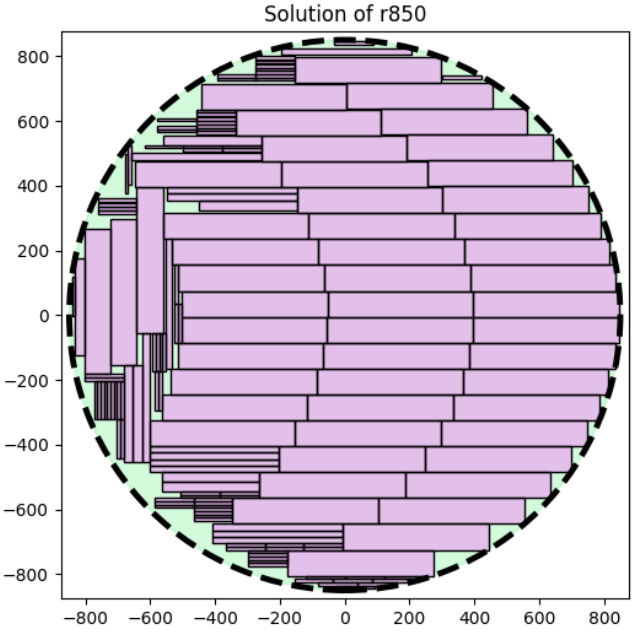
(d) Osiągnięty wynik: 32 320
Teoretyczne maximum: 39 270, osiągnięte 82.30%
Wymagana wartość: 17 500, osiągnięte 184.69%



(e) Osiągnięty wynik: 45 000
Teoretyczne maximum: 57 020, osiągnięte 78.92%
Wymagana wartość: 25 000, osiągnięte 180.00%



(f) Osiągnięty wynik: 46 780
Teoretyczne maximum: 56 549, osiągnięte 82.73%
Wymagana wartość: 30 000, osiągnięte 155.93%



Rysunek 7:
Osiągnięty wynik: 586 960
Teoretyczne maximum: 693 550, osiągnięte 84.63%
Wymagana wartość: brak

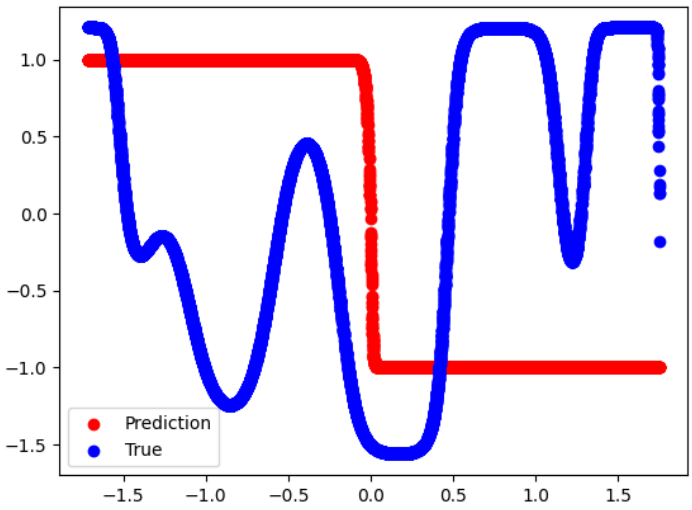
Jak widać na załączonych obrazkach w zadaniach, w których był ustalony pułap do osiągnięcia, moje rozwiązania osiągają ponad półtora krotność pułapu, jednak we większości przypadków jest to wartość większa o 80% od ustalonego pułapu. Wyjątkowo wymagający (w kontekście zadania) okazał się problem r1200 w którym pomimo osiągnięcia 82.73% teoretycznej wartości maksymalnej, jest większy od pułapu o jedynie 55.93%. Przedstawione przeze mnie rozwiązania oscylują woków 81% teoretycznej wartości maksymalnej, co uważam za dobry wynik. Najlepsze rozwiązanie znalazłem dla problemu r850, gdzie osiągnąłem prawie 85% wartości teoretycznej. Warto zaznaczyć, że otrzymane przeze mnie wyniki można by poprawić bardziej standardowym i klasycznym algorytmem genetycznym.

