P O L I T E C H N I K A   R Z E S Z O W S K A

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Eryk Jabłoński, Jakub Jędrzejczyk, Radosław Drąg

nr. Index 173148, 173150, 173137

Kierunek
Inżynieria i Analiza Danych

Projektowanie systemów i sieci komputerowych
Praca Projektowa: „Model Axelroda"

Rzeszów, 2024

Spis treści

Wprowadzenie

W naszym codziennym życiu możemy zauważyć, że zwykle otaczamy się osobami podobnymi do nas. Ale jeśli spędzamy nasz czas w grupie osób, od których różnimy się to najprawdopodobniej zaczniemy przejmować cechy naszych rówieśników. Mogą to być poglądy, zachowania, nawyki.

Moglibyśmy więc dojść do następującego pytania: „Jeśli stajemy się coraz bardziej podobni do osób blisko nas to dlaczego wszyscy ludzie nie są tacy sami?”

Model stworzony przez Roberta Axelroda pomaga nam zrozumieć odpowiedz na to pytanie. Dowodzi on, że dziedziczenie cech od naszych rówieśników nie musi prowadzić do całkowitego zjednolicenia społeczeństwa.

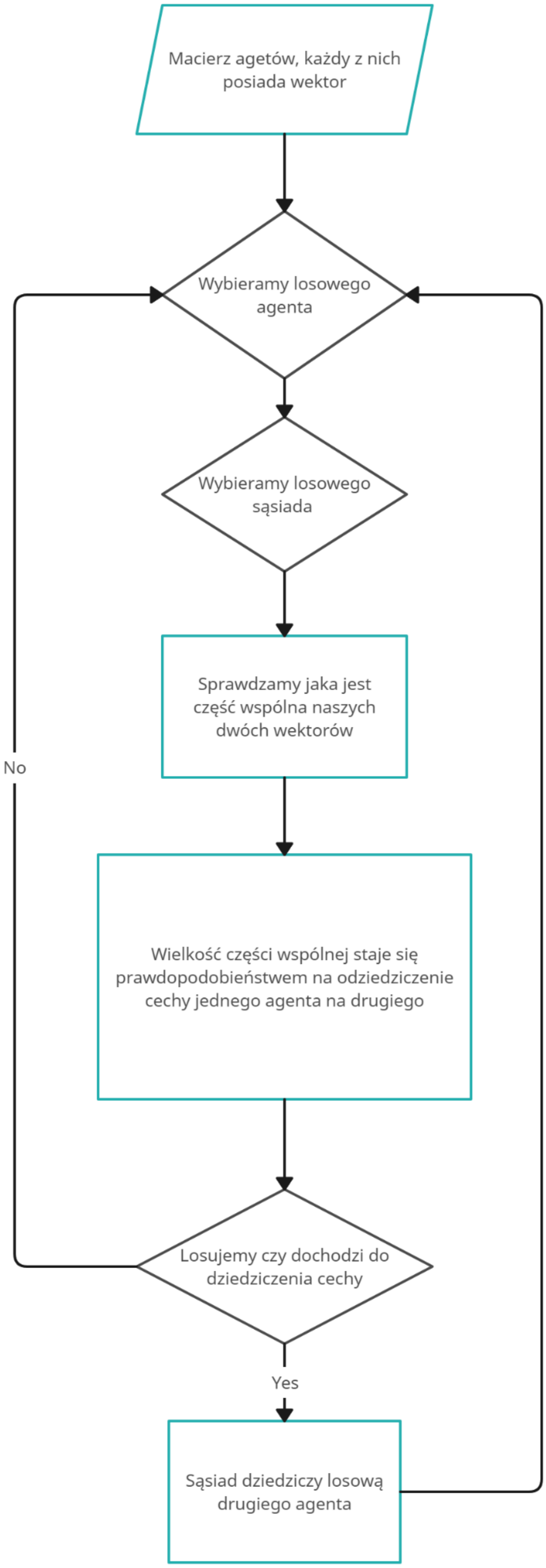
Zasady działania modelu

Model Axelroda to model agentów. Żyją oni na płaszczyźnie 2D i każdy z nich posiada przypisany do niego wektor cech.

Sąsiedzi mogą w nich dziedziczy od siebie cechy z odpowiednim prawdopodobieństwem. Prawdopodobieństwo to jest określane tym jak duża jest cześć wspólna wektorów naszych agentów.

Jeśli jest ona równa zeru to do dziedziczenia nie może dojść. Jeśli jest ona równa 50% to jest 50% szans by jeden agent odziedziczył cechy od drugiego.

Schemat blokowy modelu



Problem Wizualizacji

Do wizualizacji ewolucji naszej społeczności użyjemy heatmapy. Niestety najwygodniejszy sposób wizualizacji, czyli obliczenie średniej arytmetycznej każdego wektora i przypisanie jej koloru jest nie wystarczający.

