Politechnika Rzeszowska
im. Ignacego Łukasiewicza

Wydział Matematyki I Fizyki Stosowanej

Projektowanie Systemów i Sieci Komputerowych | *Projekt*

Algorytm genetyczny przy optymalizacji sieci komputerowej

Vitalii Morskyi
Bartosz Pękała

L4 | 2FS-DI

Rzeszów, 2022

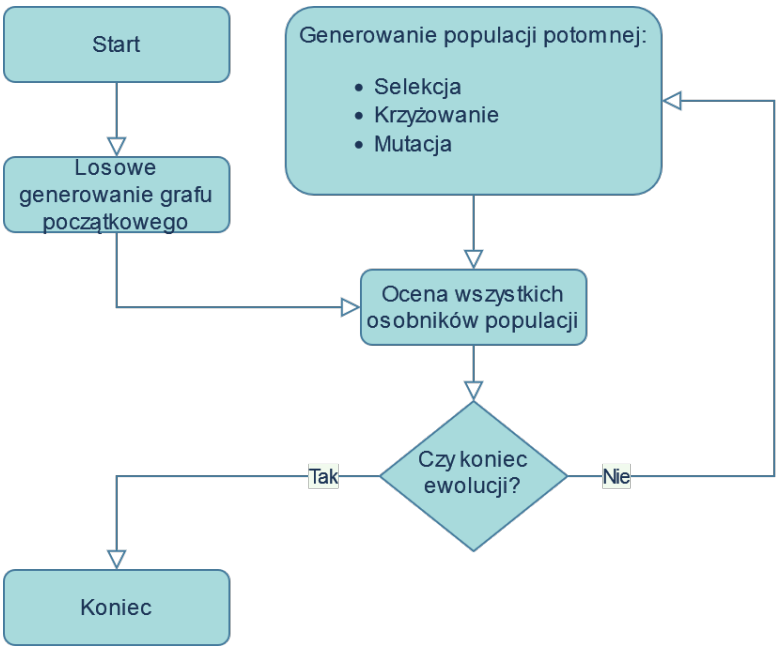
Spis treści

1 Opis rozważanego problemu

Głównym celem tego projektu było stworzenie algorytmu ewolucyjnego, który potrafi optymalizować sieć komputerową do wskazanego stopnia spójności przy czym uwzględniając inne metryki, takie jak łączna długość wszystkich krawędzi. Taki algorytm mógłby zostać użyty przez kompanii, które dostarczają internet i chcą, żeby ich sieć była niezawodną.

2 Opis stworzonego algorytmu

W tym rozwiązaniu został wykorzystany standardowy algorytm genetyczny, który zawiera generację populacji początkowej, jej ocenę i generację populacji potomnej (Rys. 1).



Rysunek 1: Schemat działania algorytmu ewolucyjnego.

Algorytm tworzy populację z n przypadkowych macierzy sąsiedztw, które następnie są ewaluowane. Wybieramy z nich $\sqrt{n}$ najlepszych i wykonujemy najpierw krzyżowanie (ang. cross-over), a następnie mutację.

Ewaluację grafu wykonujemy za pomocą następnego wzoru:

$$S_G = \frac{5}{3} \cdot \frac{R_L \cdot L_m}{L} + C_T + \frac{2}{3} \cdot gauss(C_{ANC}, R_L, R_S) + \frac{1}{3} \cdot gauss(C_{NC}, [R_L], R_S)$$

gdzie R_L - preferowany średni stopień spójności, L_m - łączna długość krawędzi minimalnego drzewa rozpinającego, L - łączna długość krawędzi ewaluowanego grafu, C_T przyjmuje wartość 1, jeżeli graf jest spójny, lub 0, jeżeli graf nie jest spójny, C_{ANC} - średni stopień spójności ewaluowanego grafu, R_S - dopuszczalne odchylenie od preferowanego stopnia spójności, C_{NC} - stopień spójności ewaluowanego grafu. Stałe współczynniki zostały wybrane po dokonaniu wielu testów.

