Implementation of Network Analysis using Markov Chains in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to implement network analysis using Markov chains in Python. Networks exist almost everywhere in life. There are networks of computers, people, articles, posts, etc. Network analysis is used to understand the structure, function, and performance of the network. Markov chains method is used to predict the future state based on the present state and not on the previous states.

The basic steps of network analysis using Markov chains are explained: defining network (states, transition matrix, and distribution vector), performing matrix multiplication (computing stationary distribution vector and computing stationary transition vector), performing random walk (computing stationary distribution vector), comparing results, and plotting charts.

The developed program was tested on an experimental data. The program has successfully performed the basic steps of network analysis using Markov chains and provided the required results.

Keywords
Computer Science, Artificial Intelligence, Machine Learning, Network Analysis, Markov Chains, Python, Programming.

1. INTRODUCTION
In the recent years, machine learning has played a major role in the development of computer systems. Machine learning (ML) is a branch of Artificial Intelligence (AI) which is focused on the study of algorithms and methods to improve the performance and efficiency of computer programs [1-11].

Network analysis is an important area in the field of machine learning. It is sharing knowledge with many other fields like: programming, data science, mathematics, statistics, and numerical methods [12-15, 16-20].

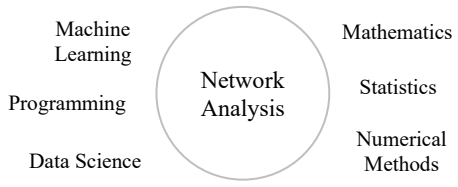

Fig 1: Area of Network Analysis

Network analysis is used to understand the structure, function, and performance of the network. It is performed using Markov chains to predict the future state based on the present state. It is applied in many applications, for example: prediction, generation, ranking, clustering, etc.

2. LITERATURE REVIEW
The literature was reviewed to explore the fundamental concepts, methods, and applications of network analysis using Markov chains [21-26, 27-33].

Network analysis is an important area in machine learning. It has a wide range of applications in different areas like: technology, business, education, psychology, sociology, biology, environment, traffic, sports, etc.

Networks exist everywhere in life. There are networks of computers, people, articles, posts, tweets, words, etc. Simply, network is a set of nodes (or states) and edges (or transitions).

The network model is built to provide a basic representation of the network. Then, the network analysis is performed to understand the network structure, function, performance. It helps to predict the future behavior of the network.

In this research, network analysis is performed using Markov chains. It is a powerful mathematical method used to predict the next state based on the current state and not on the previous states.

Markov chains method was developed by the Russian mathematician Andrey Markov in 1906 [34]. It was used to solve the random processes for large numbers. Now, it is widely used in many fields: mathematics, statistics, computing, weather, sociology, communications, etc.

The fundamental concepts of network analysis using Markov chains are explained in the following section.

Network Analysis:
Network analysis is the process of studying network to understand its structure, function, and performance. Networks exist almost everywhere in life. There are networks of computers, devices, mobiles, students, players, web pages, articles, elements, processes, animals, cities, countries, and so on.

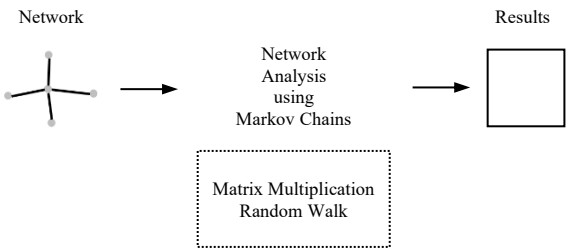

Fig 2: Concept of Network Analysis

Markov Chains:

Markov chains is a mathematical method used to predict the next state based on the current state. The probability of moving from one state to another depends only on the current state and not on the previous states. Simply, Markov chains model is represented by the following formula:

$$P = P(X_{n+1} = S_{n+1} \mid X_n = S_n, X_{n-1} = S_{n-1}, \ldots, X_0 = S_0)$$
$$= P(X_{n+1} = S_{n+1} \mid X_n = S_n)$$

Where: (X) is the random variable, and (*S*) is the state.

