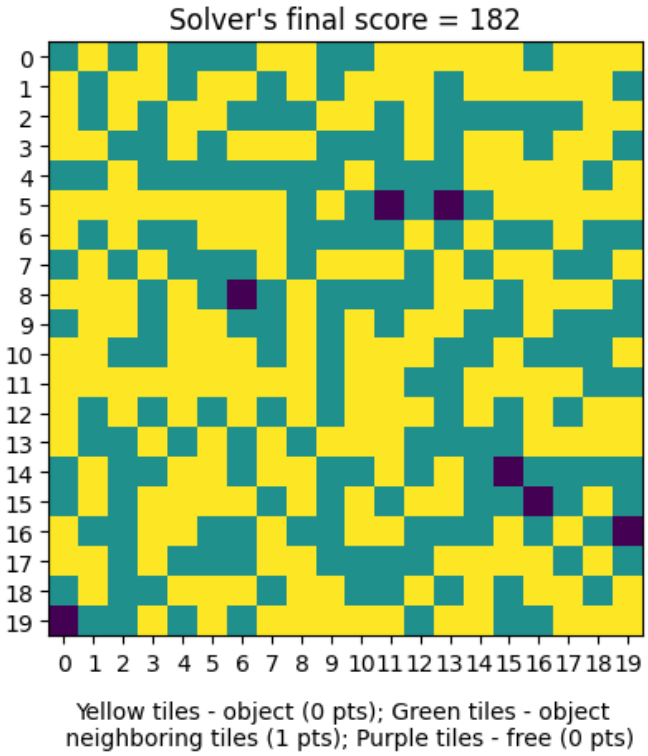
WSI - ćwiczenie 2.
Algorytmy ewolucyjne i genetyczne

Jakub Romankiewicz 325063

1. Rozwiązanie losowe

Pracę rozpoczęto od przygotowania rozwiązania losowego omawianego problemu. W tym celu utworzono klasę RandomSolver() z funkcją solve() zwracającą wektor o zadanej długości z losowo aktywowanymi polami. Prawdopodobieństwo aktywacji pola wynosi 50%.
Następnie przykładowe losowe rozwiązanie oceniono daną funkcją evaluate() i zwizualizowano:



Rys. 1 – Przykładowe rozwiązanie losowe problemu z wielkością pola 20x20. Wynik = 182, pod wykresem wyjaśniono znaczenie poszczególnych kolorów.

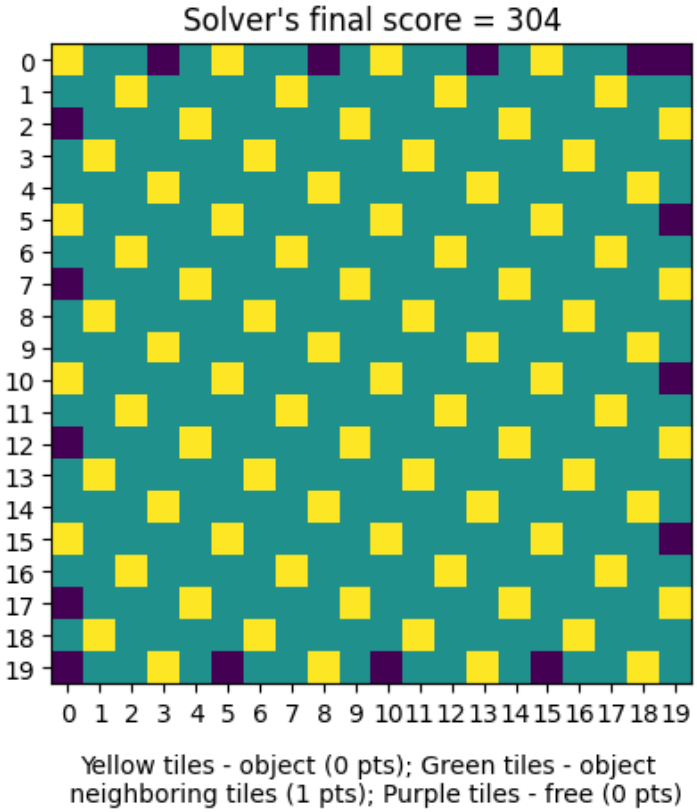
Następnie zbadano średni wynik takiego algorytmu wyliczając wynik z 25 itereacji.

```
Average points earned: 184.44
```

Użyta tutaj liczba iteracji równą 25 będzie stosowana dla reszty eksperymentów z uśrednianiem wyników (czyli takich w których losowość pełni ważną rolę w końcowym wyniku).

