WSI - ćwiczenie 2.
Algorytmy ewolucyjne i genetyczne

17 marca 2025

1 Sprawy organizacyjne

1. Ćwiczenie realizowwane jest samodzielnie.
2. Ćwiczenie wykonywane jest w języku Python.
3. Ćwiczenie powinno zostać oddane najpóźniej na 5. zajęciach. W ramach oddawania ćwiczenia należy zademonstrować prowadzącemu działanie kodu oraz utworzyć pull request (z kodem oraz raportem) który prowadzący będzie mógł komentować.
4. Implementacja powinna być zgodna z interfejsem dostępnym w repozytorium `https://gitlab-stud.elka.pw.edu.pl/jlyskawa/wsi-template`
5. Raport powinien być w postaci pliku .pdf, .html albo być częścią notebooka jupyterowego. Powinien zawierać opis eksperymentów, uzyskane wyniki wraz z komentarzem oraz wnioski.
6. Na ocenę wpływa poprawność oraz jakość kodu i raportu.
7. Można korzystać z pakietów do obliczeń numerycznych, takich jak *numpy*
8. Implementacja powininna być ogólna - w szczególności działać dla dowolnej funkcji $\{0,1\}^n \to \mathbf{R}$

2 Ćwiczenie

Celem ćwiczenia jest implementacja algorytmu genetycznego z mutacją, selekcją ruletkową, krzyżowaniem jednopunktowym oraz sukcesją generacyjną.

Następnie należy zbadać działanie algorytmu na przykładzie problemu opisanego w rozdziale 3.

- Zastanów się, jakie mogło by być trywialne rozwiązanie. Jaki byłby jego wynik? Potraktuj to rozwiązanie jako punkt odniesienia w eksperymentach.

- Jakie jest górne ograniczenie wyniku?
- Zmierz, jaki wynik osiąga rozwiązanie losowe.

Należy znaleźć zestaw hiperparametrów który daje dobry wynik (zauważalnie lepszy niż rozwiązanie losowe), a następnie znaleźć zbadać wpływ wybranego przez siebie hiperparametru.

Należy zwizualizować najlepsze uzyskane rozwiązanie.

Dla chętnych Zmodyfikuj metodę krzyżowania tak, aby była lepiej dostosowana do rozwiązywanego problemu.

Ta część zadania nie wpływa bezpośrednio na wynik.

3 Opis problemu

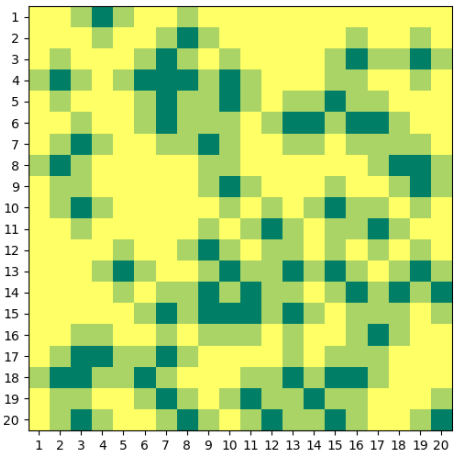
Zadaniem jest optymalizacja obiektów na planszy 20x20 tak, aby zmaksymalizować liczbę punktów.

Każde pole daje 1 punkt jeżeli jest wolne i sąsiaduje z przynajmniej jednym obiektem, w przeciwnym wypadku daje 0 punktów.

Pojedynczym osobnikiem jest wektor 400 wartości (odpowiadających kolejnym elementom planszy 20x20) 0 i 1, gdzie 0 oznacza, że pole jest wolne, a 1, że na polu znajduje się obiekt.

Przykładowe rozwiązanie zostało zwizualizowane na rysunku 3

Sekcja 4 zawiera implementację funkcji celu z użyciem pakietu *numpy*.



Rysunek 1: Wizualizacja przykładowego rozwiązania o wartości 158 punktów. Ciemnozielone pola zawierają obiekty. Jasnozielone pola, sąsiadujące z ciemnozielonymi, dają 1 punkt. Żółte pola nie zawierają ani obiektów ani nie dają punktów.

4 Kod ewaluacji osobnika (z użyciem numpy)

Poniższa implementacja zadziała zarówno dla pojedynczego osobnika jak i dla kolekcji osobników.

```
import numpy as np

def shift_r(x):
    return np.concatenate((np.zeros_like(x[..., :, -1:]), x[..., :, :-1]), axis=-1)

def shift_l(x):
    return np.concatenate((x[..., :, 1:], np.zeros_like(x[..., :, :1])), axis=-1)

def shift_t(x):
    return np.concatenate((np.zeros_like(x[..., -1:, :]), x[..., :-1, :]), axis=-2)

def shift_b(x):
    return np.concatenate((x[..., 1:, :], np.zeros_like(x[..., :1, :])), axis=-2)

def evaluate(x, size=20):
    assert np.shape(x)[-1] == size * size
    grid = np.cast[np.int_](np.reshape(x, (-1, size, size)))
    points = np.minimum(
        shift_r(grid) + shift_l(grid) + shift_t(grid) + shift_b(grid),
        1 - grid
    )

    return points.reshape(np.shape(x)).sum(-1)
```