Warszawa 16.01.2019

Dokumentacja projektu na Podstawy Sztucznej Inteligencji, semestr 18Z

Autorzy:

Adrian Nadulny, 283706;

Dariusz Stalewski, 283771;

Szymon Bieńkowski, 283665;

Treść zadania:

SK.AE.2 Stworzyć program, który za pomocą algorytmu ewolucyjnego zaplanuje rozmieszczenie szpitalnych oddziałów udarowych w Polsce przy założeniu, że oddział może znajdować się w mieście o populacji powyżej 60 tysięcy mieszkańców (z dnia 01.01.2018 r.), każdy pacjent w Polsce powinien być dowieziony do takiego oddziału w czasie nie większym niż 150 minut, a średnia prędkość karetki pogotowia to 75 km/h. Program powinien minimalizować liczbę miast, w których znajdują się oddziały udarowe. Interfejs programu powinien umożliwiać graficzną prezentację wyniku.

Wstępne założenia:

Program podczas swojego uruchomienia wyświetli wiersz poleceń (CLI) i zapyta użytkownika o podanie startowych parametrów dla algorytmu ewolucyjnego. Zadane parametry modyfikują algorytm ewolucyjny poprzez ilość iteracji, ilość osobników algorytmu N->M, itp. Dodatkowo dane wejściowe użytkownika będą odpowiednio zabezpieczone przed podaniem znaków o niewłaściwej strukturze np. ujemna liczba iteracji algorytmu.

Sam algorytm podczas uruchomienia będzie korzystał z pre-wygenerowanej bazy miast o populacji wyższej niż 60 tysięcy mieszkańców(z dnia 01.01.2018), którą pobierzemy ze strony http://stat.gov.pl/ i wygeneruje listę miast która spełnia zadane założenia. Lista miast zostanie zaprezentowana na mapie google, wraz z kołem reprezentującym zasięg oddziału szpitalnego.

Zasięg oddziału szpitalnego na podstawie treści zadania został obliczony i wynosi on 2,5 * 75 / 2 = 93,75 km, gdzie wynik został podzielony przez 2, gdyż uwzględniamy wariant, że karetka wyjeżdża z danego oddziału szpitalnego i do niego wraca w czasie nie dłuższym niż 150 min. Zatem rejon reprezentujące zasięg oddziału szpitalnego będzie reprezentowany kołem o promieniu 93,75 km.

Kroki zastosowanego algorytmu ewolucyjnego:

1. Generowanie populacji początkowej.
2. Ocena wszystkich genotypów.
3. Wybranie z populacji metodą ruletki *elite_size* genotypów i zapisanie ich do tablicy matingpool.
4. Generacja nowych genotypów wykorzystując *matingpool* i zapisanie ich do tablicy *children*.
5. Mutacja wszystkich genotypów z *children*.
6. Zapisanie *matingpool* i *children* jako aktualną populację.
7. Powrót do pkt. 2, jeżeli nie nastąpił warunek stopu.
8. Ocena wszystkich genotypów.
9. Zwrócenie fenotypu dla najlepszego genotypu.

Budowa programu

Program będący rozwiązaniem problemu z zadania, przy pomocy algorytmu ewolucyjnego, został napisany w języku Python. Składa się z pliku głównego oraz z plików pomocniczych.

Główny plik - main.py

Jest to plik w którym znajduje się interfejs użytkownika.

common.py

Plik zawierający wspólne funkcje, z których korzystają pozostałe części programu.

gmaps.py

Plik zawierający funkcje ułatwiające korzystanie z danych geograficznych. Zawiera obsługę zarówno generowania mapy, jak i otrzymywanie lokalizacji wybranych miast i polski.

geneticAlgorithm.py

Plik zawierający algorytm genetyczny wraz z niezbędnymi funkcjami do jego działania.

