Corruption Networks Analysis using Matrices and Eigenvalues

Alvin Christopher Santausa, 13523033[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523033@itb.ac.id, [2]alvinchrisantausa@gmail.com

***Abstract*—Corruption is a massive issue, especially in Indonesia, where it is happening in various sectors, including government, business, and others. Corruption is done with complex networks, where actors and their interactions form patterns that can be analyzed mathematically using matrices and eigenvalues. Actors as nodes and the interactions as edges form a network that can be studied to retrieve information about influential individuals, clusters of corrupt activity, and the overall structure of corruption. This paper will apply principles of linear and geometry algebra to model, analyze, and interpret corruption networks, leveraging adjacency matrices and eigenvalue analysis to identify components of the corruption network.**

***Keywords*—corruption, corruption networks, eigenvalue, eigenvector, adjacency matrices, network analysis, centrality.**

I. Introduction

Corruption is a serious worldwide issue that affects many sectors, including government, economy, and society. This act must be stopped as it only has a negative effect and has no value offered. It usually involves illegal activities that are often carried out through secret networks of connections. It is essential to analyze the networks formed by corrupt entities to understand the complex networks better. These networks often exhibit intricate structures that can be unraveled using mathematical tools such as matrices and eigenvalues. Linear algebra, particularly matrix factorization techniques, provides powerful methods to represent and analyze relationships within corruption networks, also revealing insights into key players and their influence within the networks.

This paper will explore how matrix operations and eigenvalues can be used to model and analyze corruption networks. By using adjacency matrices, eigenvector centrality, and spectral clustering concepts, hidden patterns can be analyzed and uncovered within the corruption networks. This approach will enable us to identify influential nodes (persons), hidden subgroups, and potential vulnerabilities in corrupt structures.

II. Theoretical Foundations

A. Graph Representation of Networks

Graph is a pair $G = (V, E)$, where V is a set that its elements are called vertices and E is a set of unordered pairs $\{v1, v2\}$ of vertices (vertex), whose elements are called edges (links or lines). It is used to model pairwise relations are a fundamental representation of networks, allowing for the modeling of relationships between entities in a system.

In the case of corruption networks, the entities (individuals, organizations, or government bodies) are represented as nodes (vertices), while the interactions between them (such as bribery, illegal exchanges, or collusion) are represented as edges (links). Graphs can be classified as either directed or undirected, with directed graphs being particularly useful for representing an asymmetric relationship, such as those seen in corruption.

A directed graph (digraph) is especially suited for corruption analysis, where an edge from node 1 to node 2 indicates a flow or transaction (bribe or other) from entity 1 to entity 2. Such a graph can represent both direct and indirect relationships, and its analysis will help to detect the flow of corrupt practices, and the key players involved.

B. Matrix and Adjacency Matrix

Matrix is a rectangular array of numbers, symbols, or expressions arranged in rows and columns. Matrices are used to represent linear transformations, solve systems of linear equations, and others. A matrix A with m rows and n columns is denoted as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

where a_{ij} represents the element in the ith row and jth column.

Matrix can be also used to represent a finite graph using an adjacency matrix. An $n \times n$ adjacency matrix is a square matrix that represents a graph with n vertices where:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

In the context of corruption analysis, the adjacency matrix can represent the presence or absence of connections between individuals in the network. For weighted graphs, where edges have different strengths or importance, the entries can be non-binary values representing the weight of the connections.

C. Eigenvalues and Eigenvectors

Eigen is a German word that means “own” or “characteristic”. Eigen is used in linear algebra to describe intrinsic properties of matrices and linear transformations.

In the context of matrices, an eigenvector $\boldsymbol{x}$ is a nonzero vector such that the action of a matrix A on $\boldsymbol{x}$ merely scales $\boldsymbol{x}$, rather than changing its direction. This can be expressed as:

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

Where:

- λ is the eigenvalue, a scalar representing the scaling factor.
- $\boldsymbol{v}$ is the eigenvector, which retains its direction after the transformation by A

To determine eigenvalue (λ) and eigenvector ($\boldsymbol{x}$), the characteristic equation is,

$$\det(A - \lambda I) = 0$$

Where I is the identity matrix with the same size as A, and det is determinant.