The basic concepts of Markov chains are explained in the following section.

Transition Diagram:

Transition diagram is a graph that shows the states and transitions in the network. The state is drawn as a circle and the transition is drawn as an arrow. The following diagram represents a complete transition diagram:

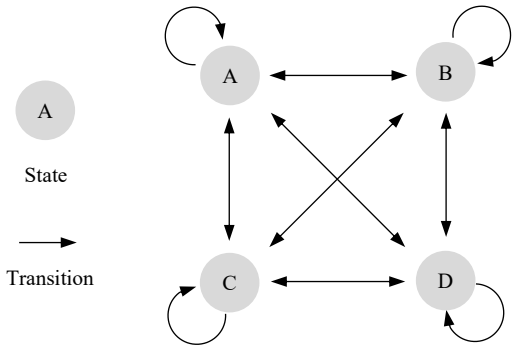

Fig 3: Representation of Transition Diagram

Transition Matrix:

Transition matrix is a matrix of size ($n \times n$) that shows the probabilities of transition between states. It can be represented as shown in the following form:

$$P = \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ i \\ \vdots \\ n\text{-}1 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & \ldots & j & \ldots & n\text{-}1 \end{array} \\ \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,j} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{i,0} & p_{i,1} & \cdots & p_{i,j} & \cdots & p_{i,n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,j} & \cdots & p_{n-1,n-1} \end{bmatrix} \end{array}$$

Fig 4: Representation of Transition Matrix

The rows represent the current states, and the columns represent the next states. The intersections of rows and columns represent the cells. For example, the cell ($p_{i,j}$) represents the probability of transition from the current state (i) to the next state (j).

The transition probability ($p_{i,j}$) is represented by the following formula:

$$p_{i,j} = P(X_{n+1} = j \mid X_n = i)$$

The transition matrix should satisfy the following two conditions:

$$1 \geq p_{i,j} \geq 0$$

and

$$\sum_j p_{i,j} = 1$$

Where each probability ($p_{i,j}$) lies in the range [0, 1] and the sum of probabilities in each row is (1).

Example:

Assume a network of four states (*A*, *B*, *C*, and *D*) with the following transition diagram:

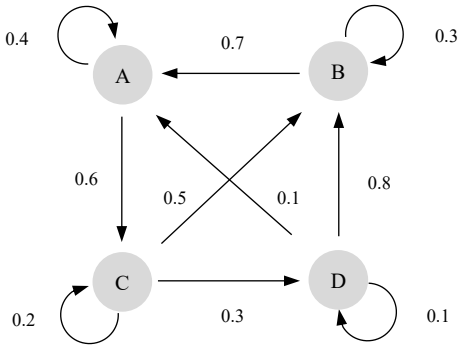

Fig 5: Example of Transition Diagram

Therefore, the transition matrix is represented as shown in the following form:

$$P = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0.4 & 0 & 0.6 & 0 \\ 0.7 & 0.3 & 0 & 0 \\ 0 & 0.5 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0 & 0.1 \end{bmatrix} \end{array}$$

Fig 6: Example of Transition Matrix

Distribution Vector:

Distribution vector is a vector of size (n) that shows the probabilities of states in the network. It can be represented as shown in the following form:

$$\pi = [p_0, \quad p_1, \quad \cdots, \quad p_{n-1}]$$

In this research, network analysis using Markov chains is performed by two methods: matrix multiplication and random walk.

Matrix Multiplication:

The matrix multiplication is used to compute the stationary distribution vector and the stationary transition matrix.