Jest to spowodowane tym, że dwa kompletnie różne wektory mogą mieć taką samą średnią arytmetyczną wartości ich liczb.

By poradzić sobie z tym problemem zminimalizowaliśmy długość wektorów do 3 i przypisaliśmy indeksom odpowiednio odpowiadanie z wartości r,g i b kolorów agenta. Ostateczny kolor jakim będzie zwizualizowany ten wektor to suma wyżej wymienionych wartości rgb.

Dzięki temu nawet wektory o tej samej średniej arytmetyczne wartości są rozróżnialne.

Sposób zapisu potrzebnych informacji

By ułatwić programowanie będziemy stworzymy macierz której dwie pierwsze kolumny zawierają współrzędne agenta i trzy kolejne zawierają jego wektor. Poniżej przedstawiamy przykładową konwersje.

Zawiera Kordynaty		Zawiera Wektor		
1	1	1	0	1
1	2	1	1	1
2	1	1	0	0
2	2	1	0	1

1,0,1	1,1,1
1,0,0	1,0,1

Kod z wyjaśnieniami

Tworzymy odpowiedni macierze zawierające koordynaty i wektory. Wypełniamy je a następnie spajamy ze sobą tworząc gotową macierz „result”.

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#ustawiamy prarametry
wielkosc_macierzy = 20

#towrzymy macierz zawierającą koordynaty macierzy opsywanej
macierz_2 = np.zeros((wielkosc_macierzy * wielkosc_macierzy, 2))
#tworzymy macierz do przechowywania wektora
macierz_3 = np.zeros((wielkosc_macierzy * wielkosc_macierzy, 3))

#wypelniamy macierz koordynatow
for i in range(1, wielkosc_macierzy + 1):
    for j in range(1, wielkosc_macierzy + 1):
        macierz_2[(i - 1) * wielkosc_macierzy + (j - 1)] = (i, j)

#wypelniamy macierz wektoraa
for i in range(wielkosc_macierzy * wielkosc_macierzy):
    for j in range(3):
        macierz_3[i, j] = random.uniform(0, 1)

#łaczymy je
result = np.concatenate((macierz_2, macierz_3), axis=1)
```

Definiujemy funkcje, w której będzie znajdował się nasz algorytm. Pierwszym krokiem jest wybranie losowego agenta. Następnie wybieramy współrzędne sąsiada.

```
def axelrod(macierz, ilosc_petli, save_path=None):
    #funkcja updatujaca wizualizacje
    def update(frame):
        #losowo wybieramy agenta
        wspolrzedna_x = random.randint(1, wielkosc_macierzy)
        wspolrzedna_y = random.randint(1, wielkosc_macierzy)

        wspolrzedna_sąsiada_x = wspolrzedna_x
        wspolrzedna_sąsiada_y = wspolrzedna_y

        #losowo wybieramy liczby potrzebne do wybrania sasiada
        pomocnicza_wybieranie = random.randrange(-1, 2, 2)
        pomocnicza_dodawanie = random.randrange(-1, 2, 2)

        #wybieramy sasiada tak by jego wspolrzedne nie wyszly poza wielkosc macierzy
        if pomocnicza_wybieranie == -1:
            wspolrzedna_sąsiada_x_nowa = max(1, wspolrzedna_sąsiada_x + pomocnicza_dodawanie)
            wspolrzedna_sąsiada_x = min(wspolrzedna_sąsiada_x_nowa, wielkosc_macierzy)
        else:
            wspolrzedna_sąsiada_y_nowa = max(1, wspolrzedna_sąsiada_y + pomocnicza_dodawanie)
            wspolrzedna_sąsiada_y = min(wspolrzedna_sąsiada_y_nowa, wielkosc_macierzy)
```

Losowo wybieramy index wektora. Wartość, która znajduje się w tym indexie może zostać odziedziczona.

```
#wybieramy index wektora ktory chcemy podmienic i wartosc miejsca o tym indeksie
wybrany_index_wektora = random.randint(1, 3 - 1)
wybrana_wartosc_punktu = macierz[
    (wspolrzedna_x - 1) * wielkosc_macierzy + wspolrzedna_y - 1, wybrany_index_wektora + 1]
```

Sprawdzamy jaka cześć naszych dwóch wybranych wektorów jest wspólna. Będzie to nasze prawdopodobieństwo na odziedziczenie cechy z pierwszego wektora do drugiego.

