Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

LAB 9 - MODELE ADAPTACYJNE

Karol Działowski

nr albumu: 39259

przedmiot: Modelowanie zachowań w sieciach złożonych

Szczecin, 27 grudnia 2020

Spis treści

1 Cel laboratorium

Celem laboratorium było rozwinięcie zadania z Lab 8 (modelu SIR (*Susceptible, Infectious, or Recovered*)) i wprowadzenie parametru dostępności w sieci zgodnie z wybranym prawdopodobieństwem.

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 adaptacyjny wprowadza dodatkowy parametr dostępności, który może służyć modelowaniu różnych poziomów ograniczeń epidemiologicznych. Małe prawdopodobieństwo dostępności, bliskie zeru, odpowiada całkowitemu ograniczeniu rozprzestrzenienia choroby. Duże prawdopodobieństwo dostępności wynoszące jeden jest klasycznym modelem SIR.

Model definiują cztery 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**.

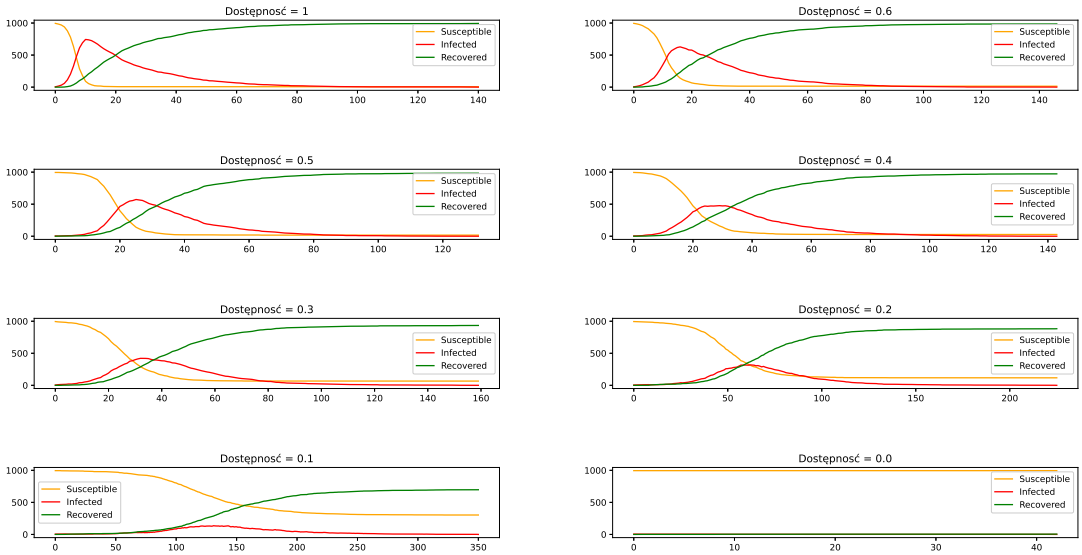
prawdopodobieństwo zarażanie b – określa z jakim prawdopodobieństwem węzeł zaraża sąsiada.

dostępność a – określa z jakim prawdopodobieństwem węzeł jest dostępnym w kroku dystrybucji choroby

2 Wyniki

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

Przeprowadzono symulację dla parametrów $m = 0.05$, $b = 0.2$, $n = 5$, oraz $a \in \{1, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0\}$. Wyniki przedstawiono na wykresie (1).



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

3 Podsumowanie

Parametr dostępności ma istotny wpływ na przebieg procesu dyfuzyjnego w sieci. W przypadku prawdopodobieństwa dostępności $a = 1$ model jest analogiczny do klasycznego modelu

SIR. To znaczy, że wszystkie węzły w etapach zarażania są dostępne i mogą przenieść chorobę lub zostać zarażone.

Wraz z zmniejszaniem prawdopodobieństwo dostępności a występuje wypłaszczenie krzywej zarażeń. To znaczy, że zarażenia są rozłożone w czasie. Ma to kluczowe znaczenie w przypadku epidemii, gdzie przeciążenie systemu medycznego jest niebezpieczne.

Zależność ograniczania dostępności w przypadku badanych parametrów od wartości szczytowej zarażeń nie jest liniowa. Przy ograniczeniu dostępności z $a = 1$ do $a = 0.5$ występuje spadek największej liczby zarażonych w danym czasie z 750 do 550. Przy parametrze $a = 0.3$ największa liczba zarażonych wynosiła 500.

Parametr dostępności oprócz zmniejszania maksymalnej liczby zarażonych (wypłaszczenia krzywej zarażeń) powoduje też rozciągnięcie procesu w czasie. Przy ograniczeniu dostępności $a = 1$ proces zakończył się w 140 iteracjach. Wraz z zmniejszaniem parametru a występuje zwiększenie liczby iteracji po jakich proces dyfuzji się zatrzymuje.

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, n, m, b, a):
8      """
9      SIR Model simulation with availability
10     :param g: graph
11     :param n: size of initial infection
12     :param m: probability of recovery
13     :param b: probability of infection
14     :param a: availability
```

```
    :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:
        set_availability(g, a=a)
        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 set_availability(g, a):
    for vertex in g.vs:
        if np.random.random() < a:
            vertex["availability"] = True
        else:
            vertex["availability"] = False


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:
        if g.vs["availability"]:
```

```
            neighbors = g.neighbors(g.vs[i])
            for neighbor in neighbors:
                if g.vs[neighbor]["state"] == "S" and g.vs[neighbor]["availability"]:
                    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)


def main():
    a_values = [1, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
    fig, axes = plt.subplots(4, 2)
    g = Graph.Watts_Strogatz(dim=1, size=1000, nei=3, p=0.8)

    for idx, a in enumerate(a_values):
        ax = axes.flat[idx]
        g_history, counts = simulation(g, n=5, m=0.05, b=0.2, a=a)
        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'Dostępnosć = {a}')
        ax.legend()

    plt.tight_layout()
    plt.show()

    # for g in g_history:
    #     plot(g)


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