Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Przeszukiwanie i Optymalizacja

Dokumentacja końcowa zadania nr 5

Tymon Kobylecki, Mateusz Palczuk

Warszawa, 06.2022

Spis treści

1. Problem

1.1. Treść zadania

W każdej komórce planszy prostokątnej o rozmiarze $4 \times n$ wpisano liczbę całkowitą z_{ij}. Masz do dyspozycji m kart, które musisz rozmieścić na planszy. Poprawny rozkład kart zakłada, że żadna para kart nie może zajmować komórek sąsiadujących w pionie lub poziomie. Twoim zadaniem jest znalezienie takiego rozkładu kart na planszy, aby suma liczb zapisanych w komórkach planszy była jak największa. Nie musisz wykorzystywać wszystkich kart.

1.2. Wizualizacja zadania

Zadanie zostało zrealizowane w języku Python, a jego wizualizacja odbywa się w terminalu. Podczas pracy algorytmu wyświetla się liczba symbolizująca stopień ukończenia zadania, zaś po jego zakończeniu wyświetla się dodatkowo kopia planszy, na której, zamiast wartości pól, przedstawione są binarne oznaczenia czy karta została położona na danym polu, czy nie. Obok kopii planszy wypisywana jest zdobycz punktowa algorytmu przy widocznym układzie kart. Wizualizacja przykładowego rozwiązania widoczna jest na obrazku 1.1.

```
[[-10  -4   3   2   6   9   1   6]
 [ -3   0  -8   5   7   2 -10   7]
 [ -1  -1  -3  -7   7  -3   5   1]
 [  5   5  -8  -8   3  -1   0  -4]]
[[0 0 1 0 0 1 0 1]
 [1 0 0 1 0 0 0 0]
 [0 1 0 0 1 0 0 1]
 [1 0 0 1 0 0 0 0]] 24
```

Rys. 1.1. Przykładowa wizualizacja z widoczną planszą, rozwiązaniem w formie planszy binarnej oraz wynikiem punktowym odpowiadającym wyświetlonemu rozwiązaniu

2. Zastosowany algorytm

Algorytmem zastosowanym do rozwiązania zadania został algorytm genetyczny. Został on wybrany z uwagi na możliwość zapisu zadania w przestrzeni wektorów binarnych.

2.1. Reprezentacja problemu

Zadanie zostało przedstawione w przestrzeni wektorów binarnych o długości $4 \cdot n$, czyli o liczbie pól równej liczbie pól na planszy. Każde pole odpowiada decyzji algorytmu dotyczącej położenia karty na danym polu. 1 oznacza położenie karty, zaś 0 opuszczenie pola. Podczas wizualizacji wektor jest ponownie „składany" do postaci prostokątnej planszy.

2.2. Selekcja

W algorytmie została zastosowana selekcja ruletkowa, ze względu na to, że umożliwia ona eksplorację lepiej, niż chociażby selekcja progowa. W tym zadaniu za maksimum lokalne można uznać sytuację, w której nie możemy poprawić wyniku po dołożeniu dodatkowych kart. Eksploracja jest więc istotna, aby móc rozważyć sytuację, gdzie zdejmujemy kartę z planszy.

2.3. Krzyżowanie

Zastosowane zostało krzyżowanie jednopunktowe, polegające na losowym wyborze punktu na wektorze-rodzicu (z równym prawdopodobieństwem dla wszystkich punktów), a następnie na zamianie końcówki „odciętej" w tym punkcie z końcówką o równej długości, odciętej od drugiego rodzica. W ten sposób z dwóch rodziców powstają dwaj potomkowie.

2.4. Mutacja

Mutacja zastosowana w algorytmie rozwiązującym zadanie polegała na zamianie jednego bitu (wybranego losowo, z równym prawdopodobieństwem dla wszystkich bitów) na przeciwny. Następnym elementem było rzutowanie rozwiązań przekraczających dopuszczalną liczbę m kart na losowo wybrane najbliższe (gdyż zawsze jest ich więcej niż jedno) rozwiązanie o liczbie kart równej m. Odbywa się to poprzez zamianę k losowo wybranych jedynek, gdzie k to różnica sumy jedynek w wektorze i m. W ten sposób ograniczana jest pula nieprawidłowych rozwiązań, dzięki czemu algorytm mniej „błądzi".