W dość ciekawy sposób zostały zrobione algorytmy realizujące mutacji i skrzyżowania pomiędzy grafami. Żeby zrobić mutację w grafie (dodać lub usunąć niektóre

wierzchołki) tworzymy *"maskę mutacyjną"* - zero-jedynkową macierz analogiczna do macierzy sąsiedztw grafu. Jedynki w takiej masce oznaczają miejsca, gdzie krawędzi zostaną dodane (jeżeli takich krawędzi niema) lub usunięte (jeżeli takie krawędzie istnieją). Żeby wykonać mutację, wystarczy użyć funkcji BITWISE XOR:

$$A_m = A \oplus M_m$$

gdzie A_m - mutowana macierz sąsiedztw, A - wejściowa macierz sąsiedztw, M_m - *"maska mutacyjna"*. Przykład realizacji takiej mutacji został podany poniżej.

```
In [1]: import numpy as np

In [2]: A = np.array([[1, 0, 1], [0, 1, 0], [0, 0, 0]])
   ...: A
Out[2]:
array([[0, 1, 0],
       [0, 1, 0],
       [0, 0, 0]])

In [3]: Mm = np.array([[1, 1, 1], [0, 0, 0], [0, 0, 1]])
   ...: Mm
Out[3]:
array([[1, 1, 1],
       [0, 0, 0],
       [0, 0, 1]])

In [4]: A ^ Mm
Out[4]:
array([[1, 0, 1],
       [0, 1, 0],
       [0, 0, 1]])
```

W analogiczny sposób została zaimplementowana operacja skrzyżowania:

$$C = (A_1 \wedge M_c) \vee (A_2 \wedge \neg M_c)$$

gdzie A, B - macierzy incydencji przodków, M_c *"maska skrzyżowania"*.

```
In [5]: A1 = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0]])
   ...: A1
Out[5]:
array([[0, 0, 1],
       [0, 1, 0],
       [1, 0, 0]])

In [6]: A2 = np.ones((3, 3), int)
   ...: A2
Out[6]:
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

```
In [7]: Mc = np.array([[1, 1, 1], [0, 1, 1], [0, 0, 1]])
   ...: Mc
Out[7]:
array([[1, 1, 1],
       [0, 1, 1],
       [0, 0, 1]])