- genetic_algorithm jest główną funkcją implementującą algorytm genetyczny. Przyjmuje 4 parametry:
  - pop_size, rozmiar populacji
  - elite_size, rozmiar elity przechodzącej do następnego pokolenia
  - mutation_rate, współczynnik określający częstotliwość mutacji
  - generations, liczba generacji rozpatrywana przez algorytm

  Funkcja ta wykorzystuje funkcje pomocnicze.

initialData.py

Zawiera klasę CitiesDataCleane, której zadaniem jest obróbka danych wejściowych z tabeli 22 z pliku: powierzchnia_i_ludnosc_w_przekroju_terytorialnym_w_2018_tablice.xlsx. Dane po pobraniu zostały zmodyfikowane odpowiednio na plik CSV zawierający trzy kolumny: id_number, towns, population. Następnie za pomocą funkcji filter_input_data dane zapisane w pliku spis_miast_2018.csv są odpowiednio filtrowane tak żeby wybrać tylko rekordy tych miast których populacja jest powyżej 60 tysięcy mieszkańców.

city-N-E.csv

Plik zawierający listę wszystkich miast znajdujących się w obrębie granic Polski mających powyżej 60000 mieszkańców. Każdemu miastu odpowiadają dwie liczby reprezentujące położenie geograficzne.

poland_border.json

Plik zawierający listę współrzędnych opisujących granicę polski.

spis_miast_2018.csv

Plik zawierający listę wszystkich miast znajdujących się w obrębie granic Polski. Każdemu miastu odpowiada specjalny numer, nazwa i populacja.

powierzchnia_i_ludnosc_w_przekroju_terytorialnym_w_2018_tablice.xlsx

Plik ten został pobrany ze strony Głównego Urzędu Statystycznego do celów realizacji projektu. Zawiera on spis miast na terenie Polski, wraz z informacją dotyczącą powierzchni i liczby ludności zamieszkującej dane miasta. Link do pobrania materiałów: http://stat.gov.pl/obszary-tematyczne/ludnosc/ludnosc/powierzchnia-i-ludnosc-w-przekroju-terytorialnym-w-2018-roku,7,15.html

Biblioteki wykorzystane:

- **numpy** - wykorzystanie struktury array
- **random** - generowanie liczby losowej
- **operator** - użycie itemgetter
- **pandas** - użycie DataFrame
- **matplotlib.pyplot** - wykorzystany do rysowania wykresu
- **collections** - tworzenie nazwanych krotek w celu zwiększenia czytelności
- **pyclipper** - wykonywanie operacji na kształtach, w celu tworzenia kształtów opisujących Polskę bez pokrycia szpitalami
- **itertools** - użycie funkcji chain w celu efektywnego iterowania po strukturach
- **functools** - użycie cache w celu przyśpieszenia kolejnych iteracji
- **math** - do użycia standardowych funkcji matematycznych
- **typing** - dodanie informacji o typach danych
- **gmplot** - generacja wyniku w postaci mapy google
- **json** - umożliwia pracę z danymi formatu JSON
- **argparse** - parsowanie argumentów od użytkownika
- **csv** - do zapisywania i wczytywania danych z pliku typu CSV.

Wywoływanie programu:

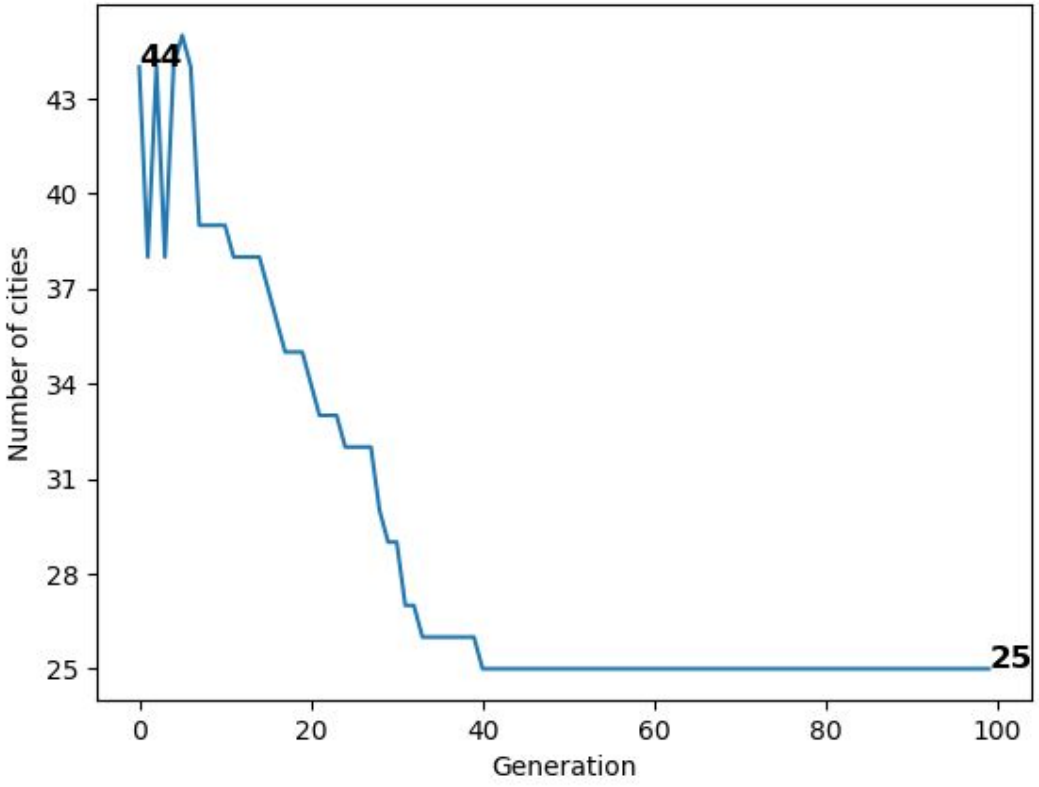
py main.py -i [iterations number] -s [elite_size] [population_size] -r [mutation_rate]

