Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wydział
Informatyki

LAB 8 - MODELE DYFUZYJNE

Karol Działowski

nr albumu: 39259
przedmiot: Modelowanie zachowań w sieciach złożonych

Szczecin, 27 grudnia 2020

Spis treści

1 Cel laboratorium

Celem laboratorium było modelowanie zachowań w sieciach złożonych. Zadanie polegało na implementacji modelu SIR (*Susceptible, Infectious, or Recovered*).

Jako źródło modelu SIR uważa się pracę Kermacka i McKendricka [1]. Model ten stara się przewidzieć jak rozprzestrzenia się choroba, jaka będzie ogólna liczba zarażonych oraz ile

trwać będzie epidemia. Takie modele mogą pokazać jak różne środki zapobiegawcze mogą wpływa na wynik epidemii, na przykład wybór najskuteczniejszej dystrybucji szczepionek [2].

Model definiują trzy parametry:

początkowa liczba węzłów zarażonych n – określa liczbę węzłów zarażonych w pierwszej iteracji

prawdopodobieństwo wyzdrowienia m – określa z jakim prawdopodobieństwem węzeł przechodzi ze stanu zarażony **I – infected** do stanu ozdrowienia **R – recovered**.

prawdopodobieństwo zarażanie b – określa z jakim prawdopodobieństwem węzeł zaraża sąsiada.

2 Zadanie 1

W pierwszym zadaniu należało przygotować sieć WS [3] z około 20 węzłami. Następnie należało zaimplementować w dowolnym języku programowania model SIR z wykorzystaniem biblioteki *iGraph* [4].

W pierwszym kroku symulacji t_0, aktywujemy losowo n węzłów (tak zwany *seed set* oznaczany jako S_0). Aktywne węzły oznaczono kolorem czerwonym.

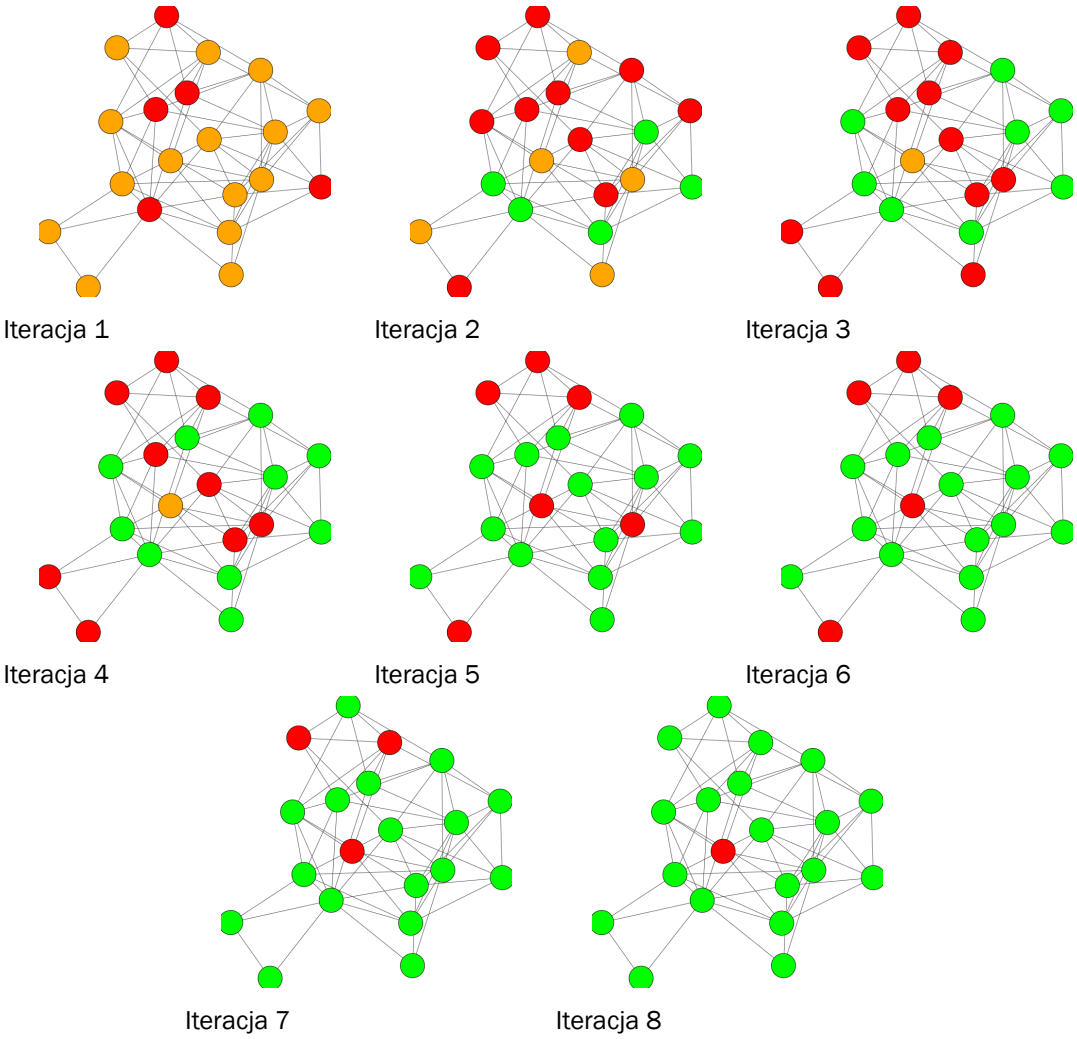
W kolejnych krakach każdy węzeł ze stanem **I** kontaktuje się z każdym sąsiadem i z prawdopodobieństwem b go infekuje. Następnie węzły zainfekowane przechodzą do stanu **R** z prawdopodobieństwem m