3. Eksperymenty

3.1. Przeprowadzone eksperymenty

Zmiennymi manipulowanymi podczas eksperymentów były:

— liczba iteracji
— liczebność populacji
— parametr m - liczba dostępnych kart
— zakres wartości, jakie mogą pojawiać się na kartach
— prawdopodobieństwo mutacji
— prawdopodobieństwo krzyżowania

Podczas eksperymentów środowisko miało stałe ziarna losowości oraz stałe n wynoszące 5. Taki wybór tego parametru poparty był zbalansowaniem między nietrywialnością rozwiązania oraz akceptowalnym czasem na przeprowadzenie eksperymentów. Warto jednak nadmienić, że algorytm jest w stanie przedstawić niezerowe rozwiązania dla planszy o wiele większych. Ziarna wynosiły odpowiednio 32493 dla losowań wewnątrz algorytmu genetycznego oraz 2137 dla losowości wewnątrz symulacji gry. Optimum globalne dla takiego środowiska wynosi 35. Pozostałymi parametrami bazowymi, które były następnie manipulowane pojedynczo, były:

— liczba pokoleń = 100000
— liczebność populacji = 100
— $m = 20$
— $p_c = 0{,}7$
— $p_m = 0{,}01$
— limit górny wartości punktowych pól = 10
— limit dolny wartości punktowych pól = -10

3.2. Analiza eksperymentów

Na wykresie 3.1 widoczny jest wynik działania algorytmu w zależności od liczby pokoleń populacji. Łatwo zauważyć, że skuteczność algorytmu rośnie wraz z liczbą iteracji, ale po 50 tysiącach operacji zdobycz punktowa algorytmu zatrzymuje się na wartości 34.

Podobną zmianę można zaobserwować przy manipulacji liczbą kart (rys. 3.2), przy czym warto zauważyć, że dla $n = 5$ wartości $m \geq 10$ są równoważne i wszelkie różnice widoczne na wykresie po przekroczeniu tej wartości wynikają tylko i wyłącznie z losowości algorytmu. Wspomniana zmiana wartości zdobyczy punktowej to znów stabilizacja osiąganych wyników w okolicach optimum globalnego po przekroczeniu pewnego progu. W przypadku liczby dostępnych kart tym progiem jest liczba 6. Oznacza to najprawdopodobniej, że rozwiązanie optymalne dla tego układu środowiska wymaga właśnie tylu rozłożonych kart.

Kolejnym wykresem o podobnym kształcie jest wykres 3.3, dotyczący relacji zdobyczy punktowej algorytmu i liczebności populacji. Widać na nim, że po osiągnięciu populacji bliskiej 30 algorytm zaczyna oscylować blisko wartości optimum.

Na wykresach 3.4 i 3.5 widać zależność łącznego wyniku uzyskanego przez algorytm od zmian prawdopodobieństw odpowiednio: krzyżowania (p_c) oraz mutacji (p_m). Nie wskazują one jednoznacznych wniosków, co wynika najprawdopodobniej z faktu wybrania pojedynczego środowiska do testów.

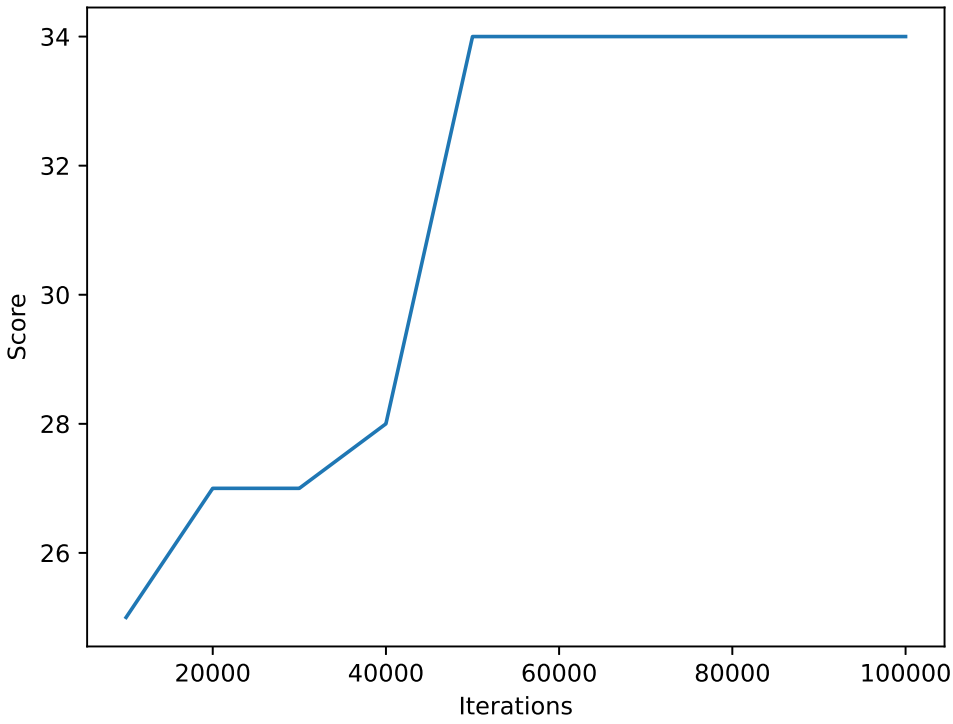
Na podstawie wykresów można jednak zauważyć, ze prawdopodobieństwo mutacji przynosi lepsze wyniki, gdy jest bliższe 0 lub 1 niż 0,5.