After solving the equation, get the eigenvalue and substitute it to this equation to get the eigenvector

$$(A - \lambda I)x = 0$$

Eigen centrality is a concept derived from eigenvectors, primarily used in network analysis. It measures the importance of a node within a graph based on the principle that connections to highly connected nodes contribute more to a node’s centrality. The eigen centrality vector ($\boldsymbol{x}$) with adjacency matrix A in graph satisfies

$$A\boldsymbol{x} = \lambda_{max}\boldsymbol{x}$$

The λ_{max} is the largest eigenvalue (dominant eigenvalue / principal eigenvalue) so that the $\boldsymbol{x}$ is principal eigenvector that represents the centrality scores. Eigenvector x is usually normalized to ensure that the scores are scale-invariant and comparable. In terms of corruption networks, eigen centrality can be used to get information about key actors, most influential actors, and network vulnerabilities.

III. IMPLEMENTATION

In this implementation, I will demonstrating the use of eigenvalue and eigenmatrix at 2 cases

A. Simple corruption case

B. More complex corruption case

A. Simple corruption case

In this first case, I will demonstrate how to analyze a simple corruption case that form a simple corruption network (consists of 3 actors/nodes) (this is a dummy case).

Actor’s Name	Code
Person 1	A
Person 2	B
Person 3	C

No	Details
1	Person 1 (A) send illegal fund (IDR 2 billion) to person 2 (B)
2	Person 1 (A) send illegal fund (IDR 1 billion) to person 3 (C)
3	Person 2 (B) send illegal fund (IDR 500 million) to person 1 (A)
4	Person 2 (B) send illegal fund (IDR 1 billion) to person 3 (C)
5	Person 3 (C) send illegal fund (IDR 3 billion) to person 2 (C)

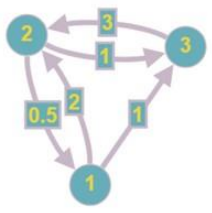


Figure 1. Simple Corruption Network Graph
Source: https://github.com/Incheon21/AlgeoMakalah

Then can the adjacency matrix

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 0.5 & 0 & 1 \\ 0 & 3 & 0 \end{bmatrix}$$
*units in trillion

From the adjacency matrix, find eigenvalue to get the eigen vector. First, calculate the eigenvalue using the formula mentioned before

$$\det(\lambda I - A) = 0$$

$$\det(\lambda I - A) = \det\left(\begin{bmatrix} \lambda & -2 & -1 \\ -0.5 & \lambda & -1 \\ 0 & -3 & \lambda \end{bmatrix}\right)$$

$$= (\lambda).\det\left(\begin{bmatrix} \lambda & -1 \\ -3 & \lambda \end{bmatrix}\right) - (-2).\det\left(\begin{bmatrix} -0.5 & -1 \\ 0 & \lambda \end{bmatrix}\right) + (-1).\det\left(\begin{bmatrix} -0.5 & \lambda \\ 0 & -3 \end{bmatrix}\right)$$

$$= \lambda(\lambda^2 - 3) + 2(-0.5\lambda) - (1.5)$$

$$= \lambda^3 - 3\lambda - \lambda - 1.5$$

$$= \lambda^3 - 4\lambda - 1.5 = 0$$

Using a calculator, the eigenvalues are
$\lambda_1 = 2.167,\ \lambda_2 = -0.389, \lambda_3 = -1.776$

Then take the largest eigenvalue (principal eigenvalue) $\lambda_1 = 2.677$ and calculate the eigenvector by substitute $\lambda = 2.677$ into the equation $(\lambda I - A)x = 0$

$$(\lambda I - A) = \begin{bmatrix} 2.167 & -2 & -1 \\ -0.5 & 2.167 & -1 \\ 0 & -3 & 2.167 \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} 2.167 & -2 & -1 \\ -0.5 & 2.167 & -1 \\ 0 & -3 & 2.167 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

From the equation, the x values can be found

$$\begin{cases} 2.167x_1 - 2x_2 - x_3 = 0 \\ -0.5x_1 + 2.167x_2 - x_3 = 0 \\ -3x_2 + 2.167x_3 = 0 \end{cases}$$

From the third equation

$$x_2 = \frac{2.167}{3} x_3 \approx 0.722x_3$$

Substitute x_2 into the second equation

$$-0.5x_1 + 2.167(0.722x_3) - x_3 = 0$$

$$-0.5x_1 + 0.565x_3 = 0$$

$$x_1 = -\frac{0.5654}{0.5} x_3 \approx 1.129x_3$$

Let x_3 = t, then:

$$x \approx \begin{bmatrix} 1.129t \\ 0.722t \\ t \end{bmatrix} = t \begin{bmatrix} 1.129 \\ 0.722 \\ 1 \end{bmatrix}$$