1. Stationary Distribution Vector:

The distribution vector (π) is computed for each iteration as shown in the following steps:

$$\pi_1 = \pi_0 \cdot P$$
$$\pi_2 = \pi_1 \cdot P$$
$$\pi_3 = \pi_2 \cdot P$$
$$\ldots$$
$$\pi_{n+1} = \pi_n \cdot P \qquad (1)$$

After a long run, the distribution vector will be stable and will not change. This indicates that the distribution vector is converging to the stationary form.

$$\therefore \pi = \pi \cdot P \qquad (2)$$

Here, the implementation of matrix multiplication to compute the stationary distribution vector is explained step by step in the following algorithm:

Algorithm 1: Matrix Multiplication to Compute the Stationary Distribution Vector

```
# define states
states = […]
# define transition matrix
T = [[…],
      …
     […]]
# define distribution vector
V = […]
# start with V
V_old = V
# initialize distance
D = []
# max number of iterations
N = 10^6
for t = 1 to N do
     # compute V_new
     V_new = multiply(V_old, T)
     print(t, V_new)
     # compute distance
     d = distance(V_new, V_old)
     # add distance
     D.append(d)
     # check if equal
     if (V_new = V_old) then
          break
     # make V_new as V_old
     V_old = V_new
end for
```

2. Stationary Transition Matrix

The matrix multiplication is used to compute the stationary transition matrix.

Using formula (1) to do further analysis as shown in the following steps:

$$\begin{aligned}\pi_1 &= \pi_0 \cdot P \\ \pi_2 &= \pi_1 \cdot P \\ &= (\pi_0 \cdot P) \cdot P \\ &= \pi_0 \cdot P^2 \\ \pi_3 &= \pi_2 \cdot P \\ &= (\pi_0 \cdot P^2) \cdot P \\ &= \pi_0 \cdot P^3 \\ &\cdots \\ \therefore \pi_n &= \pi_0 \cdot P^n \qquad (3)\end{aligned}$$

This shows that the distribution vector at step "n" (π_n) is the product of the initial distribution vector (π_0) by the transition matrix raised to the power "n" (P^n).

Here, the implementation of matrix multiplication to compute the stationary transition matrix is explained step by step in the following algorithm:

Algorithm 2: Matrix Multiplication to Compute the Stationary Transition Matrix

```
# define states
states = […]
# define transition matrix
T = [[…],
      …
     […]]
# start with T
T_old = T
# max number of iterations
N = 10^6
for t = 1 to N do
     # compute T_new
     T_new = multiply(T_old, T)
     print(t, T_new)
     # check if equal
     if (T_new = T_old) then
          break
     # make T_new as T_old
     T_old = T_new
end for
```

Random Walk:

The random walk is a simulation method used to compute the stationary distribution vector. It assumes a random surfer that moves randomly between states based on their probabilities.

After a long run, the distribution vector will be stable and will not change. This indicates that the distribution vector is converging to the stationary form.

Here, the implementation of random walk to compute the stationary distribution vector is explained step by step in the following algorithm:

Algorithm 3: Random Walk to Compute the Stationary Distribution Vector

```
# define states
states = […]
# define transition matrix
T = [[…],
      … ,
     […]]
# define distribution vector
V = […]
# start with state
current_state = …
# add state to walk path
walk_path = [current_state]
# number of iterations
N = 10^6
for t = 1 to N do
     # select random state
     next_state = random.choice(states, T[current_state])
     # update state count
     V[next_state] += 1
     # add state to walk path
     walk_path.append(next_state)
     # make next_state as current_state
     current_state = next_state
end for
```

```
# normalize V
V = normalize(V)
print(V)
```

Network Analysis System:

The network analysis system is summarized in the following outline:

Input: Network.
Output: Results.
Processing: First, the network is defined (states, transition matrix, and distribution vector). Then, the matrix multiplication is performed to compute the stationary distribution vector and the stationary transition matrix. Next, the random walk is performed to compute the stationary distribution vector. After that, the results are compared and the charts are plotted.

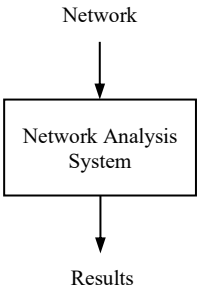

Fig 7: Network Analysis System

Python:

Python [35] is an open source, object-oriented, and general-purpose programming language. It is simple to code, easy to learn, and powerful. It is the most popular programming language especially in the field of machine learning.