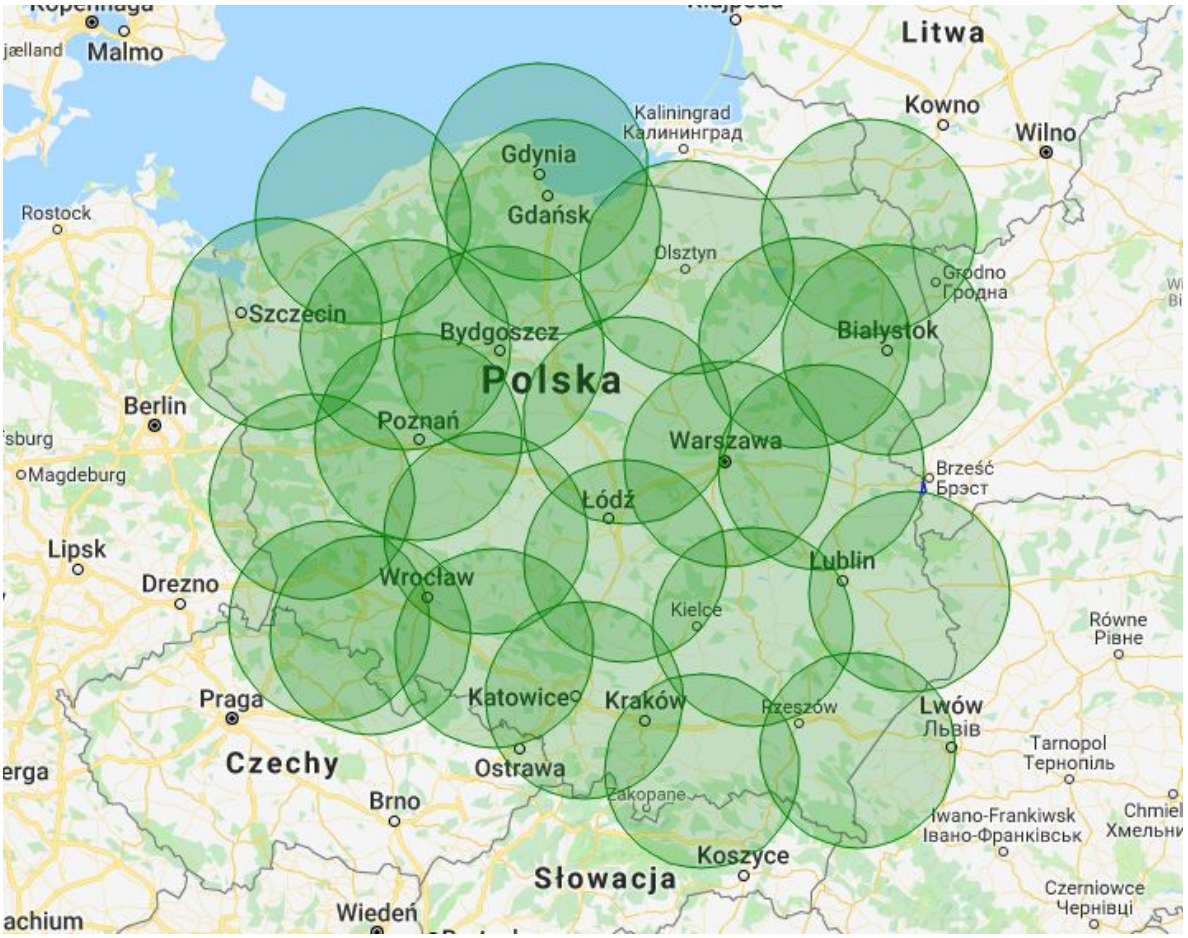
gdzie,
[iterations number] > 0;
[elite_size] > 0;
[children_size] > 0;
[mutation_rate] > 0. < 1