3.3 AE3

AE3 było dla mnie wymagające, raczej nie przez samą tematykę jednak przez natłok rzeczy do zrobienia pod koniec semestru. Ten etap to połączenie kodu z sieci neuronowych i AE1. Wyobrażam sobie, że dla kogoś kto nie wykonał AE1 lub nie napisał modularnego kodu w NN to zadanie mogło sprawić spore problemy. Próbowałem je zrobić je jeszcze przed oficjalnym terminem na drugą część AE2, aby niepotrzebnie odkładać sobie obowiązki na zakończenie semestru, jednak moje wyniki były niesatysfakcjonujące i chciałem je poprawić, jednak nie miałem wystarczająco czasu. Oto co udało mi się zrobić:

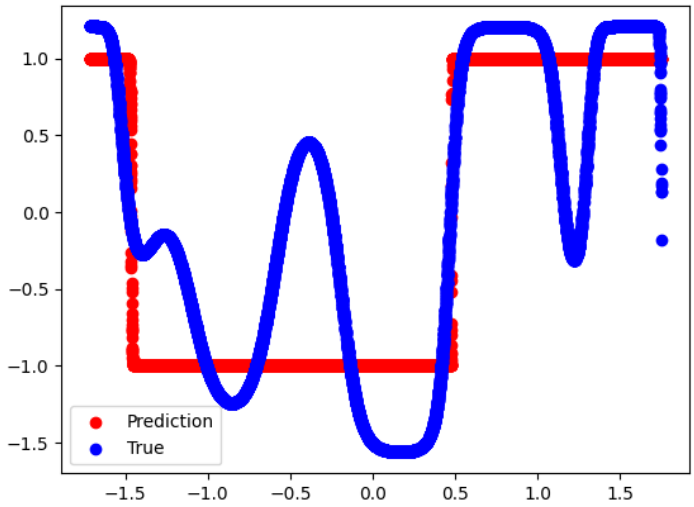
- napisać kod, który działa,
- przetestować, poprawność rozwiązania na, z tego co słyszałem, najtrudniejszym problemie na, którym nie powinienem się spodziewać fantastycznych wyników,
- eksperymentować z różnymi hiperparametrami algorytmu genetycznego i z różnymi architekturami sieci.

Podobnie jak w AE1 używam, krzyżowania jednopunktowego, a mutacja to losowanie wektora z rozkładu normalnego, (tym razem z dynamiczną wariancją) i dodanie go do mojego rozwiązania. Na początku generuje losowo osobniki, testowałem różne funkcje inicjalizacji, jednak teraz nie są one tak istotne jak w NN gdyż, nie używamy propagacji wstecznej tylko optymalizujemy, przy użyciu algorytmów ewolucyjnych. Jedyne o co należy zadbać to aby wagi nie były zbyt małe, gdyż ciężej wtedy jest przeszukiwać przestrzeń rozwiązań. Początkowe przykładowe rozwiązanie dla problemu multimodal large wygląda następująco:



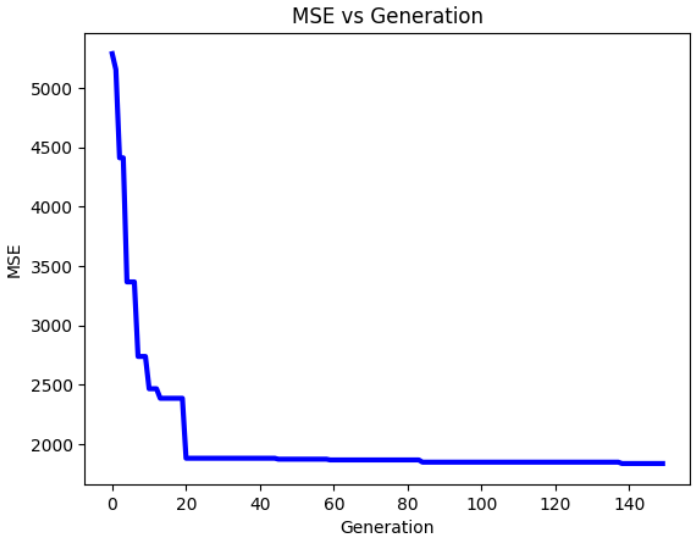
Rysunek 8: Losowe rozwiązanie dla problemu multimodal