In [8]: A1 & Mc | A2 & ~Mc
Out[8]:
array([[0, 0, 1],
       [1, 1, 0],
       [1, 1, 0]])
```

Więcej informacji na temat operatorów logicznych w Python można znaleźć w rozdziale 5.

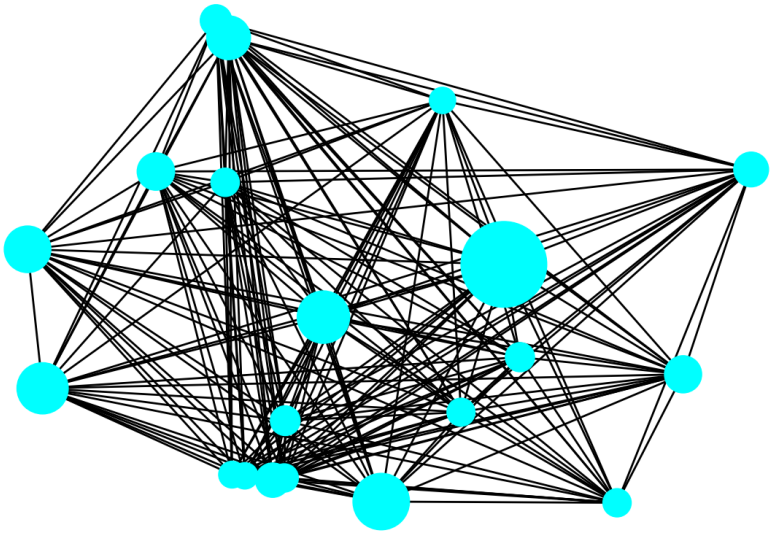
3 Demonstracja implementacji

Dla przeprowadzenia demonstracji możliwości stworzonego modelu została stworzona baza danych większości polskich miast łącznie z ich współrzędnymi. Dane zostały pobrane z dwóch źródeł: Cybermoon i Polska w liczbach. Dane z obu tabel zostały minimalnie przekształcone i połączone w jednej tabeli, pierwsze 20 wierszy z której zostały zaprezentowane na Rys. 2.

	City	District	Province	Population	Latitude	Longitude
0	Warszawa	powiat Warszawa	mazowieckie	1794166	52.259	21.020
1	Kraków	powiat Kraków	małopolskie	779966	50.060	19.959
2	Łódź	powiat Łódź	łódzkie	672185	51.770	19.459
3	Wrocław	powiat Wrocław	dolnośląskie	641928	51.110	17.030
4	Poznań	powiat Poznań	wielkopolskie	532048	52.399	16.900
5	Gdańsk	powiat Gdańsk	pomorskie	470805	54.360	18.639
6	Bydgoszcz	powiat Bydgoszcz	kujawsko-pomorskie	344091	53.120	18.010
7	Lublin	powiat Lublin	lubelskie	338586	51.240	22.570
8	Białystok	powiat Białystok	podlaskie	296958	53.139	23.159
9	Katowice	powiat Katowice	śląskie	290553	50.259	19.020
10	Gdynia	powiat Gdynia	pomorskie	244969	54.520	18.530
11	Częstochowa	powiat Częstochowa	śląskie	217530	50.810	19.129
12	Radom	powiat Radom	mazowieckie	209296	51.399	21.159
13	Toruń	powiat Toruń	kujawsko-pomorskie	198613	53.020	18.609
14	Sosnowiec	powiat Sosnowiec	śląskie	197586	50.279	19.120
15	Kielce	powiat Kielce	świętokrzyskie	193415	50.889	20.649
16	Rzeszów	powiat Rzeszów	podkarpackie	196638	50.049	21.999
17	Gliwice	powiat Gliwice	śląskie	177049	50.310	18.669
18	Zabrze	powiat Zabrze	śląskie	170924	50.299	18.779
19	Olsztyn	powiat Olsztyn	warmińsko-mazurskie	171249	53.779	20.489

Rysunek 2: Początek utworzonego zbioru danych.
Źródła: Cybermoon i Polska w liczbach.

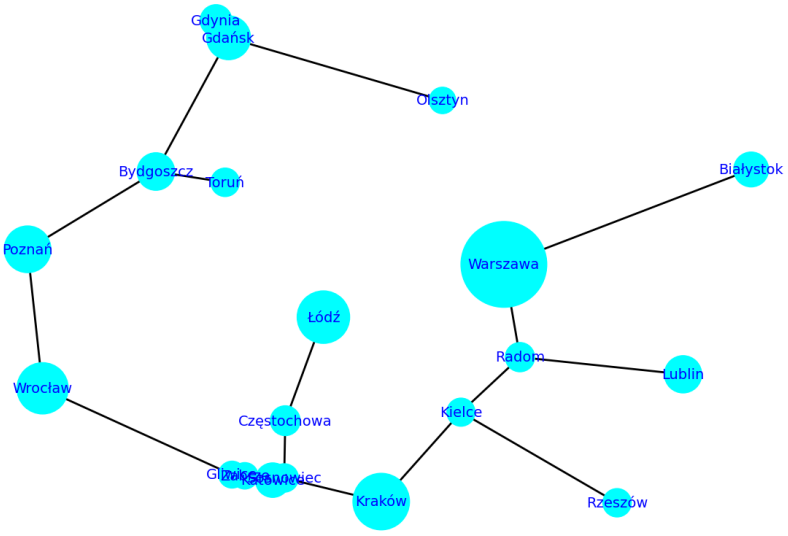
Na podstawie powyższej tabeli został stworzony graf, wierzchołkami którego są polskie miasta (Rys. 3)



Rysunek 3: Graf pełny 20 największych miast w Polsce.

```
average node connectivity = 19.0
total length of the graph = 48899.052 km
```

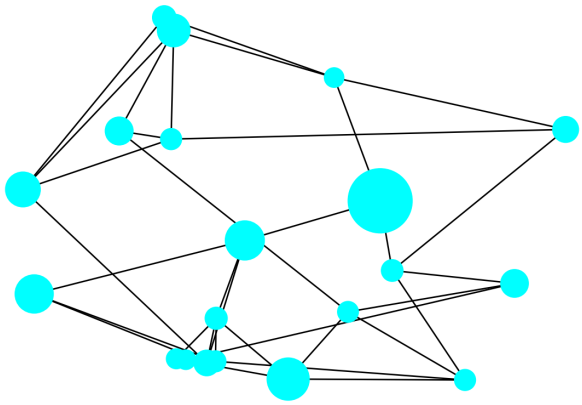
Stwórzmy minimalne drzewo rozpinające dla powyższego grafu używając algorytm Kruskala:



Rysunek 4: Minimalne drzewo rozpinające 20 największych miast w Polsce.