```
#resetujemy prawdopodobienstwo z poprzedniej iteracji petli
prawdopodobienstwo = 0

#liczymy nowe prawdopodobienstwo
for i in range(3):
    if macierz[(wspolrzedna_x - 1) * wielkosc_macierzy + wspolrzedna_y - 1, i + 2] == \
            macierz[(wspolrzedna_sąsiada_x - 1) * wielkosc_macierzy + wspolrzedna_sąsiada_y - 1, i + 2]:
        prawdopodobienstwo += 1

#losujemy liczbe od 1 do 3
losowa_liczba = random.randint(1, 3)

#jesli jest ona mniejsza od prawdopodonienstwo to dochodzi do dziedziczenia cechy
if losowa_liczba >= prawdopodobienstwo:
    macierz[(wspolrzedna_sąsiada_x - 1) * wielkosc_macierzy + wspolrzedna_sąsiada_y - 1,
    wybrany_index_wektora + 1] = wybrana_wartosc_punktu
```

By zwizualizować naszą macierz posłużymy się heatmapą, kolor jakim jest opisany agent jest determinowany wartościami jakie znajdują się w jego wektorze. Nasz indexy odpowiadaj kolejno wartością kolorów red, green i blue. Kolor jaki widzimy ich sumą. Jest to koniec naszego algorytmu.

```
#kazdej z 3 liczb w wektorze piszypisujemy kolor r, g lub b. dziala on tylko na pijedynczym piksleu
r, g, b = macierz[(wspolrzedna_sąsiada_x - 1) * wielkosc_macierzy + wspolrzedna_sąsiada_y - 1, 2:5]
r, g, b = r * 255, g * 255, b * 255  # Skalowanie do zakresu 0-255
macierz_wizualizacyjna[wspolrzedna_sąsiada_x - 1, wspolrzedna_sąsiada_y - 1] = (r, g, b)

cax.set_array(macierz_wizualizacyjna)
return cax,
```

Tworzymy macierz wizualizacyjną potrzebną nam w naszym modelu i przypisujemy odpowiednie kolory jej elementom. Proszę zauważyć, że tą część kody wywołuje się tylko raz na samym początku działania naszego kodu, jeszcze przed aktywacją funkcji „axelrod”.

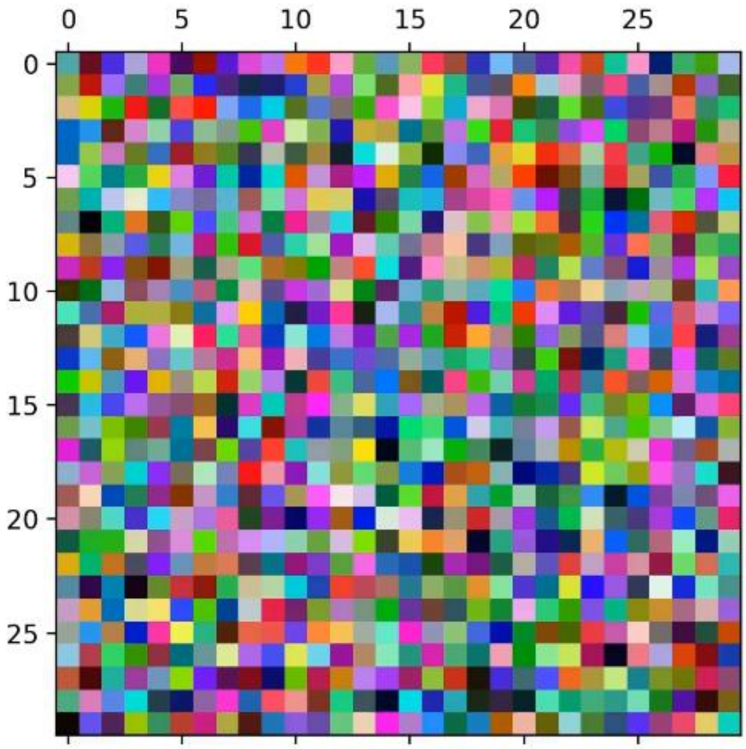
```
macierz_wizualizacyjna = np.zeros((wielkosc_macierzy, wielkosc_macierzy, 3), dtype=int)

#przypisywanie kolorów ale na całe macierzy
#wykonuje sie tylko raz na początku by stworzyc kolor dla kazdego pixela w pierwszej klatce animacji
for i in range(wielkosc_macierzy * wielkosc_macierzy):
    row_idx = int(macierz[i, 0]) - 1
    col_idx = int(macierz[i, 1]) - 1
    r, g, b = macierz[i, 2:5]
    r, g, b = r * 255, g * 255, b * 255  # Skalowanie do zakresu 0-255
    macierz_wizualizacyjna[row_idx, col_idx, :] = (r, g, b)
```

Podajemy inne potrzebne parametry do funkcji wizualizacyjnej i inicjujemy naszą funkcje w której znajduje się nasz algorytm.