After getting the eigenvector for the largest eigenvalue (principal eigenvalue), then calculate the centrality scores by normalizing the eigenvector.

$$\|x\| = \sqrt{1.129^2 + 0.722^2 + 1^2}$$

$$\approx \sqrt{1.274 + 0.521 + 1} = \sqrt{2.795} \approx 1.671$$

$$c = -\frac{x}{\|x\|} \approx \begin{bmatrix} \frac{1.129}{1.671} \\ \frac{0.722}{1.671} \\ \frac{1}{1.671} \end{bmatrix} \approx \begin{bmatrix} 0.676 \\ 0.432 \\ 0.598 \end{bmatrix}$$

In the matrix c, each person's centrality score is showed from A to C with the highest score is for person A.

Person A: 0.676
Person B: 0.432
Person C: 0.598

The centrality scores indicate that person A is the most influential individual within the corruption network, followed by person C and person B. The influence is formed by the number and importance of their connections, as well as the fact that they are connected to other influential individuals.

This result shows the use of eigenvector centrality for uncovering the relative importance of individuals in a corruption network. By identifying the key players (most influential), anti-corruption party could focus their efforts on individuals with the highest centrality because stopping their activities could have the most significant impact so the corruption can be stopped.

B. More complex case

For more complex case, I will use recent corruption case in Indonesia for a study case. This case involved by many parties including Harvey Moeis and Helena Lim as the main suspect. But, because there is no valid and enough data found, I use an assumption that recent corruption case is happened like the table below (this is a dummy case).

Actor's Name	Code
Harvey Moeis	H
Helena Lim	L
Government Official 1	G1
Government Official 2	G2
Smelter 1	S1
Smelter 2	S2
Money Launderer	Z

No	Details
1	Harvey Moeis (H) bribed a Government Official 1 (G1) for project approval (IDR 100 trillion)
2	Harvey Moeis (H) sent money to Helena Lim (H) to help him convinced the Government Official 1 (G1) (IDR 70 trillion)
3	Harvey Moeis (H) also sent money to each Smelter 1 (S1) and Smelter 2 (S2) each IDR 10 trillion to make sure his plan runs smoothly
4	Helena Lim (L) also involved by providing funds and connections to Government Official 1 (G1) (IDR 50 trillion)
5	Government Official 1 (G1) transferred funds to Government Official 2 (G2) for coordination (IDR 40 trillion)

6	Government Official 2 (G2) transferred funds to Smelter 1 (S1) for smelting operations (IDR 30 trillion)
7	Smelter 1 (S1) processed and transferred funds to smelter 2 (S2) for coordination (IDR 10 trillion)
8	Smelter 1 (S1) and Smelter 2 (S2) sent funds to launder (Z) for money laundering (IDR 20 trillion)
9	Launder (Z) transfered money to Harvey Moeis (H) (IDR 273 trillion)

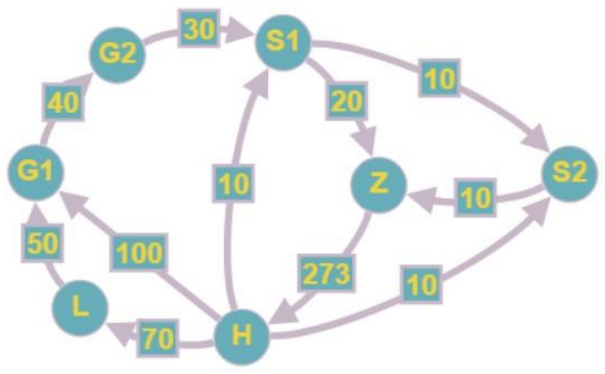


Figure 2. Complex Corruption Network Graph
Source: https://github.com/Incheon21/AlgeoMakalah

First, make the adjacency matrix from the action table and it will form like this

$$A = \begin{bmatrix} 0 & 50 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 40 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 20 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 273 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

From the adjacency matrix, the eigenvalue can be found to get the eigen vector. But with this more complex case a program will be used to find the eigenvalue, eigenvector, and the eigen centrality score.