Ostatni przeprowadzony eksperyment polegał na zmianie zakresu wartości umieszczanych na planszy. Wyniki tegoż eksperymentu widoczne są na wykresie 3.6.

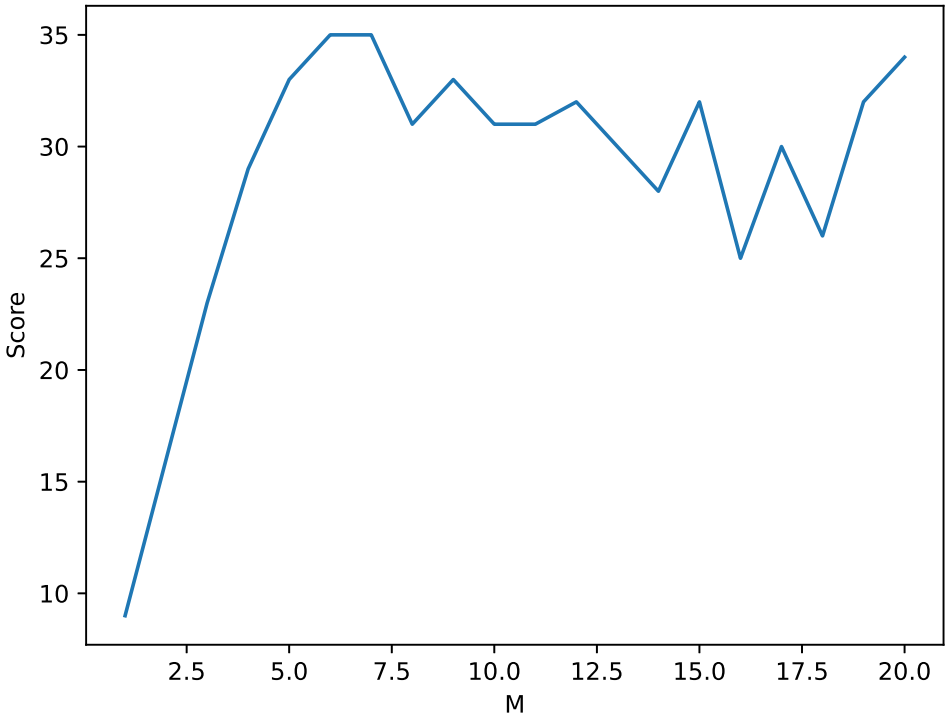
Można na jego podstawie zaobserwować, że, wraz z powiększaniem się zakresu wartości na planszy, algorytm osiąga coraz lepsze rezultaty. Najprawdopodobniej wynika to jednak z faktu, że optimum globalne takiego środowiska jest wyższe, gdyż zwiększenie górnego limitu wartości komórki sprawia, iż pola dodatnie przynoszą średnio większy zysk. Pola o ujemnej wartości, po zwiększeniu limitu dolnego (w przypadku ich zakrycia) przynoszą średnio większą stratę. Wyższy wynik oznacza, że algorytm „chętniej" zakrywa pola o dodatniej wartości niż o ujemnej, zatem wykonuje swoje zadanie poprawnie.