2. Próba znalezienia trywialnego rozwiązania

Podjęto próbę znalezienia rozwiązania, które intuicyjnie wydawało się najprostsze. Zastosowano układ, w którym obiekty rozmieszczone są na planszy w sposób przypominający ruchy konia w szachach. Takie ustawienie wydawało się autorowi najbardziej optymalne.



Rys. 2 – Trywialne rozwiązanie problemu wykonane przez autora. Wynik = 304 punkty

Kod realizujący takie rozmieszczenie pól to:

```
solution = np.zeros(400)
size = 20
off = 0
for row in range(size):
    for col in range(size):
        if (col+off) % 5 == 0:
            solution[row*size+col] = 1
    off += 3
```

Osiągnienty tu wynik będzie punktem odniesienia w następnych eksperymentach.

3. Podstawowy algorytm genetyczny i jego problemy

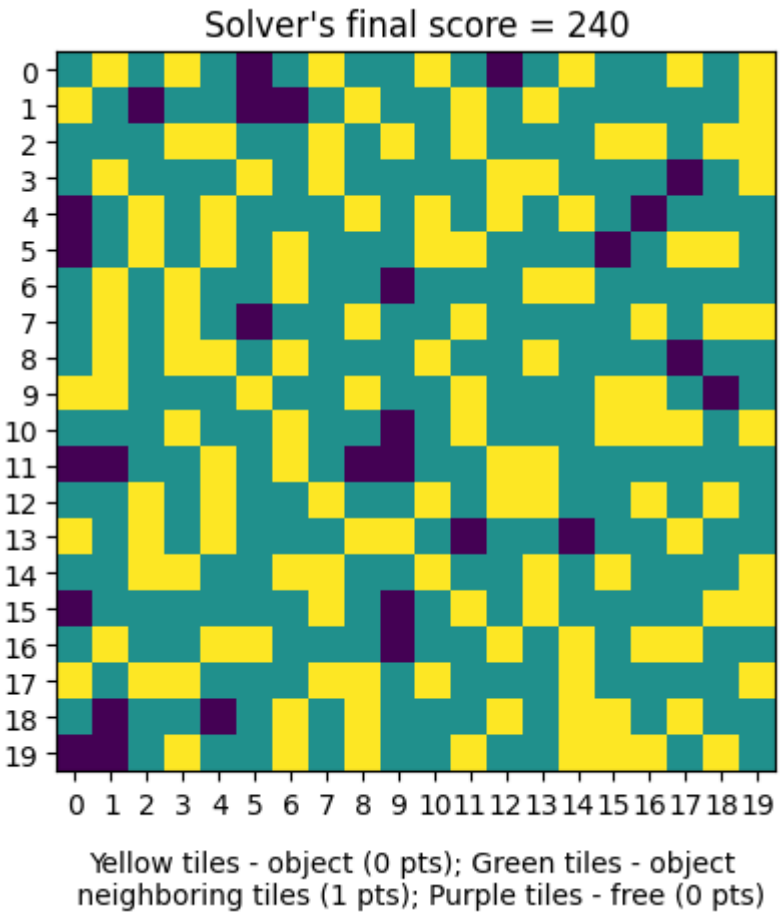
Utworzono klasę GeneticSolver posiadającą funkcję solve() realizującą algorytm genetyczny z mutacją, selekcją ruletkową, krzyżowaniem jednopunktowym oraz sukcesją generacyjną. Funkcja jako jeden z argumentów przyjmuje instancję opisanej wcześniej klasy RandomSolver (W ten sposób algorytm może działać dla dowolnej wielkości planszy*), służy ona do inicjalizacji osobników z pierwszego pokolenia.