Mój algorytm działa, to znaczy optymalizuje, jednak nie robi tego najlepiej. Wydaje mi się, że jest to spowodowane słabym przeszukiwaniem przestrzeni rozwiązań. Dodanie funkcjonalności bezwarunkowego przechodzenia najlepszego osobnika do kolejnej generacji znacznie pomogło, i spowodowało, że nie zapominam najlepszych rozwiązań. Mam wrażenie, że w tym problemie domyślna selekcja metodą ruletkową słabo działa. Po pewnym czasie wyniki osobników są do siebie zbliżone i wtedy praktycznie losujemy kolejną generację jednostajnie. Oto wyniki optymalizacji:



Rysunek 9: Najlepsze rozwiązanie po algorytmie ewolucyjnym: MSE = 1805

Osobnikiem w tym eksperymencie jest siec o rozmiarze [1, 10, 10, 1]. Z moich testów ta konfiguracja radzi sobie dobrze, jednak mniejsze sieci takie jak [1, 10, 1], czy [1, 5, 5, 1], też się optymalizują. Z pewnością dało by się znaleźć lepsze rozwiązanie, jednak to wymaga czasu, o który ciężko pod koniec semestru. Na koniec pokażę jeszcze wykres jak zmienia się najlepsze rozwiązanie w czasie, da to wgląd na zachowanie algorytmu:



Rysunek 10: Jak zmienia się MSE w kolejnych generacjach

Pomimo, że rozwiązanie działa, to optymalizowanie danego problemu wymaga poświęcania dużej ilości czasu na dostrajanie hiperparametrów. Z wsteczną propagacją błędu nie było takich problemów, po prostu odpalało się kod i otrzymywało się bardzo dobre rozwiązanie, co więcej otrzymywało się je szybko. Z racji sekwencyjnej natury algorytmów ewolucyjnych, ciężko zrównoleglić ich działanie co powoduje, że znalezienie przez nie rozwiązania zajmuje bardzo dużo czasu. Są to ogromne wady i nie bez powodu używa się propagacji wstecznej do optymalizowania parametrów sieci neuronowych. Nie mniej jednak było to ciekawe doświadczenie które, rzuciło nowe światło na moje zrozumienie zarówno sieci neuronowych jak i algorytmów ewolucyjnych.

4 Podsumowanie

Algorytmy ewolucyjne są potężnym narzędziem do rozwiązywania złożonych problemów optymalizacyjnych. Dzięki inspiracji procesami biologicznymi, AE oferują elastyczność i efektywność w wielu zastosowaniach. Co do mojego projektu to oceniam go za udany. Nie był on tak bardzo czasochłonny jak sieci neuronowe czy sieci Kohonena i według mnie był prostszy, może jest to kwestia, że miałem już wcześniej styczność z algorytmami ewolucyjnymi. Nie mniej jednak nauczyłem się niezwykle dużo i bez wątpienia poszerzyłem swoją wiedzę z zakresu algorytmów ewolucyjnych. Zadania, nie wliczając AE3 na które zabrakło mi czasu oraz w którym słabe wyniki mnie frustrowały, uważam za wykonane poprawne, otrzymywałem relatywnie dobre wyniki. Jestem zadowolony z tego co zrobiłem, a robiąc to, dobrze się bawiłem (poza AE3, na które nie miałem wystarczająco czasu).