Proces trwa tak długo aż nie występują węzły zarażone **I**.

Kod źródłowy przedstawiono na listingu (1).

2.1 Eksperyment

Przetestowano działanie modelu aktywacji progowej na grafie WS [3] o rozmiarze 20 węzłów i parametrem zagęszczenia połączeń $nei = 4$. Uzyskane kroki symulacji przedstawiono na rysunku (1). Kod źródłowy przedstawiono na listingu (2).



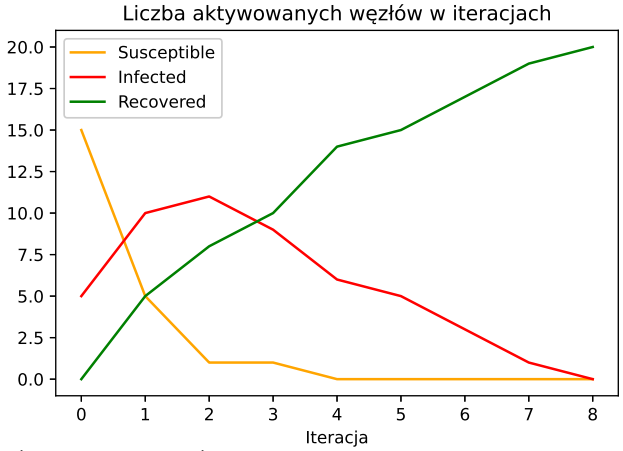
Rysunek 1: Kroki symulacji modelu SIR $n = 5$, $m = 0.3$, $b = 0.5$
Źródło: Opracowanie własne

Rysunek (2) przedstawia liczbę węzłów w poszczególnych stanach w kolejnych iteracjach.

3 Zadanie 2

Drugie zadanie polegało na przetestowanie modelu na większej sieci (2000 węzłów) i przeprowadzenie symulacji dla różnych parametrów m i b. Przygotowano cztery scenariusze:

1. $b = 0.4\%$, $m = 0.4\%$
2. $b = 0.4\%$, $m = 0.5\%$
3. $b = 0.5\%$, $m = 0.4\%$
4. $b = 0.5\%$, $m = 0.5\%$



Rysunek 2: Liczba węzłów w poszczególnych stanach w iteracjach
Źródło: Opracowanie własne

Wszystkie scenariusze testowano miały ustawiony parametr $n = 100$ (liczba początkowych zarażonych).

Zebrane wyniki liczby węzłów w iteracjach pokazano na rysunku (3). Wraz ze wzrostem parametru b (prawdopodobieństwo zarażania) liczba wszystkich zarażonych rośnie oraz szczytowy punkt epidemii jest wyższy.

