Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

LAB 7 - SOCIAL INFLUENCE

Karol Działowski

nr albumu: 39259

przedmiot: Modelowanie zachowań w sieciach złożonych

Szczecin, 26 listopada 2020

Spis treści

1 Cel laboratorium **1**

2 Zadanie 1 **2**

2.1 Eksperyment 2

3 Zadanie 2 **3**

4 Podsumowanie **4**

Bibliografia **4**

A Implementacja modelu aktywacji progowej **6**

B Implementacja doświadczenia **7**

B.1 Zadanie 1 7

B.2 Zadanie 2 8

1 Cel laboratorium

Celem laboratorium było modelowanie zachowań w sieciach złożonych. Zadanie polegało na implementacji modelu aktywacji progowej (ang. *Linear Threshold*).

Model aktywacji progowej po raz pierwszy był zaproponowany w pracy [2]. Jest to model w którym zbiór agentów S_0 jest aktywny w momencie inicjalizacji sieci i w wyniku

działania symulacji kolejni agenci stają się aktywni jeżeli odpowiednio dużo liczba sąsiadów jest aktywna (stosunek aktywnych sąsiadów jest większy od przyjętego progu n).

2 Zadanie 1

W pierwszym zadaniu należało przygotować sieć WS [3] z około 20 węzłami. Następnie należało zaimplementować w dowolnym języku programowania model aktywacji progowej z wykorzystaniem biblioteki *iGraph* [1]. Gdzie parametrami modelu są:

zadany próg aktywacji n – określa jaki procent węzłów w sąsiedztwie jest wymagany do aktywacji węzła

inicjalna liczba węzłów m – określa liczbę węzłów aktywnych w pierwszej iteracji

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

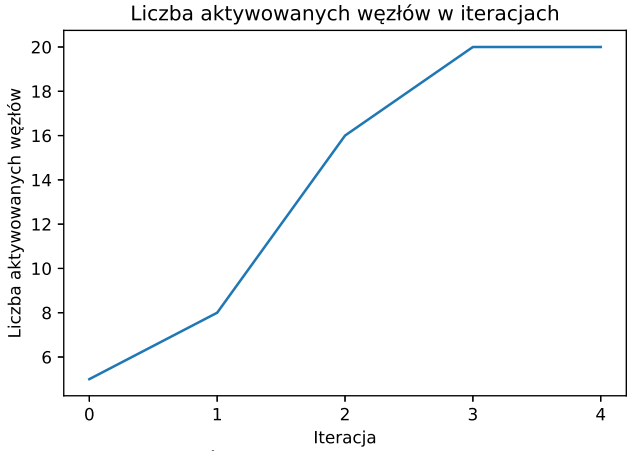
W kolejnych krokach dla każdego węzła sprawdzany jest stan sąsiadów i jeśli liczba węzłów aktywnych jest $\leq n$ to zmieniamy stan węzła na aktywny.

Proces trwa tak długo aż nie wystąpią kolejne aktywacje.

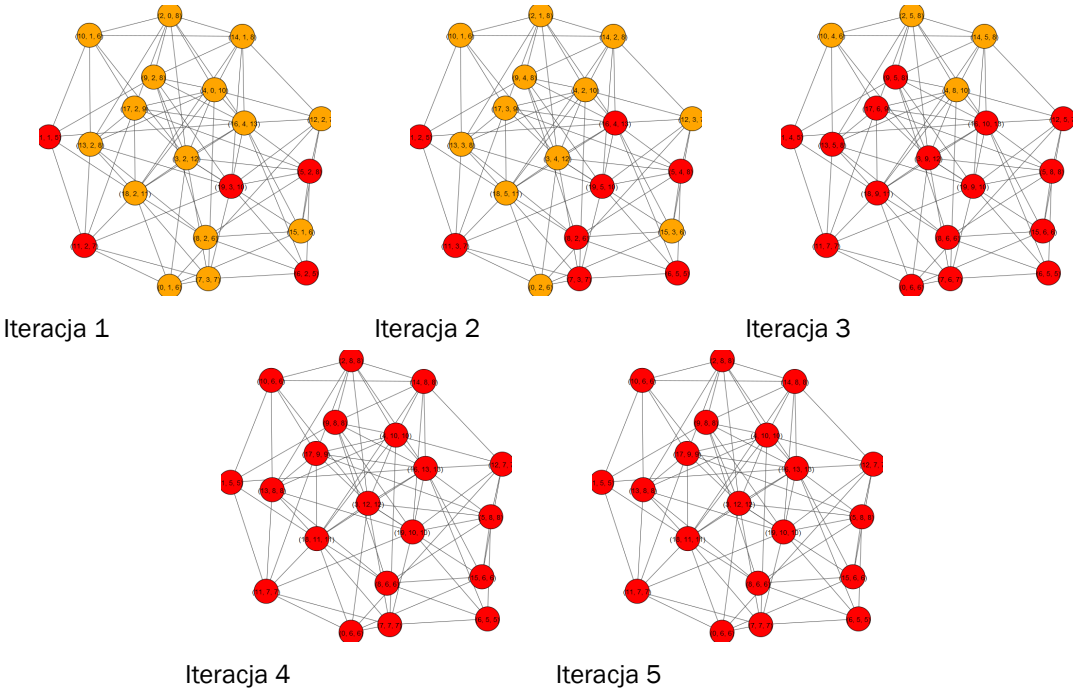
Kod źródłowy przedstawiono na listingu (1).