3.3. Instrukcja odtworzenia eksperymentów

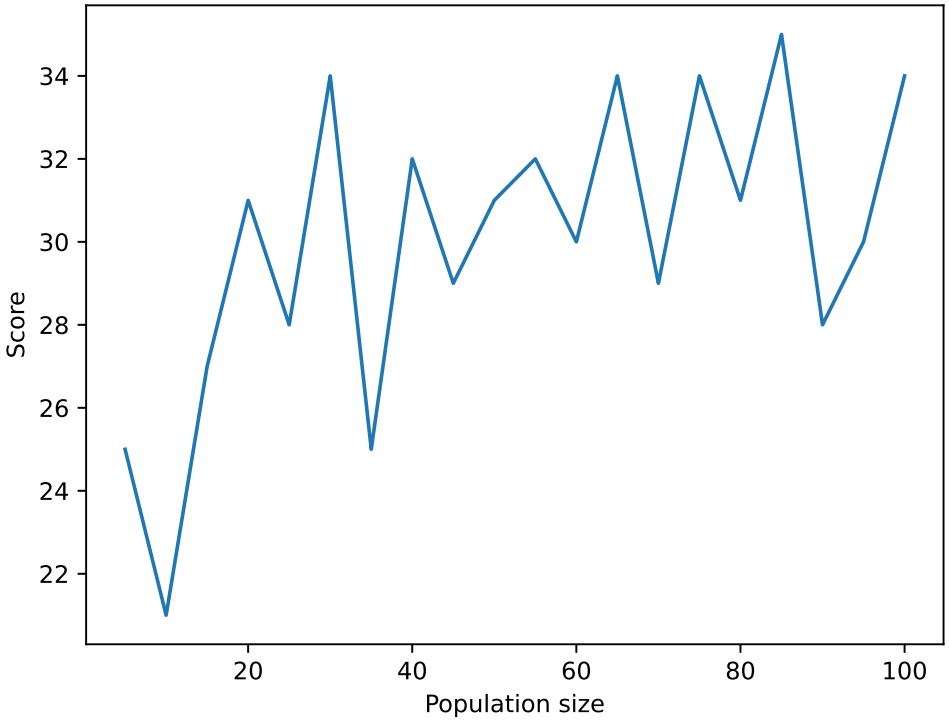
Przeprowadzone eksperymenty znajdują się w folderze `/results`. Każdemu eksperymentowi odpowiada zestaw plików o nazwach w formacie `XXX_board.txt`, `XXX_params.yaml` oraz `XXX_result.txt`, gdzie `XXX` odpowiada nazwie danego eksperymentu i jest takie samo dla wszystkich trzech plików. Plik `XXX_board.txt` służy wyłącznie do manualnego podglądu środowiska i zawiera widok planszy, a plik `XXX_params.yaml` zawiera wszystkie parametry algorytmu razem z



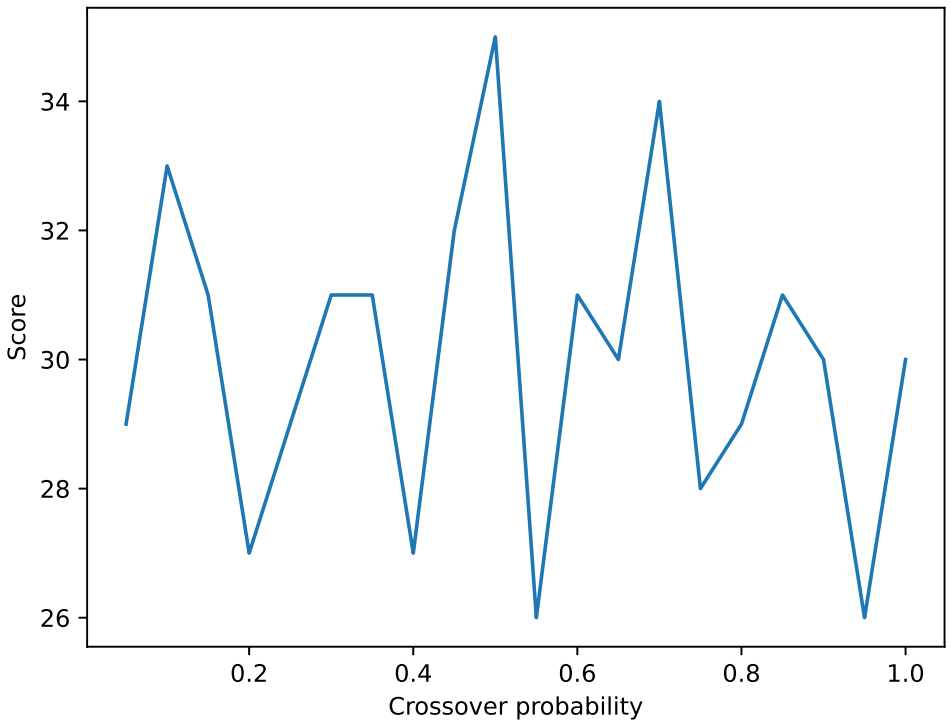
Rys. 3.1. Wynik działania algorytmu w zależności od liczby iteracji