Python provides many additional libraries for different purposes. For example: Numpy [36], Pandas [37], Matplotlib [38], Seaborn [39], SciPy [40], NLTK [41], and SK Learn [42].

3. RESEARCH METHODOLOGY

The basic steps of network analysis using Markov chains are: (1) defining network: states, transition matrix, and distribution vector, (2) performing matrix multiplication: computing stationary distribution vector and computing stationary transition matrix, (3) performing random walk: computing stationary distribution vector, (4) comparing results, and (5) plotting charts.

- Defining Network:
  - Defining States
  - Defining Transition Matrix
  - Defining Distribution Vector
- Performing Matrix Multiplication:
  - Computing Stationary Distribution Vector
  - Computing Stationary Transition Matrix
- Performing Random Walk:
  - Computing Stationary Distribution Vector
- Comparing Results
- Plotting Charts

Fig 8: Basic Steps of Network Analysis

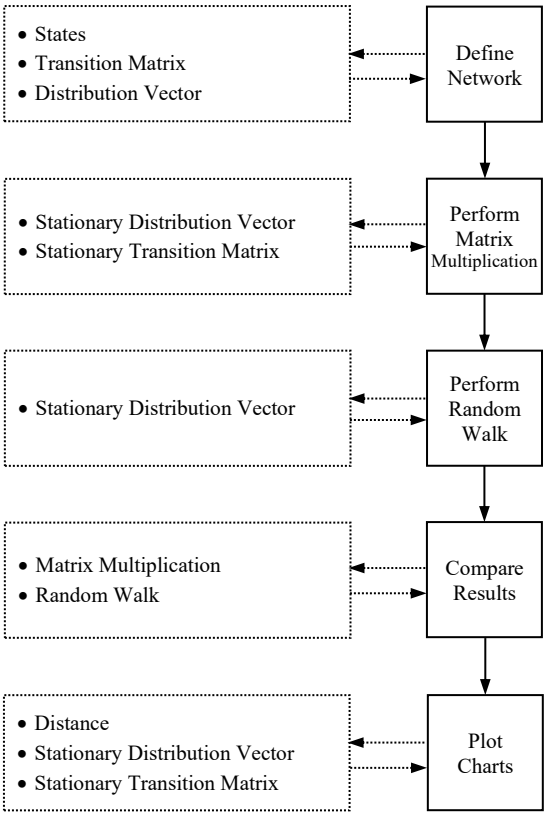

Fig 9: Flowchart of Network Analysis

The basic steps of network analysis using Markov chains are explained in the following section.

Note: The program is developed using only the standard functions of Python without any additional library.

1. Defining Network:

The network is defined by the following steps:

1.1. Defining States:

The states are defined by the following code:

```
states = [0, 1, ..., n-1]
```

1.2. Defining Transition Matrix:

The transition matrix (T) is defined by the following code:

```
T = [[p0,0, p0,1, ..., p0,n-1],
     [p1,0, p1,1, ..., p1,n-1],
      ...
     [pn-1,0, pn-1,1, ..., pn-1,n-1]]
```

1.3. Defining Distribution Vector:

The distribution vector (V) is defined by the following code:

```
V = [p0, p1, ..., pn-1]
```

2. Performing Matrix Multiplication:

The matrix multiplication is performed to compute the stationary distribution vector (V) and the stationary transition matrix (T).