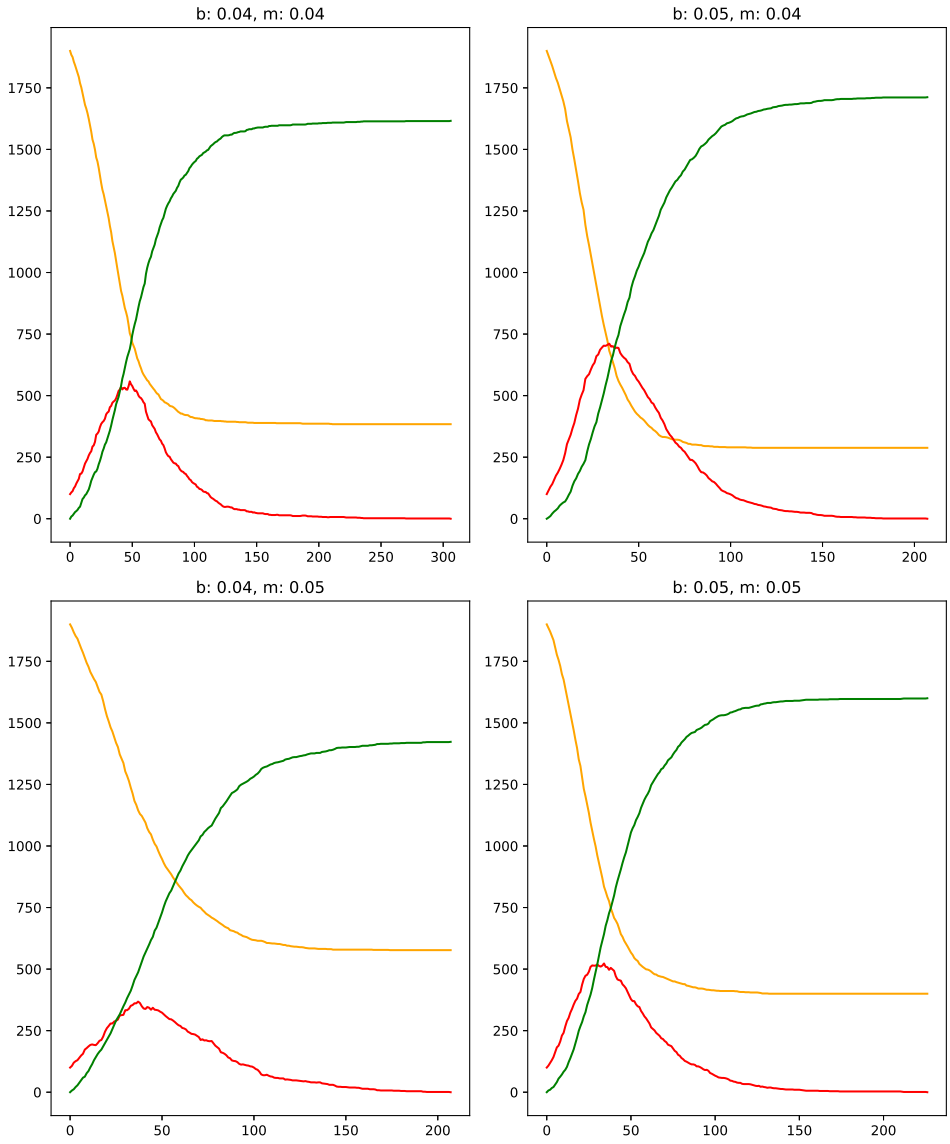
W przypadku większego prawdopodobieństwa zarażania może wystąpić przeciążenie systemu zdrowotnego. Ograniczenie współczynnika b o 1 punkt procentowy wyhamowuje rozwój epidemii. Przykładowo zmniejszając b z 0.05 na 0.04 dla takiego samego parametru $m = 0.04$ maksymalna liczba zarażonych wynosi około 750 porównując do 500 zarażonych dla mniejszego b. Zyskiem jest też mniejsza liczba zarażonych ogółem dla mniejszego prawdopodobieństwa przenoszenia choroby.

Parametr prawdopodobieństwa ozdrowienia m wpływa na szybkość wyjścia z choroby. Można to porównać do efektywności systemu zdrowotnego.

Kod źródłowy przedstawiono na listingu (3).

4 Podsumowanie

Na laboratorium zaimplementowano model SIR w Pythonie, który można dostrajać parametrami m, n oraz b. Przeprowadzono testy na małej sieci syntetycznej WS oraz dla większej sieci liczącej 2000 węzłów dla różnych parametrów modelu SIR.



Rysunek 3: Porównanie symulacji dla różnych parametrów m i b. Wykres przedstawia zależność liczby węzłów od iteracji, czyli prezentuje dynamikę zachorowań i ozdrowień w sieci.

Źródło: Opracowanie własne

Bibliografia

[1] Kermack W. O., McKendrick A. G.: Contributions to the mathematical theory of epidemics—i. *Bulletin of mathematical biology*, vol. 53, no. 1-2, pp. 33–55, 1991.

[2] Wikipedia: *Compartmental models in epidemiology — Wikipedia, the free encyclopedia*, [Online; accessed 28-November-2020], 2020.

[3] Watts D. J., Strogatz S. H.: Collective dynamics of 'small-world' networks. *Nature*, vol. 393, no. 6684, pp. 440–442, czer. 1998, DOI: 10.1038/30918.