Testowanie działania programu

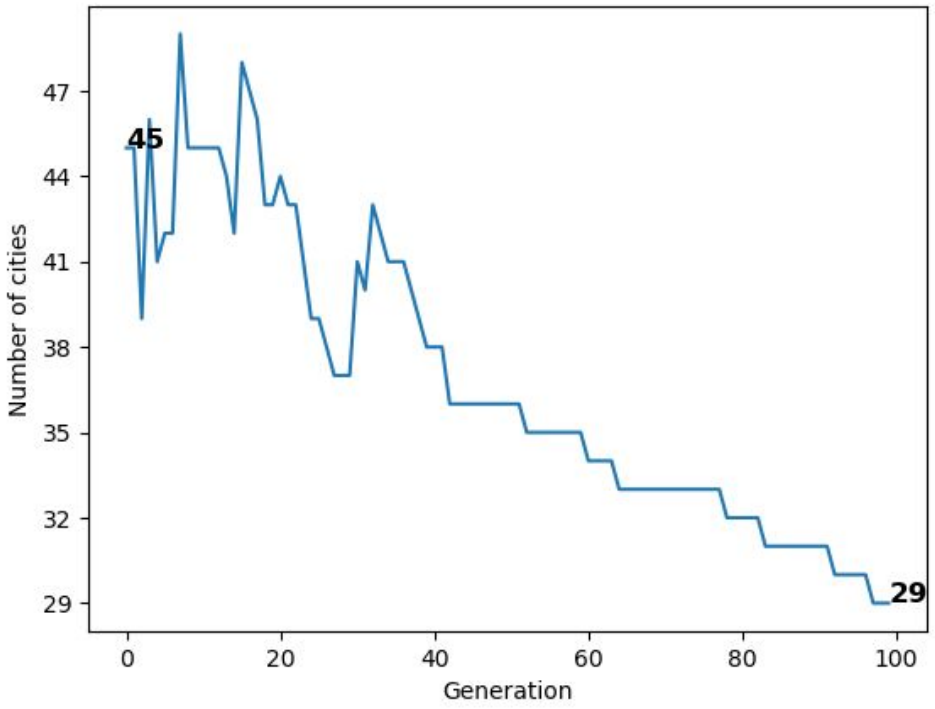
Wyniki z pracy programu w dwóch różnych wariantach zostały przedstawione poniżej.

