Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

LAB 10 - PROCESY WSPÓŁBIEŻNE

Karol Działowski

nr albumu: 39259

przedmiot: Modelowanie zachowań w sieciach złożonych

Szczecin, 27 grudnia 2020

Spis treści

1 Cel laboratorium 1

2 Wyniki 2

3 Podsumowanie 2

Bibliografia 3

A Implementacja 3

1 Cel laboratorium

Celem laboratorium było rozwinięcie modelu SIR (*Susceptible, Infectious, or Recovered*)) i wprowadzenie drugiego konkurującego procesu (np. wirusa) zgodnie z modelem SI1SI2S.

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 SI1SI2S wprowadza drugi proces konkurujący, który równolegle rozpostrzenia się w sieci. W implementacji przyjęto, że każdy węzeł może zarazić się tylko jeden raz, albo wirusem 1 albo wirusem 2.

Model definiują parametry:

początkowa liczba węzłów zarażonych n_1 – określa liczbę węzłów zarażonych w pierwszej

iteracji wirusem 1

początkowa liczba węzłów zarażonych n_2 – określa liczbę węzłów zarażonych w pierwszej iteracji wirusem 2

prawdopodobieństwo zarażanie b_1 – określa z jakim prawdopodobieństwem węzeł z wirusem 1 zaraża sąsiada.

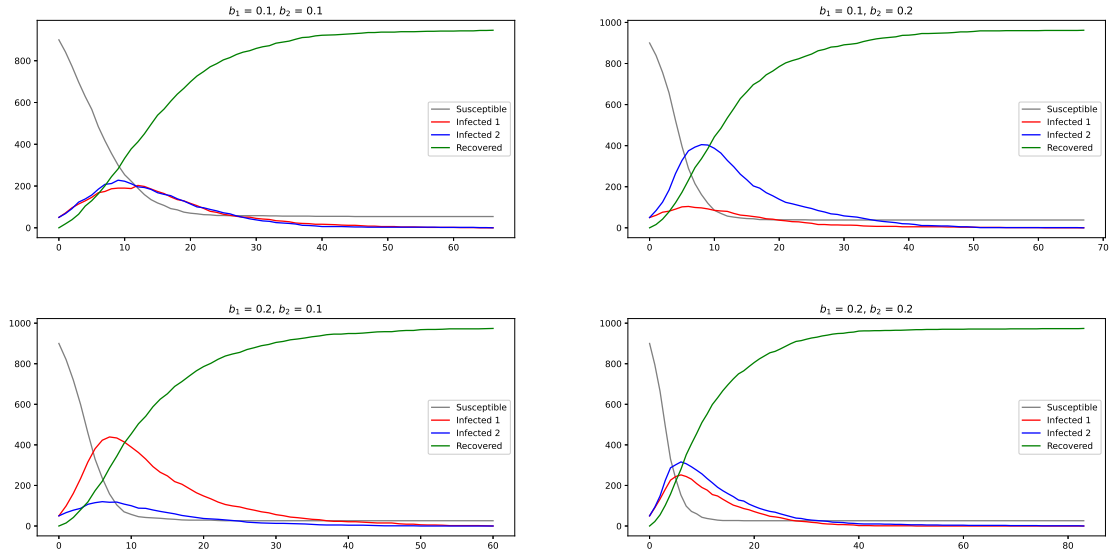
prawdopodobieństwo zarażanie b_2 – określa z jakim prawdopodobieństwem węzeł z wirusem 2 zaraża sąsiada.

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

2 Wyniki

W pierwszym zadaniu należało przygotować sieć WS [3] z około 1000 węzłami. Następnie należało zaimplementować w dowolnym języku programowania model SI1SI2S z wykorzystaniem biblioteki *iGraph* [4].

Przeprowadzono eksperyment dla czterech par parametrów $\{(b_1 = 0.1, b_2 = 0.1), (b_1 = 0.1, b_2 = 0.2), (b_1 = 0.2, b_2 = 0.1), (b_1 = 0.2, b_2 = 0.2)\}$. Przy początkowej liczbie zarażonych wynoszącej 5% populacji, $m = 0.1$. Wyniki przedstawiono na wykresie (1).



Rysunek 1: Dynamika procesu w zależności od parametrów b
Wykresy są w formie wektorowej, można je bezstratnie przybliżać.

3 Podsumowanie

Parametr b (prawdopodobieństwo zarażania) ma istotny wpływ na to, który z procesów będzie miał większą liczbę zarażonych. Jako że zaimplementowano proces SI1SI2S, polegający

na regule „kto pierwszy ten lepszy” to widoczna jest zależność, że większe b przekłada się na większy udział w zarażeniach.

Porównując wykres o parametrach $(b_1 = 0.2, b_2 = 0.1)$ z wykresem o parametrach $(b_1 = 0.2, b_2 = 0.2)$, możemy zauważyć, że prawdopodobieństwo zarażania wirusa ma istotny wpływ na proces konkurenta. W przypadku pierwszego wykresu wirus 1 zaraził miał w szczytowym momencie ponad 400 zarażonych. Gdy zwiększono b_2 do 0.2, liczba zarażeń wirusa 1 w szczytowym momencie zmalała do ponad 200.