```
average node connectivity = 1.0
total length of the graph = 1683.942 km
```

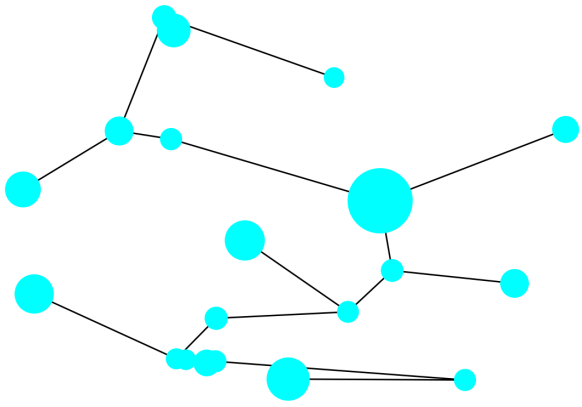
Stwórzmy teraz graf dla którego stopień spójności wynosi 3, a średni stopień spójności wierzchołków wynosi 3.5, używając naszego algorytmu genetycznego:



Rysunek 5: Graf 20 największych miast w Polsce zoptymalizowany za pomocą przygotowanego algorytmu.

```
node connectivity = 3
edge connectivity = 3
average node connectivity = 3.495
total length of the graph = 5549.356 km
```

Utworzony algorytm potrafi również stworzyć graf podobny do minimalnego drzewa rozpinającego:



Rysunek 6: Graf 20 największych miast w Polsce zoptymalizowany za pomocą przygotowanego algorytmu.

```
node connectivity = 1
edge connectivity = 1
average node connectivity = 1.032
total length of the graph = 1969.169 km
```

4 Wnioski

Został przygotowany zbiór klas i funkcji, który pozwala na optymalizację dowolnego grafu nieskierowanego. W trakcie wykonania projektu nauczyliśmy się rozwiązywać różne problemy, związane ze zbieraniem danych wejściowych, tworzeniem grafów oraz dobraniem metryk, które pozwolą stabilizować ewolucję. Wszystkie pliki źródłowe można znaleźć w naszym repozytorium na GitHub.

5 Appendix A

Będziemy przede wszystkim używać danych bibliotek:

```
# Imports and settings
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from optinet.create_graph import CitiesNodes

plt.rcParams['figure.figsize'] = [6, 4]
plt.rcParams['figure.dpi'] =  200
plt.style.use('ggplot')
```

Numpy - Biblioteka, która zapewnia podstawowy pakiet do obliczeń naukowych w Pythonie. Dostarcza różne obiekty typu tablicy wielowymiarowej, macierze z maskami, operacjami na tabelach itd.

Pandas - Biblioteka służąca do przetwarzania danych sekwencyjnych i tabelarycznych. Możemy myśleć o jego strukturach danych jak o tabelach bazy danych lub arkuszach kalkulacyjnych.

Networkx - Biblioteka wprowadzająca grafy, generatory do tworzenia grafów, analiza wyników sieci itd.

Matplotlib - Biblioteka, dzięki której można rysować różne wykresy, wizualizować dane.

Wprowadzamy tabelę 2 i tworzymy graf:

```
# Read the database
df = pd.read_csv("../Data/Cities.csv", sep=";", decimal=",")
tdf = df[:20]
df.head(20)
```

```
graph = CitiesNodes(tdf)
graph.update(nx.complete_graph(n=len(graph.nodes)))
graph.set_edge_lengths()
```

Optymalizujemy graf za pomocą algorytmu genetycznego i rysujemy Rys. 5:

```
graph.update(nx.complete_graph(n=len(graph.nodes)))
graph.optimise(
        n_generations=100,
        population_survival_size=12,
        redundancy_level=3.5,
        redundancy_sigma=0.2,
        reproduction_mutation_prob=0.001)
print(f"Total length of the graph:\t{graph.total_length:.3f} km")
print(f"Edge connectivity:\t\t{nx.edge_connectivity(graph)}")
print(f"Node connectivity:\t\t{nx.node_connectivity(graph)}")
print(f"Average node connectivity:\t{nx.average_node_connectivity(graph):.3f}")
nx.draw(graph,
        pos=list(zip(tdf.Longitude, tdf.Latitude)),
        with_labels=False,
        node_color="cyan",
        node_size=tdf.Population / 1e3,
        labels=nx.get_node_attributes(graph,"City"),\
        font_color="k")
```

Przygotowany algorytm genetyczny można znaleźć w naszym repozytorium na GitHub.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Rysunek 7: Tabela stanów funkcji XOR.