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 2: Liczba aktywowanych węzłów w iteracjach
Źródło: Opracowanie własne



Rysunek 1: Kroki symulacji modelu aktywacji progowej $m = 5$, $n = 0.3$
Źródło: Opracowanie własne

Na rysunku (1) przedstawione są w węzłach kolejno: (identyfikator węzła, liczba aktywnych sąsiadów, liczba sąsiadów ogółem).

Rysunek (2) przedstawia liczbę aktywowanych węzłów w kolejnych iteracjach.

3 Zadanie 2

Drugie zadanie polegało na przetestowanie modelu na większej sieci (1000 węzłów) i przeprowadzenie symulacji dla różnych parametrów m i n. Przygotowano cztery scenariusze:

1. $m = 5\%$, $n = 10\%$
2. $m = 5\%$, $n = 30\%$
3. $m = 10\%$, $n = 10\%$
4. $m = 10\%$, $n = 30\%$

Zebrane wyniki liczby aktywnych węzłów w iteracjach pokazano na rysunku (3). Wraz ze wzrostem parametru m (liczby początkowo aktywnych węzłów) niezbędna liczba iteracji do aktywacji całej sieci jest mniejsza, czyli zmiany w sieci są propagowane szybciej.

Parametr n wpływa na szybkość rozprzestrzeniania się aktywacji. Gdy parametr n jest mniejszy proces aktywacji wykonuje się w mniejszej liczbie iteracji.

Kod źródłowy przedstawiono na listingu (3).

4 Podsumowanie

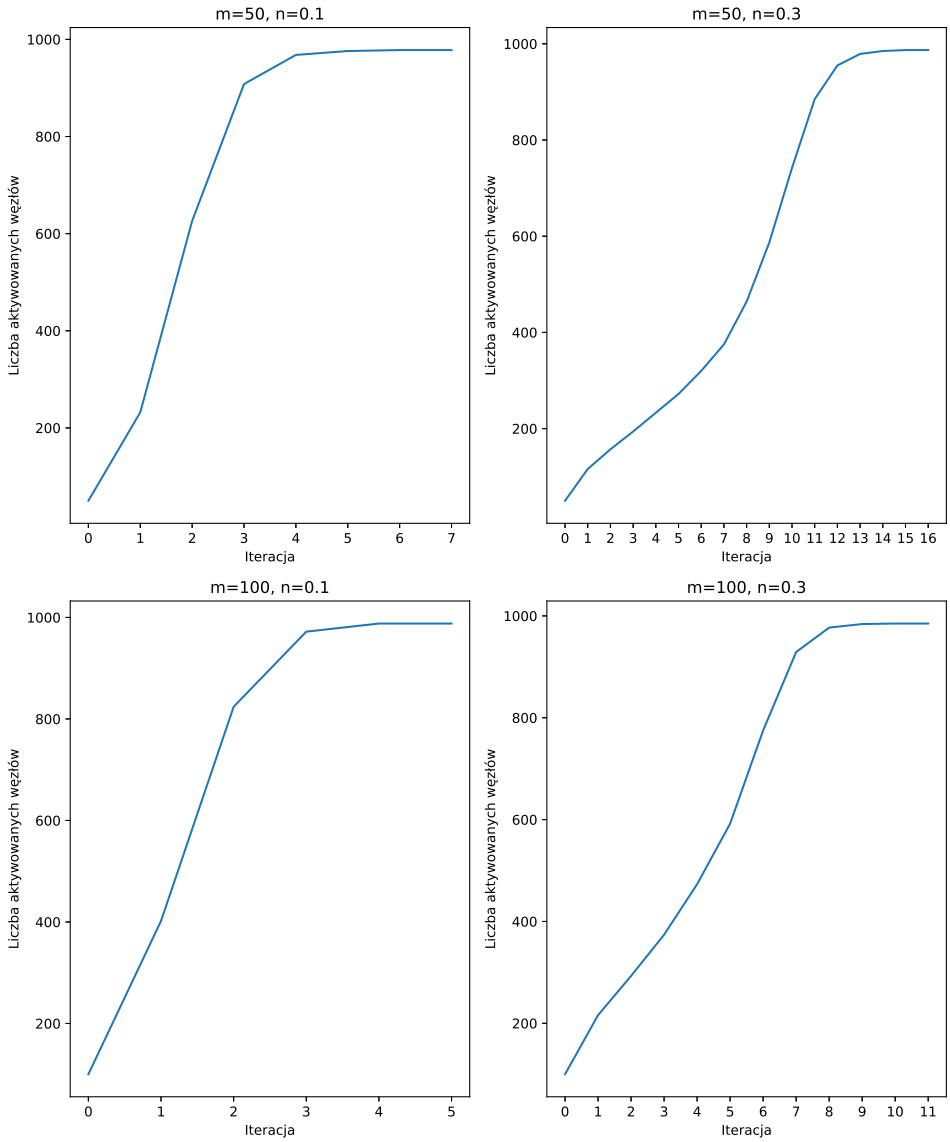
Na laboratorium zaimplementowano model aktywacji progowej w Pythonie, który można dostrajać parametrami m oraz n. Przeprowadzono testy na małej sieci syntetycznej WS oraz dla większej sieci liczącej 1000 węzłów dla różnych parametrów modelu aktywacji progowej.