2.1. Computing Stationary Distribution Vector:

The stationary distribution vector (*V*) is computed by the following code:

```
# start with V
V_old = V
# initialize distance
D = []
# max number of iterations
N = 10**6
for t in range(1, N):
    # compute V_new
    V_new = multiply_vm(V_old, T)
    print(t, ":", V_new)
    # compute distance
    d = distance(V_new, V_old)
    # add distance
    D.append(d)
    if equal_v(V_new, V_old, digits):
        break
    # make V_new as V_old
    V_old = V_new
```

The distance between the new and old distribution vectors is computed by the following code:

```
def distance(v1, v2):
    sum = 0
    for i in range(len(v1)):
        sum += (v1[i] - v2[i])**2
    return sum**0.5
```

2.2. Computing Stationary Transition Matrix:

The stationary transition matrix (*T*) is computed by the following code:

```
# start with T
T_old = T
# max number of iterations
N = 10**6
for t in range(1, N):
    # compute T_new
    T_new = multiply_mm(T_old, T)
    print(t, ":")
    print(T_new)
    if equal_m(T_new, T_old, digits):
        break
    # make T_new as T_old
    T_old = T_new
```

The matrix multiplication functions (*multiply_vm*) and (*multiply_mm*) are done by the following code:

```
# dot product of 2 vectors
def dot(v1, v2):
    sum = 0
    for i in range(len(v1)):
        sum += v1[i]*v2[i]
    return sum
# multiply vector by matrix
def multiply_vm(v, m):
    mt = transpose(m)
    t = []
    for i in range(len(v)):
        t.append(dot(v, mt[i]))
    return t
# multiply matrix by matrix
def multiply_mm(m1, m2):
    m2t = transpose(m2)
    t = []
    for i in range(len(m1)):
        row = []
        for j in range(len(m2t)):
            row.append(dot(m1[i], m2t[j]))
        t.append(row)
    return t
```

3. Performing Random Walk: Computing Stationary Distribution Vector:

The random walk is performed to compute the stationary distribution vector (*V*). It is done by the following code:

```
import numpy as np

# select state
current_state = 0
# add state to walk path
walk_path = [current_state]
# number of iterations
N = 10**6
for t in range(1, N):
    # select random state
    next_state = np.random.choice(states,
                 p=T[current_state])
    # update state count
    V[next_state] += 1
    # add state to walk_path
    walk_path.append(next_state)
    # make next_state as current_state
    current_state = next_state
# normalize V
V = normalize(V)
print("V = ", V)
print("Walk Path :")
print(walk_path)
```

The distribution vector (*V*) is normalized by the following code:

```
def normalize(v):
    total = sum(v)
    for i in range(len(v)):
        v[i] /= total
    return v
```

4. Comparing Results:

The results of performing matrix multiplication and random walk are printed by the following code:

```
# Matrix Multiplication
print("(1) Matrix Multiplication:")
print("Stationary Distribution Vector (V):")
print(V_new)
print("Stationary Transition Matrix (T):")
print(T_new)

# Random Walk
print("(2) Random Walk:")
print("Stationary Distribution Vector (V):")
print(V)
```

Then, the results are compared to check if they match or not.

5. Plotting Charts:

The plotting libraries are imported by the following code:

```
import matplotlib.pyplot as plt
import seabron as sns
```

The distance between the new and old distribution vectors (*V_new*, and *V_old*) is plotted by the following code:

```
plt.plot(D)
plt.show()
```

The distribution vector (*V*) is plotted by the following code:

```
plt.bar(V)
plt.show()
```

The transition matrix (*T*) is plotted by the following code:

```
sns.heatmap(T)
plt.show()
```

4. RESULTS AND DISCUSSION

The developed program was tested on an experimental data. The program has successfully performed the basic steps of network analysis using Markov chains and provided the required results. The program output is explained step by step in the following section.

Defining Network:

The network is defined by the following steps:

1. Defining States:

The states are defined and printed as shown in the following view:

```
States = [0, 1, 2, 3]
```

2. Defining Transition Matrix:

The transition matrix (*T*) is defined and printed as shown in the following view:

```
Transition Matrix (T):
[0.2, 0.1, 0.3, 0.4]
[0.6, 0.1, 0.1, 0.2]
[0.1, 0.2, 0.2, 0.5]
[0.4, 0.1, 0.2, 0.3]
```

3. Defining Distribution Vector:

The distribution vector (*V*) is defined and printed as shown in the following view:

```
Distribution Vector (V):
[1, 0, 0, 0]
```

Performing Matrix Multiplication:

The matrix multiplication is performed to compute the stationary distribution vector (*V*) and the stationary transition matrix (*T*).