Normalize function to normalize the eigenvector

```
def normalize(vector):
  squared_sum = sum(x*x for x in vector)
  magnitude = squared_sum ** 0.5

  if magnitude == 0:
    return vector
  return [x/magnitude for x in vector]
```

Solve_linear_system function to solve a linear system of equations $(A - \lambda I)x = 0$ using Gaussian elimination

```
def solve_linear_system(matrix, precision=1e-
10):
  n = len(matrix)
  augmented = [row[:] + [0] for row in
matrix]

  # Forward elimination
  for i in range(n):
    pivot = augmented[i][i]
```

```
    if abs(pivot) < precision:
      for j in range(i+1, n):
        if abs(augmented[j][i]) > precision:
          augmented[i], augmented[j] =
augmented[j], augmented[i]
          pivot = augmented[i][i]
          break

    if abs(pivot) < precision:
      continue

    for j in range(i+1, n):
      factor = augmented[j][i] / pivot
      for k in range(i, n+1):
        augmented[j][k] -= factor *
augmented[i][k]

  # Back substitution
  solution = [1.0] * n
  for i in range(n-1, -1, -1):
    s = sum(augmented[i][j] * solution[j] for
j in range(i+1, n))
    if abs(augmented[i][i]) > precision:
      solution[i] = -s / augmented[i][i]

  return normalize(solution)
```

Solve_linear_system function to calculate eigenvectors from a list of eigenvalues

```
def find_eigenvectors(matrix, eigenvalues,
precision=1e-10):
  """Find eigenvectors for each
eigenvalue."""
  n = len(matrix)
  eigenvectors = []

  for eigenval in eigenvalues:
    # Construct matrix A - λI
    shifted_matrix = [
      [matrix[i][j] - (eigenval if i == j
else 0)
      for j in range(n)]
      for i in range(n)
    ]
    eigenvector =
solve_linear_system(shifted_matrix,
precision)
    eigenvectors.append(eigenvector)

  return eigenvectors
```

- get_minor function to compute the minor matrix for calculating determinant
- determinant function to calculate the determinant of a matrix with recursion
- evaluate_characteristic_polynomial function to evaluate the characteristic polynomial $P(x) = det(A - \lambda I)$
- derivative_characteristic function to estimate the derivative of the characteristic polynomial

```
def get_minor(matrix, i, j):
  """Get the minor matrix by removing row i
and column j."""
  return [[matrix[row][col] for col in
range(len(matrix)) if col != j]
          for row in range(len(matrix)) if
row != i]

def determinant(matrix):
  """Calculate determinant recursively using
cofactor expansion."""
  if len(matrix) == 1:
    return matrix[0][0]
  if len(matrix) == 2:
    return matrix[0][0]*matrix[1][1] -
matrix[0][1]*matrix[1][0]

  det = 0
  for j in range(len(matrix)):
    cofactor = (-1) ** j * matrix[0][j] *
determinant(get_minor(matrix, 0, j))
    det += cofactor
  return det

def
evaluate_characteristic_polynomial(matrix,
x):
  """Evaluate characteristic polynomial at x
for any size matrix."""
  n = len(matrix)
  # Create matrix A - xI
  shifted = [[matrix[i][j] - (x if i == j
else 0) for j in range(n)] for i in range(n)]
  return determinant(shifted)

def
derivative_characteristic_polynomial(matrix,
x, h=1e-7):
  """Numerical derivative of characteristic
polynomial."""
  return
(evaluate_characteristic_polynomial(matrix, x
+ h) -
          evaluate_characteristic_polynomial(
matrix, x)) / h
```

Is_duplicate_eigenvalue function to check if an eigenvalue is already computed

```
def is_duplicate_eigenvalue(value,
eigenvalues, tolerance=1e-6):
  """Check if an eigenvalue is already in the
list within tolerance."""
  return any(abs(value - ev) < tolerance for
ev in eigenvalues)
```

Solve_eigenvalues function to find eigenvalues of the matris using Newton’s method

```
def solve_eigenvalues(matrix,
initial_guesses, max_iter=100, tolerance=1e-
10):
  """Find eigenvalues using Newton's method
with duplicate checking."""
  eigenvalues = []
  n = len(matrix)

  for guess in initial_guesses:
    x = guess
    converged = False

    for _ in range(max_iter):
      fx =
evaluate_characteristic_polynomial(matrix, x)
      if abs(fx) < tolerance:
        converged = True
        break

      dfx =
derivative_characteristic_polynomial(matrix,
x)
      if abs(dfx) < tolerance:
        break

      x_new = x - fx/dfx
      if abs(x_new - x) < tolerance:
        converged = True
        break
          `
      x = x_new

    if converged and not
is_duplicate_eigenvalue(x, eigenvalues):
      eigenvalues.append(x)
      if len(eigenvalues) == n:  # Found all
eigenvalues
        break

  return eigenvalues
```