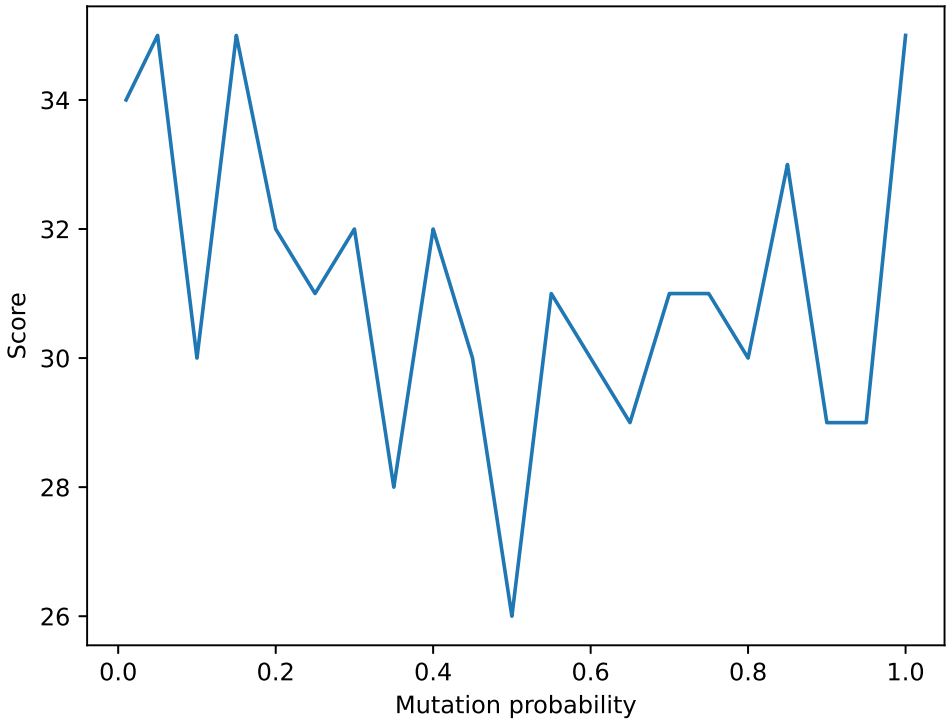
Rys. 3.2. Wynik działania algorytmu w zależności od liczby kart którymi można przykrywać pola na planszy



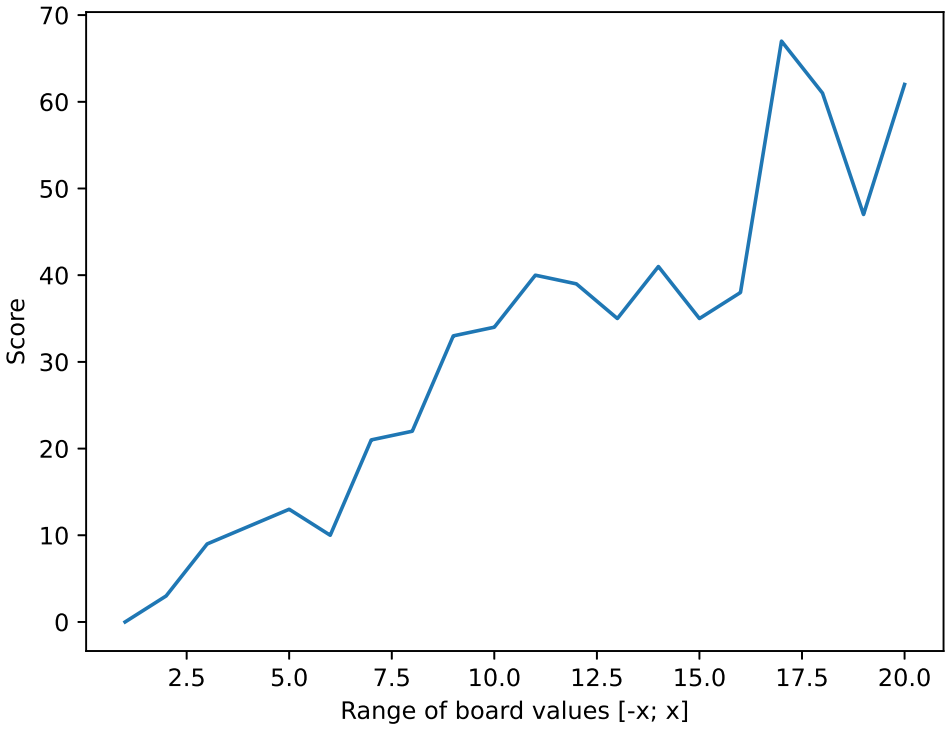
Rys. 3.3. Wynik działania algorytmu w zależności od liczby osobników w populacji