1. Computing Stationary Distribution Vector:

The distribution vector (*V*) is computed for each iteration and printed as shown in the following view:

```
Distribution Vector (V):
 1: [0.2, 0.1, 0.3, 0.4]
 2: [0.29, 0.13, 0.21, 0.37]
 3: [0.305, 0.121, 0.216, 0.358]
 4: [0.2984, 0.1216, 0.2184, 0.3616]
 5: [0.29912, 0.12184, 0.21768, 0.36136]
 6: [0.29924, 0.121768, 0.217728, 0.361264]
 7: [0.299187, 0.121773, 0.217747, 0.361293]
 8: [0.299193, 0.121775, 0.217741, 0.361291]
 9: [0.299194, 0.121774, 0.217742, 0.36129]
10: [0.299193, 0.121774, 0.217742, 0.36129]
...
```

The stationary distribution vector (*V*) is computed and printed as shown in the following view:

```
Stationary Distribution Vector (V):
[0.299194, 0.121774, 0.217742, 0.36129]
```

2. Computing Stationary Transition Matrix:

The transition matrix (*T*) is computed for each iteration and printed as shown in the following view:

```
Transition Matrix (T):
1:
[0.29, 0.13, 0.21, 0.37]
[0.27, 0.11, 0.25, 0.37]
[0.36, 0.12, 0.19, 0.33]
[0.28, 0.12, 0.23, 0.37]

2:
[0.305, 0.121, 0.216, 0.358]
[0.293, 0.125, 0.216, 0.366]
[0.295, 0.119, 0.224, 0.362]
[0.299, 0.123, 0.216, 0.362]

3:
[0.2984, 0.1216, 0.2184, 0.3616]
[0.3016, 0.1216, 0.2168, 0.36]
[0.2976, 0.1224, 0.2176, 0.3624]
[0.3, 0.1216, 0.2176, 0.3608]
...
```

The stationary transition matrix (*T*) is computed and printed as shown in the following view:

```
Stationary Transition Matrix (T):
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]
```

The stationary distribution vector (*V*) can also be computed using formula (3). It is computed and printed as shown in the following view:

```
Stationary Distribution Vector (V):
[0.299194, 0.121774, 0.217742, 0.36129]
```

Performing Random Walk: Computing Stationary Distribution Vector:

The random walk is performed to compute the stationary distribution vector (*V*). It is computed and printed as shown in the following view:

```
Stationary Distribution Vector (V):
[0.29901, 0.121713, 0.218088, 0.361189]
```

The walk path is computed and printed as shown in the following view:

```
Walk Path:
[0, 1, 3, 0, 2, 3, 0, 3, 3, 2, 2, 0, 2, 3, ...
```

Comparing Results:

The results of matrix multiplication and random walk are printed as shown in the following view:

```
(1) Matrix Multiplication:
Stationary Distribution Vector (V):
[0.299194, 0.121774, 0.217742, 0.36129]

Stationary Transition Matrix (T):
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]
[0.299194, 0.121774, 0.217742, 0.36129]

(2) Random Walk:
Stationary Distribution Vector (V):
[0.29901, 0.121713, 0.218088, 0.361189]
```

Now, by comparing the results, it is clear that they match.