Bibliografia

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

[2] Granovetter M.: Threshold models of collective behavior. *American journal of sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.

[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.



Rysunek 3: Porównanie symulacji dla różnych parametrów m i n. Wykres przedstawia zależność liczby aktywnych węzłów od iteracji, czyli prezentuje dynamikę aktywacji sieci.

Źródło: Opracowanie własne

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 seeding(g, m):
    """
    Random initialization of m nodes in graph g
    :param g: graph
    :param m: number of nodes to activate
    """
    indexes = g.vs.indices
    indexes_to_activate = random.sample(indexes, m)
    g.vs[indexes_to_activate]["activated"] = True


def update_colors(g):
    """
    :param g: graph
    """
    activated_indexes = list(np.where(g.vs["activated"])[0])
    g.vs[activated_indexes]["color"] = "red"


def update_labels(g):
    """
    :param g: graph
    """
    for i in g.vs.indices:
        g.vs[i]["neighbors_activated_count"] = np.sum([g.vs[n]["activated"] for n in
g.neighbors(g.vs[i])])
    g.vs["label"] = [(g.vs.indices[i], g.vs[i]["neighbors_activated_count"],
g.vs[i]["neighbors_count"]) for i in
                     range(len(g.vs.indices))]


def simulation_step(g, n):
    """
    Perform single iteration. For each node check if percentage of activated
neighbors exceed
    threshold defined by n.
    :param g: graph
    :param n: threshold, required ratio of activated neighbors to activate
    """
    has_activation_happened = False
    for i in g.vs.indices:
```

```
        if not g.vs[i]["activated"]:
            neighbors_count = g.vs[i]["neighbors_count"]
            if neighbors_count:
                neighbors_activated_count = g.vs[i]["neighbors_activated_count"]
                neighbors_activated_ratio = neighbors_activated_count/neighbors_count
                if neighbors_activated_ratio > n:
                    g.vs[i]["activated"] = True
                    has_activation_happened = True

    update_colors(g)
    update_labels(g)
    activated_nodes_count = np.sum(g.vs["activated"])
    return g, activated_nodes_count, has_activation_happened


def simulation(g, m, n):
    g.vs["activated"] = False
    for i in g.vs.indices:
        g.vs[i]["neighbors_count"] = len(g.neighbors(g.vs[i]))
    g.vs["color"] = "orange"
    g.vs["size"] = 50

    seeding(g, m=m)
    update_colors(g)
    update_labels(g)
    # plot(g)

    g_history = [deepcopy(g)]
    activated_nodes_counts = [np.sum(g.vs["activated"])]
    has_activation_happened = True
    while has_activation_happened:
        g, activated_nodes_count, has_activation_happened = simulation_step(g, n)
        g_history.append(deepcopy(g))
        activated_nodes_counts.append(activated_nodes_count)
        # plot(g)
    return g_history, activated_nodes_counts
```

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=4, p=0.8)
g_history, activated_nodes_counts = simulation(g, m=5, n=0.3)
plt.plot(activated_nodes_counts)
plt.xticks(np.arange(len(activated_nodes_counts)))
plt.title("Liczba aktywowanych węzłów w iteracjach")
plt.xlabel("Iteracja")
```

```
plt.ylabel("Liczba aktywowanych węzłów")
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 = 1000
n_values = [0.1, 0.3]
m_values = [int(num_nodes*0.05), int(num_nodes*0.1)]

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

idx = 0
for m in m_values:
    for n in n_values:
        g = Graph.Watts_Strogatz(dim=1, size=num_nodes, nei=2, p=0.8)
        g_history, activated_nodes_counts = simulation(g, m=m, n=n)
        ax = axes.flat[idx]
        ax.plot(activated_nodes_counts)
        ax.set_xticks(np.arange(len(activated_nodes_counts)))
        ax.set_title(f"m={m}, n={n}")
        ax.set_xlabel("Iteracja")
        ax.set_ylabel("Liczba aktywowanych węzłów")
        idx += 1
fig.tight_layout()
fig.savefig("output/zad2_plot.eps")
```