[4] Csardi G., Nepusz T.: The igraph software package for complex network research. *InterJournal*, vol. Complex Systems, s. 1695, 2006, URL: https://igraph.org.

A Implementacja modelu aktywacji progowej

Kod źródłowy 1: Kod modelu aktywacji progowej
Źródło: Opracowanie własne

```
from igraph import *
import random
import numpy as np
import matplotlib.pyplot as plt


def simulation(g, n, m, b):
    """
    SIR Model simulation
    :param g: graph
    :param n: size of initial infection
    :param m: probability of recovery
    :param b: probability of infection
    :return: list of graphs in each step, array of S, I, R counts
    """
    g.vs["state"] = "S"
    g.vs["color"] = "orange"
    g.vs["size"] = 50

    seeding(g, n=n)
    update_colors(g)

    g_history = [deepcopy(g)]
    counts = [count_states(g)]
    flag = True
    while flag:
        infection(g, b=b)
        recovery(g, m=m)
        update_colors(g)
        s_count, i_count, r_count = count_states(g)
        counts.append((s_count, i_count, r_count))
        flag = i_count != 0
```

```
        g_history.append(deepcopy(g))

    return g_history, np.array(counts)


def seeding(g, n):
    indexes = g.vs.indices
    indexes_to_infect = random.sample(indexes, n)
    g.vs[indexes_to_infect]["state"] = "I"
    print(g.vs[indexes_to_infect])


def update_colors(g):
    infected_indexes = list(np.where(np.array(g.vs["state"]) == "I")[0])
    g.vs[infected_indexes]["color"] = "red"
    recovered_indexes = list(np.where(np.array(g.vs["state"]) == "R")[0])
    g.vs[recovered_indexes]["color"] = "green"


def infection(g, b):
    infected_indexes = list(np.where(np.array(g.vs["state"]) == "I")[0])
    for i in infected_indexes:
        neighbors = g.neighbors(g.vs[i])
        for neighbor in neighbors:
            if g.vs[neighbor]["state"] == "S":
                if np.random.rand() < b:
                    g.vs[neighbor]["state"] = "I"


def recovery(g, m):
    infected_indexes = list(np.where(np.array(g.vs["state"]) == "I")[0])
    for i in infected_indexes:
        if np.random.rand() < m:
            g.vs[i]["state"] = "R"


def count_states(g):
    infected_indexes = list(np.where(np.array(g.vs["state"]) == "I")[0])
    recovered_indexes = list(np.where(np.array(g.vs["state"]) == "R")[0])
    susceptible_indexes = list(np.where(np.array(g.vs["state"]) == "S")[0])
    return len(susceptible_indexes), len(infected_indexes), len(recovered_indexes)
```

B Implementacja doświadczenia

B.1 Zadanie 1

Kod źródłowy 2: Eksperyment na małej sieci
Źródło: Opracowanie własne

```
g = Graph.Watts_Strogatz(dim=1, size=20, nei=3, p=0.8)
```

```
g_history, counts = simulation(g, n=5, m=0.3, b=0.5)
plt.plot(counts[:, 0], label="Susceptible", c='orange')
plt.plot(counts[:, 1], label="Infected", c='red')
plt.plot(counts[:, 2], label="Recovered", c='green')

plt.legend()
plt.title("Liczba aktywowanych węzłów w iteracjach")
plt.xlabel("Iteracja")
plt.savefig("output/zad1_plot.eps")


for idx, g in enumerate(g_history):
    plot(g).save(f"./output/zad1_{idx}.png")
```

B.2 Zadanie 2

Kod źródłowy 3: Eksperyment dla dużej sieci z różnymi parametrami m i n
Źródło: Opracowanie własne

```
num_nodes = 2000
b_values = [0.04, 0.05]  # prawdopodobienstwo infeckji
m_values = [0.04, 0.05]  # prawdopodobienstwo wyzdrowienia
n = 100  # liczba poczatkowo chorych

fig, axes = plt.subplots(len(m_values), len(b_values), figsize=(10, 12))

idx = 0
for m in m_values:
    for b in b_values:
        g = Graph.Watts_Strogatz(dim=1, size=num_nodes, nei=2, p=0.8)
        g_history, counts = simulation(g, n=100, m=m, b=b)

        ax = axes.flat[idx]
        ax.plot(counts[:, 0], label="Susceptible", c='orange')
        ax.plot(counts[:, 1], label="Infected", c='red')
        ax.plot(counts[:, 2], label="Recovered", c='green')
        ax.set_title(f"b: {b}, m: {m}")
        idx += 1

fig.tight_layout()
fig.savefig("output/zad2_plot.eps")
```