Plotting Charts:

The distance between the new and old distribution vectors (*V_new* and *V_old*) is plotted as shown in the following chart:

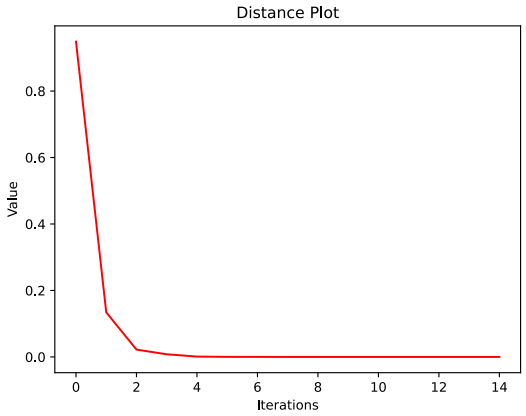

Fig 10: Distance Plot

The plot shows that the distance is decreasing with iterations, which indicates that the distribution vector (*V*) is converging to the stationary form.

Here, the stationary distribution vector (*V*) is plotted as shown in the following chart:

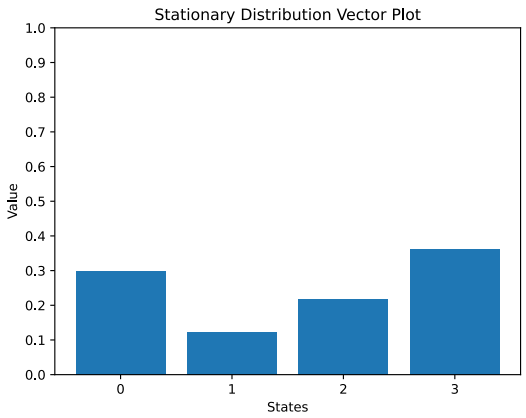

Fig 11: Stationary Distribution Vector Plot

The plot shows the importance of states. They sorted in the following order (3, 0, 2, 1)

Now, the initial and stationary transition matrices (*T* and *T_new*) are plotted as shown in the following charts:

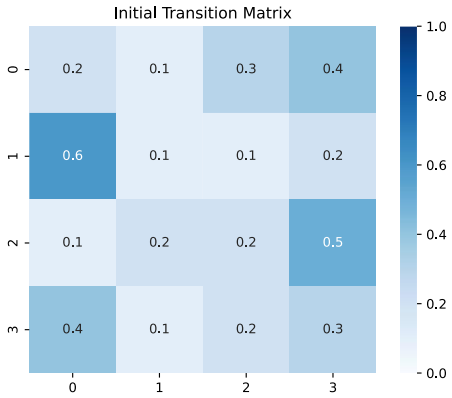

Fig 12: Initial Transition Matrix Heatmap

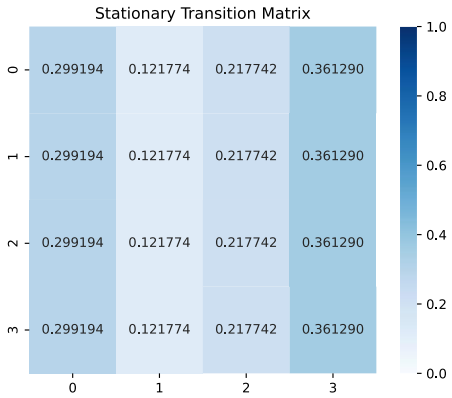

Fig 13: Stationary Transition Matrix Heatmap

The heatmaps show that the transition matrix is converging to the stationary form.

In summary, it is clear that the program has successfully performed the basic steps of network analysis using Markov chains and provided the required results.

5. CONCLUSION

In this research, the goal was to implement network analysis using Markov chains in Python. The literature was reviewed to explore the fundamental concepts of network analysis using Markov chains: network analysis, Markov chains, transition diagram, transition matrix, distribution vector, matrix multiplication, and random walk.

The author developed a program in Python to perform the basic steps of network analysis using Markov chains: defining network (states, transition matrix, and distribution vector), performing matrix multiplication (computing stationary distribution vector and computing stationary transition matrix), performing random walk (computing stationary distribution vector), comparing results, and plotting charts.

The developed program was tested on an experimental data. The program has successfully performed the basic steps of network analysis using Markov chains and provided the required results.

In the future, more work is needed to improve the current methods of network analysis using Markov chains. In addition, they should be more investigated on different fields and domains.