Rys. 3.4. Wynik działania algorytmu w zależności od prawdopodobieństwa krzyżowania



Rys. 3.5. Wynik działania algorytmu w zależności od prawdopodobieństwa mutacji



Rys. 3.6. Wynik działania algorytmu w zależności od zakresu wartości na polu gry

ziarnami liczb pseudo-losowych do prawidłowego odtworzenia każdego eksperymentu. Natomiast plik `XXX_result.txt` zawiera już wynik danego eksperymentu na który składają się:

— plansza uzupełniona wartościami 1 i 0 oznaczającymi odpowiednio, czy dana komórka została przykryta, czy nie
— liczbę punktów zgromadzoną przez algorytm w wyniku przykrywania planszy kartami
— czas wykonywania się programu w przypadku pierwotnego wykonania (podczas odtwarzania czas ten może się różnić w zależności od użytego sprzętu)

Aby odtworzyć eksperyment, należy nazwę (bez rozszerzenia) interesującego nas eksperymentu `XXX` umieścić w jedynej liście w pliku `recreate.yaml` (należy pamiętać o odpowiednim uzupełnianiu przecinków!), a następnie uruchomić skrypt `recreate.py`. W wyniku tych działań każdy z umieszczonych w pliku `recreate.yaml` eksperymentów zostanie wykonany po kolei, a wyniki zostaną wyświetlone w konsoli tak jak na rys. 3.7.

```
---------- 5001 ----------
[[-10  -4   3   2   6]
 [  9   1   6  -3   0]
 [ -8   5   7   2 -10]
 [  7  -1  -1  -3  -7]]
[[0 0 0 0 1]
 [1 0 1 0 0]
 [0 1 0 1 0]
 [1 0 1 0 0]] 34
---------- iters5002 ----------
[[-10  -4   3   2   6]
 [  9   1   6  -3   0]
 [ -8   5   7   2 -10]
 [  7  -1  -1  -3  -7]]
[[0 0 0 0 1]
 [1 0 1 0 0]
 [0 1 0 1 0]
 [0 0 1 0 0]] 27
---------- pop_size5005 ----------
[[-10  -4   3   2   6]
 [  9   1   6  -3   0]
 [ -8   5   7   2 -10]
 [  7  -1  -1  -3  -7]]
[[0 0 0 0 0]
 [1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 0]] 28
---------- range5020 ----------
[[ 12  18   1  -8  -4]
 [ -9  -4  19  14   2]
 [  7 -20   8  -3 -11]
 [ 11   9   0   8  19]]
[[1 0 0 0 1]
 [0 0 1 0 0]
 [1 0 0 0 0]
 [0 1 0 0 1]] 62
```

Rys. 3.7. Wynik działania skryptu odtwarzającego przykładowe eksperymenty

4. Struktura projektu

Kod źródłowy składa się z następujących plików:

— `GA.py` - plik zawierający implementację algorytmu genetycznego z omówionymi wcześniej etapami selekcji, krzyżowania oraz mutacji
— `game.py` - plik zawierający symulację gry wraz z implementacją zasad oraz funkcją celu, automatycznie zapisuje eksperymenty w folderze `/results`
— `main.py` - plik uruchamiający pojedynczą symulację działania algorytmu dla określonych parametrów, pozwala również na opcjonalny zapis eksperymentu w folderze `/results` z nazwą podaną przez użytkownika
— `loop.py` - plik uruchamiający wiele symulacji równolegle
— folder `/results` - folder zawierający pliki z zapisanymi parametrami symulacji, powstałe w wyniku uruchomienia plików `main.py` bądź `loop.py`
— `recreate.py` - skrypt umożliwiający odtworzenie eksperymentów na podstawie zapisów z folderu `/results`
— `recreate.yaml` - plik wejściowy do skryptu `recreate.py`, umożliwiający wczytanie parametrów symulacji (wraz z ziarnami funkcji pseudolosowych) z folderu `/results`