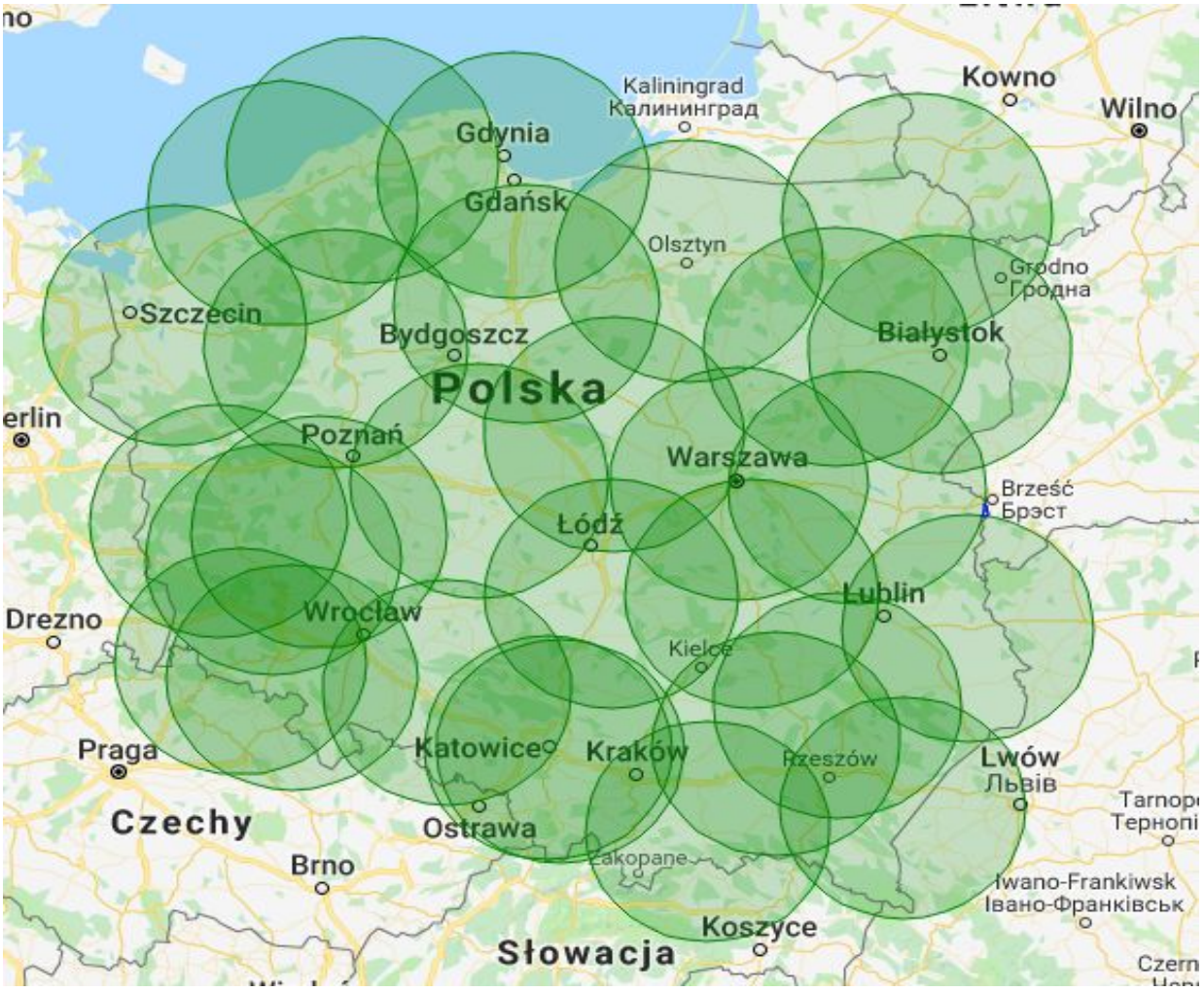
Wariant 1 - (py main.py -i 100 -s 20 100):





Wariant 2 - (py main.py -i 100 -s 2 12):





Wyniki testów wydajnościowych			
Iteracje	**elite_size**	**pop_size**	**Czas [s]**
100	**20**	**80**	**26**
100	**2**	**10**	**4.5**
500	**2**	**10**	**15.9**
500	**20**	**80**	**118.9**

Profilowanie kodu dla 500 iteracji:

```
         62422846 function calls (62393679 primitive calls) in 131.960 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    615/1    0.013    0.000  131.960  131.960 {built-in method builtins.exec}
        1    0.000    0.000  131.960  131.960 main.py:4(<module>)
        1    0.000    0.000  131.369  131.369 main.py:49(main)
        1    0.028    0.028  131.365  131.365 geneticAlgorithm.py:237(genetic_algorithm)
   100100    0.115    0.000  114.882    0.001 geneticAlgorithm.py:107(score)
   100100    0.172    0.000  112.535    0.001 geneticAlgorithm.py:69(score)
   100100    0.385    0.000  112.348    0.001 geneticAlgorithm.py:56(area)
   100101    4.149    0.000  111.590    0.001 geneticAlgorithm.py:46(_shape)
      500    0.008    0.000   73.769    0.148 geneticAlgorithm.py:228(next_generation)
      501    0.047    0.000   57.758    0.115 geneticAlgorithm.py:143(rank_this)
     1125    0.037    0.000   57.245    0.051 {built-in method builtins.sorted}
    50000    0.026    0.000   57.208    0.001 geneticAlgorithm.py:242(<lambda>)
   100101   54.866    0.001   54.866    0.001 {method 'Execute' of 'pyclipper.Pyclipper' objects}
  2954420    1.069    0.000   42.243    0.000 geneticAlgorithm.py:61(__to_pyclipper)
  2954420   41.174    0.000   41.174    0.000 geneticAlgorithm.py:63(<listcomp>)
      500    1.183    0.002   14.909    0.030 geneticAlgorithm.py:150(selection)
```

Możemy zobaczyć, że najwięcej czasu zabiera nam obliczanie kształtów, na podstawie których liczymy pokrycie Polski. Dzięki zastosowaniu cachowania, udało się znacznie przyspieszyć działanie programu.