Wynika to z konkurencji pomiędzy wirusami, gdzie jeden wirus zabierał zdrowych i ograniczał w ten sposób rozprzestrzenianie się konkurującej choroby.

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

Kod źródłowy 1: Implementacja modelu
Źródło: Opracowanie własne

```
1  from igraph import *
2  import random
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6
7  def simulation(g, m, infection_1_params, infection_2_params):
8      """
9      Concurent SIR Model simulation
10     :param g: graph
11     :param m: probability of recovery
12     :param infection_1_params: (n, b) tuple with n - random seeding , b -
   transmission rate (probability of infection)
13     :param infection_2_params: (n, b) tuple with n - random seeding , b -
   transmission rate (probability of infection)
14     :return: list of graphs in each step, array of S, I1, I2, R counts
15     """
16     g.vs["state"] = "S"
17     g.vs["color"] = "gray"
```

```
    g.vs["size"] = 50

    (n_1, b_1) = infection_1_params
    (n_2, b_2) = infection_2_params

    seeding(g, n=n_1, new_state="I1")
    seeding(g, n=n_2, new_state="I2")
    update_colors(g)

    g_history = [deepcopy(g)]
    counts = [count_states(g)]
    flag = True
    while flag:
        infection(g, b_1, b_2)
        recovery(g, m=m)
        update_colors(g)
        s_count, i_1_count, i_2_count, r_count = count_states(g)
        counts.append((s_count, i_1_count, i_2_count, r_count))
        flag = (i_1_count + i_2_count) != 0
        g_history.append(deepcopy(g))

    return g_history, np.array(counts)


def seeding(g, n, new_state):
    indexes = list(np.where(np.array(g.vs["state"]) == "S")[0])
    indexes_to_infect = random.sample(indexes, n)
    g.vs[indexes_to_infect]["state"] = new_state
    print(g.vs[indexes_to_infect])


def update_colors(g):
    infected_1_indexes = list(np.where(np.array(g.vs["state"]) == "I1")[0])
    g.vs[infected_1_indexes]["color"] = "red"
    infected_2_indexes = list(np.where(np.array(g.vs["state"]) == "I2")[0])
    g.vs[infected_2_indexes]["color"] = "blue"
    recovered_indexes = list(np.where(np.array(g.vs["state"]) == "R")[0])
    g.vs[recovered_indexes]["color"] = "green"


def infection(g, b_1, b_2):
    infected_indexes = get_infected_indexes(g)
    for i in infected_indexes:
        neighbors = g.neighbors(g.vs[i])
        for neighbor in neighbors:
            if g.vs[neighbor]["state"] == "S":
                b = b_1 if g.vs[i]["state"] == "I1" else b_2
                if np.random.rand() < b:
                    g.vs[neighbor]["state"] = g.vs[i]["state"]


```

```
def recovery(g, m):
    infected_indexes = get_infected_indexes(g)
    for i in infected_indexes:
        if np.random.rand() < m:
            g.vs[i]["state"] = "R"


def get_infected_indexes(g):
    infected_1_indexes = list(np.where(np.array(g.vs["state"]) == "I1")[0])
    infected_2_indexes = list(np.where(np.array(g.vs["state"]) == "I2")[0])
    infected_indexes = infected_1_indexes + infected_2_indexes
    random.shuffle(infected_indexes)
    return infected_indexes


def count_states(g):
    infected_1_indexes = list(np.where(np.array(g.vs["state"]) == "I1")[0])
    infected_2_indexes = list(np.where(np.array(g.vs["state"]) == "I2")[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_1_indexes),
        len(infected_2_indexes),
        len(recovered_indexes),
    )


def main():
    graph_size = 1000
    g = Graph.Watts_Strogatz(dim=1, size=graph_size, nei=3, p=0.8)

    transmission_rates = [0.1, 0.2]
    n = int(graph_size * 0.05) # 5% of nodes infected

    fig, axes = plt.subplots(2, 2)
    ax_idx = 0

    for b_1 in transmission_rates:
        for b_2 in transmission_rates:
            g_history, counts = simulation(
                g, m=0.1, infection_1_params=(n, b_1), infection_2_params=(n, b_2)
            )

            ax = axes.flat[ax_idx]
            ax.plot(counts[:, 0], label="Susceptible", c="gray")
            ax.plot(counts[:, 1], label="Infected 1", c="red")
            ax.plot(counts[:, 2], label="Infected 2", c="blue")
            ax.plot(counts[:, 3], label="Recovered", c="green")
            ax.set_title(f"$b_1$ = {b_1}, $b_2$ = {b_2}")
            ax.legend()
```

```
            ax_idx += 1

    plt.tight_layout()
    plt.show()

    # for g in g_history:
    #     plot(g)


if __name__ == "__main__":
    main()
```