```
    fig, ax = plt.subplots()
    cax = ax.matshow(macierz_wizualizacyjna, cmap='viridis')  # Użyj mapy kolorów viridis

    #Aktualizuje tablicę wizualizacyjną
    ani = FuncAnimation(fig, update, frames=range(0, ilosc_petli, 10), repeat=False, blit=True)

    if save_path:
        ani.save(save_path, writer='ffmpeg', fps=10, dpi=200)

    plt.show()

# Wywołanie funkcji
ilosc_petli = 1000
video_save_path = 'F:\heatmap-1000-ostateczny.mp4'  # Wprowadź pełną ścieżkę do dowolnego folderu
axelrod(result, ilosc_petli, save_path=video_save_path)
```

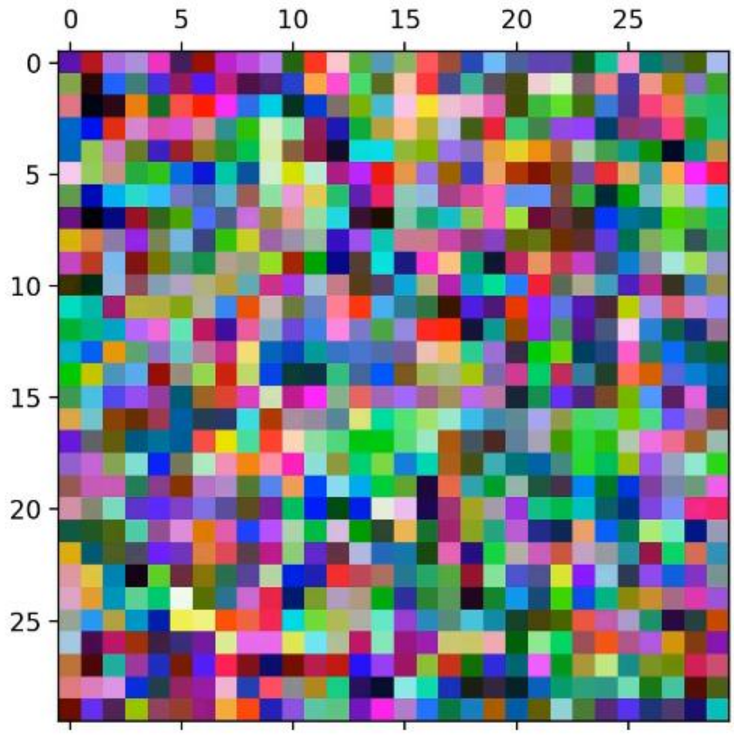
Analiza wyników

Stan początkowy



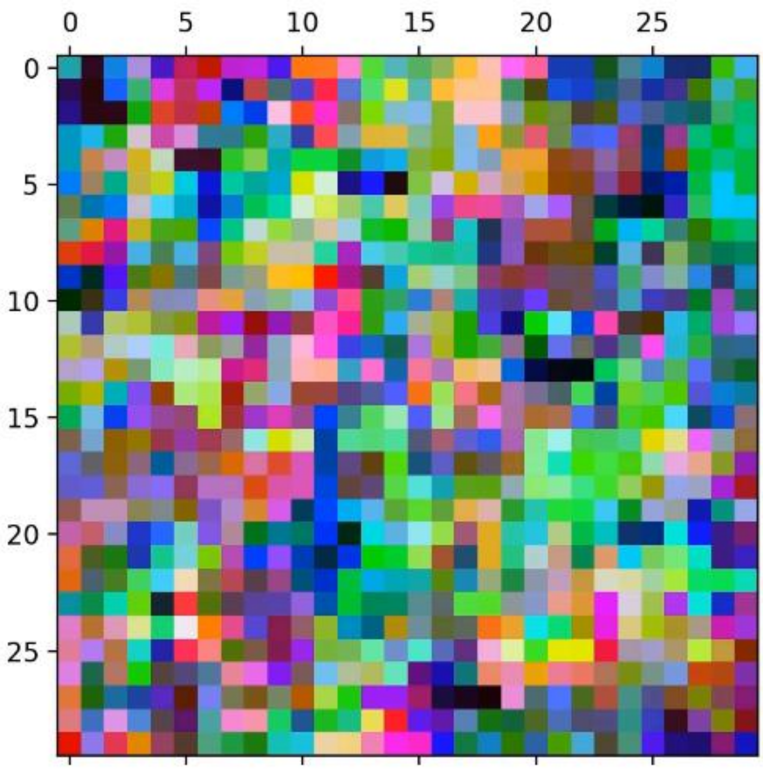
Jak widzimy na początku kolory naszych agentów są całkowicie losowe. Pokrywa się to całkowicie losowym wypełnieniem macierzy na początku algorytmu.

Stan środkowy



Z biegiem czasu wyłaniają się malutkie podgrupy o podobnych cechach.

Stan końcowy



Jak widać podgrupy stały się większe, lecz nie doszło do całkowitego ujednolicenia się macierzy.