Zainicjalizowano hiperparametry:

- T_MAX = 400
- MU = 2000
- PC = 0.6
- PM = 0.001

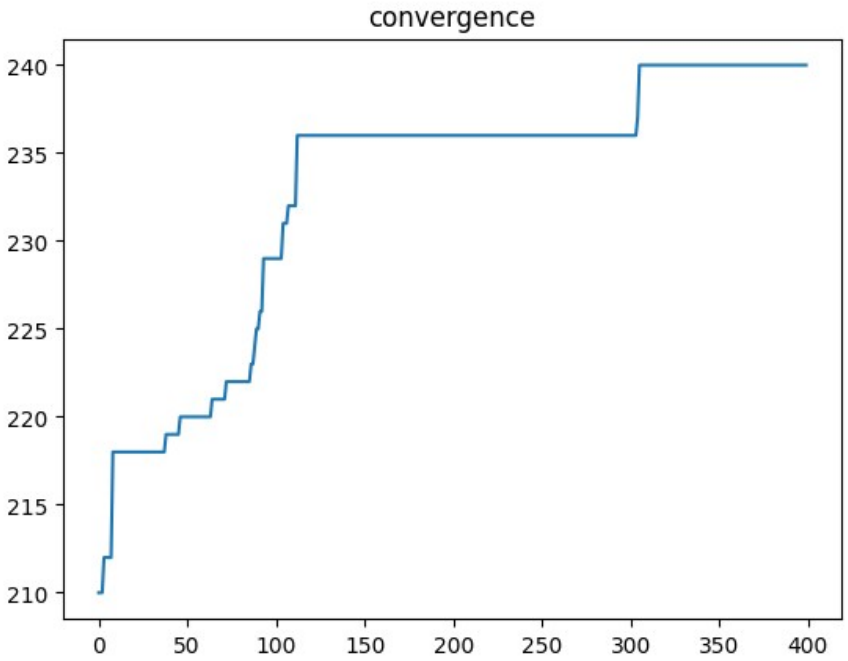
Parametry i ich dobór zostaną omówione dogłębnie w podpunkcie 5.

Przykładowy wynik działania algorytmu:



Rys. 3 – Przykładowe rozwiązanie problemu algorytmem genetycznym z wielkością pola 20x20. Wynik = 240. Jest to wynik dużo lepszy od rozwiązania losowego, jednak wciąż nie tak dobry jak manualnie opracowane rozwiązanie

** przykładowa implementacja planszy innego rozmiaru niż 20x20 na końcu nodebooka testing.ipynb*



Rys. 4 – Konwergencja wyniku przykładowego rozwązania problemu algorytmem genetycznym. Widać, że algorytm zatrzymuje się na wiele pokoleń.

Uśredniony wynik działania algorytmu:

```
Avg points earned (25 iterations): 238.72
```

4. Algorytm genetyczny z ulepszoną selekcją ruletkową

Problemem okazała się implementacja selekcji ruletkowej. W obecnej każdy osobnik miał szansę na przejście do następnego pokolenia równą swojej ilości punktów podzielonej przez sumę punktów wszystkich osobników. Okazało się to problemem ponieważ przykładowy najlepszy osobnik z pierwszego pokolenia (dane z wykresu konwergencji z punktu 3.) uzyskał około 210 punktów, podczas gdy średni osobnik uzyskał około 184 punkty (dane ze średniego losowego rozwiązania z punktu 1). Dodajmy do tego jeszcze słabego osobnika, który uzyskał 170 punktów (przykład) i zobrazujmy ich hipotetyczne szanse.

os. a = 210 pkt; os. b = 184 pkt; os. c = 170 pkt

suma = 564 pkt

w takim razie:

p(a) = 210 pkt / 564 pkt ~ 0,37; p(b) ~ 0,33; p(c) ~ 0,30

Jak widać szanse przejścia każdego z osobników są podobne, przez co algorytm przepuszcza bardzo wiele słabszych osobników w miejsce lepszych.

Algorytm ulepszono odejmując od wyniku każdego osobnika offset, równy wynikowi najgorszego osobnika. W powyższym przykładzie byłoby to:

min = os. c = 170 pkt

os. a = 210 pkt – min = 210 pkt – 170pkt = 40 pkt;

os. b = 184 pkt – min = 14 pkt;