6. REFERENCES

[1] Sammut, C., & Webb, G. I. (2011). "Encyclopedia of Machine Learning". Springer.

[2] Jung, A. (2022). "Machine Learning: The Basics". Springer.

[3] Kubat, M. (2021). "An Introduction to Machine Learning". Springer.

[4] Li, H. (2023). "Machine Learning Methods". Springer.

[5] Zollanvari, A. (2023). " Machine Learning with Python". Springer.

[6] Chopra, D., & Khurana, R. (2023). "Introduction to Machine Learning with Python". Bentham Science Publishers.

[7] Müller, A. C., & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly Media.

[8] Raschka, S. (2015). "Python Machine Learning". Packt Publishing.

[9] Forsyth, D. (2019). "Applied Machine Learning". Springer.

[10] Sarkar, D., Bali, R., & Sharma, T. (2018). "Practical Machine Learning with Python". Apress.

[11] Bonaccorso, G. (2018). "Machine Learning Algorithms: Popular Algorithms for Data Science and Machine Learning". Packt Publishing.

[12] Igual, L., & Seguí, S. (2017). "Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications". Springer.

[13] VanderPlas, J. (2017). "Python Data Science Handbook: Essential Tools for Working with Data". O'Reilly Media.

[14] Muddana, A., & Vinayakam, S. (2024). "Python for Data Science". Springer.

[15] Unpingco, J. (2021). "Python Programming for Data Analysis". Springer.

[16] Zelle, J. (2017). "Python Programming: An Introduction to Computer Science". Franklin, Beedle & Associates.

[17] Xanthidis, D., Manolas, C., Xanthidou, O. K., & Wang, H. I. (2022). "Handbook of Computer Programming with Python". CRC Press.

[18] Chun, W. (2001). "Core Python Programming". Prentice Hall Professional.

[19] Padmanabhan, T. (2016). "Programming with Python". Springer.

[20] Beazley, D. & Jones, B. (2013). "Python Cookbook: Recipes for Mastering Python 3". O'Reilly Media.

[21] Newman, M. (2018). "Networks: An Introduction". Oxford University Press.

[22] Estrada, E. & Knight, P. (2015). "A First Course in Network Theory". Oxford University Press.

[23] Menczer, F., Fortunato, S., & Davis, C. A. (2020). "A First Course in Network Science". Cambridge University Press.

[24] Lewis, T. (2009). "Network Science: Theory and Applications". John Wiley & Sons.

[25] Barabasi, A. (2016). "Network Science". Cambridge University Press

[26] Knickerbocker, D. (2023). "Network Science with Python". Packt Publishing.

[27] Norris, J. (2009). "Markov Chains". Cambridge University Press.

[28] Tolver, A. (2016). "An Introduction to Markov Chains". Department of Mathematical Sciences, University of Copenhagen.

[29] Weber, R. (2011). "Markov Chains". Department of Pure Mathematics and Mathematical Statistics. University of Cambridge.

[30] Gagniuc, P. (2017). "Markov Chains: From Theory to Implementation and Experimentation". John Wiley & Sons.

[31] Ching, W., Huang, S., Ng, M., & Siu, T. (2013). "Markov Chains: Models, Algorithms, and Applications". Springer.

[32] Privault, N. (2018). "Understanding Markov Chains: Examples and Applications". Springer.

[33] Ankan, A., & Panda, A. (2018). "Hands-On Markov Models with Python". Packt Publishing.

[34] Grinstead, C., & Snell, J. (1997). "Introduction to Probability". American Mathematical Society.

[35] Python: http://www.python.org

[36] Numpy: http://www.numpy.org

[37] Pandas: http://pandas.pydata.org

[38] Matplotlib: http://www. matplotlib.org

[39] Seaborn: http://seaborn.pydata.org

[40] SciPy: http://scipy.org

[41] NLTK: http://www.nltk.org

[42] SK Learn: http://scikit-learn.org