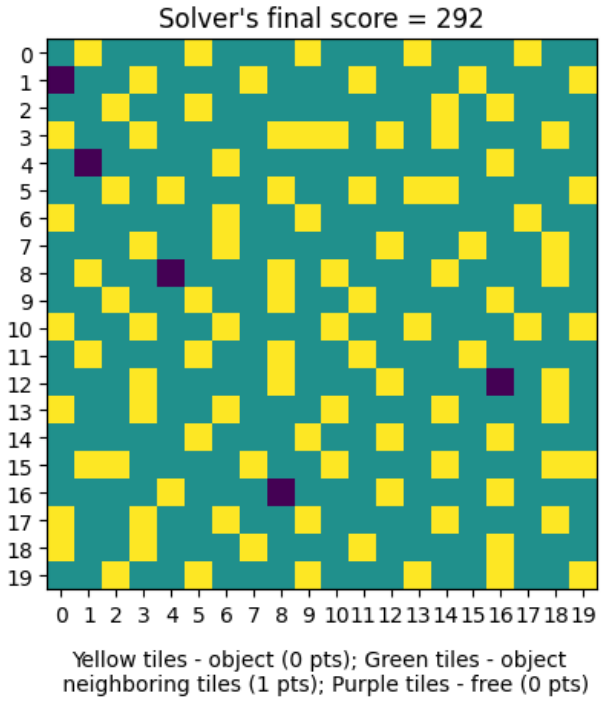
os. c = 170 pkt – min = 0 pkt

suma = 54 pkt

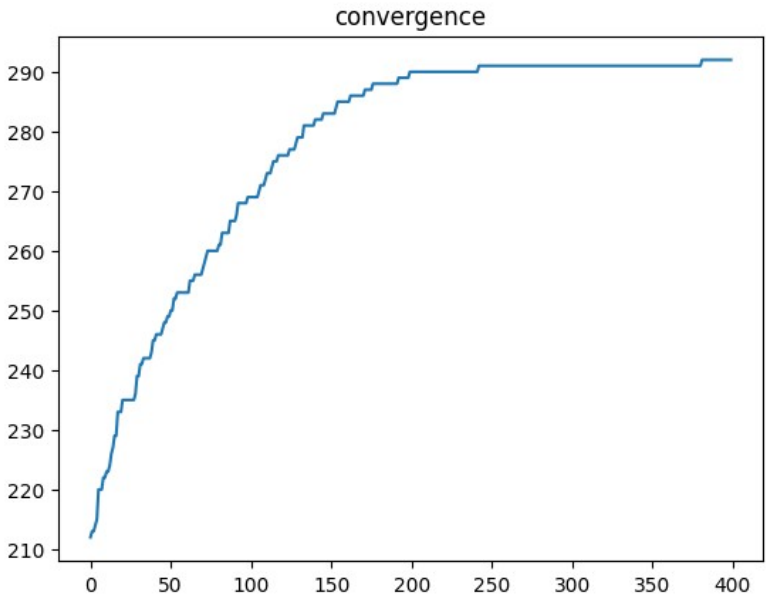
w takim razie:

p(a) = 40 pkt / 54 pkt ~ 0,74; p(b) ~ 0,26; p(c) = 0

Wyniki ulepszonego algorytmu:



Rys. 5 – Przykładowe rozwiązanie problemu ulepszonym algorytmem genetycznym z wielkością pola 20x20. Wynik = 292. Jest to wynik bliski rozwiązaniu manualnemu.



Rys. 6 – Konwergencja wyniku przykładowego rozwązania problemu ulepszonym algorytmem genetycznym.

Uśredniony wynik działania algorytmu:

```
Avg points earned (25 iterations): 290.76
```

5. Badanie wpływu hiperparametru

Zbadano wpływ prawdopodobieństwa mutacji (parametr PM) na średni wynik ulepszonego algorytmu przy następujących pozostałych hiperparametrach:

- T_MAX = 400
- MU = 2000
- PC = 0.6

Dla każdego parametru PM z listy:

```
PMS = [0.2, 0.1, 0.05, 0.01, 0.001, 0.0005, 0.0001]
```

Zbadano średni wynik rozwiązania na przestrzeni 25 iteracji.

Wyniki:

```
pm = 0.2: mean = 220.56
pm = 0.1: mean = 222.88
pm = 0.05: mean = 227.92
pm = 0.01: mean = 248.52
pm = 0.001: mean = 290.6
pm = 0.0005: mean = 290.08
pm = 0.0001: mean = 285.28
```

Jak widać najlepsze okazało się prawdopodobieństwo mutacji równe 0.001. Wartość ta daje dobre rozwiązania, ponieważ zapobiega stagnacji, jednocześnie minimalizując ryzyko destabilizacji dobrych rozwiązań.

Wartość pm = 0.001 została użyta do pozostałych eksperymentów w punkcie 3. i 4. aby uzyskać jak najlepsze rozwiązanie końcowe.