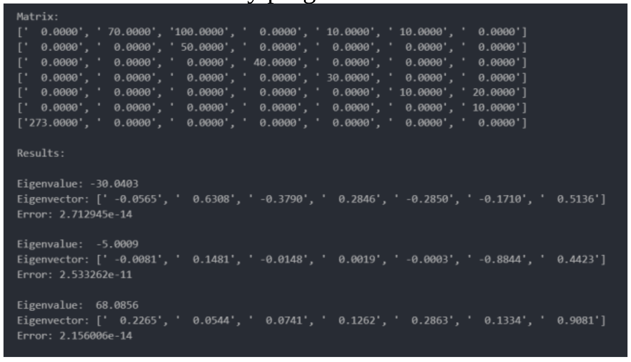
- generate_initial_guesses function to generate initial guesses for eigenvalues (to accelerate the computation)
- verify_eigenvalues function to verify the accuracy of computed eigenvalue and eigenvector

```
def generate_initial_guesses(matrix):
  n = len(matrix)
  """Generate better initial guesses for
eigenvalues."""
  guesses = []

  # Add diagonal elements
  for i in range(n):
    guesses.append(matrix[i][i])

  # Add Gershgorin disk centers and radii
  for i in range(n):
    radius = sum(abs(matrix[i][j]) for j in
range(n) if i != j)
    guesses.append(matrix[i][i] + radius)
    guesses.append(matrix[i][i] - radius)

  # Add trace/n as it's the average of
eigenvalues
  trace = sum(matrix[i][i] for i in range(n))
  guesses.append(trace / n)

  # Remove duplicates and sort
  return sorted(list(set(guesses)))

def verify_eigenvalues(matrix, eigenvalues,
eigenvectors, tolerance=1e-8):
  """Verify that computed eigenvalues and
eigenvectors are correct."""
  results = []
  n = len(matrix)
  for eigenval, eigenvec in zip(eigenvalues,
eigenvectors):
    # Calculate Ax
    Ax = [sum(matrix[i][j] * eigenvec[j] for
j in range(n)) for i in range(n)]
    # Calculate λx
    lx = [eigenval * x for x in eigenvec]
    # Check if Ax = λx
    error = sum((Ax[i] - lx[i])**2 for i in
range(n))**0.5
    results.append({
      'eigenvalue': eigenval,
      'eigenvector': eigenvec,
      'error': error,
      'valid': error < tolerance
    })
  return results
```

Main program to calculate the adjacency matrix

```
A = [
  [0, 70, 100, 0, 10, 10, 0],
  [0, 0, 50, 0, 0, 0, 0],
  [0, 0, 0, 40, 0, 0, 0],
  [0, 0, 0, 0, 30, 0, 0],
  [0, 0, 0, 0, 0, 10, 20],
  [0, 0, 0, 0, 0, 0, 10],
  [273, 0, 0, 0, 0, 0, 0]
]

"""Solve for eigenvalues and eigenvectors of
a matrix of any size."""
n = len(A)
initial_guesses = generate_initial_guesses(A)
eigenvalues = solve_eigenvalues(A,
initial_guesses)

# Ensure we have exactly n eigenvalues
if len(eigenvalues) < n:
  print(f"Warning: Found only
{len(eigenvalues)} eigenvalues out of {n}
expected")

eigenvectors = find_eigenvectors(A,
eigenvalues)
verification = verify_eigenvalues(A,
eigenvalues, eigenvectors)

print("\nMatrix:")
for row in A:
  print([f"{x:8.4f}" for x in row])

print("\nResults:")
for result in verification:
  if result['valid']:
    print(f"\nEigenvalue:
{result['eigenvalue']:8.4f}")
    print("Eigenvector:", [f"{x:8.4f}" for x
in result['eigenvector']])
    print(f"Error: {result['error']:e}")
  else:
    print(f"\nWarning: Invalid
eigenvalue/eigenvector pair (error:
{result['error']:e})")
    print("Eigenvalue:",
result['eigenvalue'])
    print("Eigenvector:",
result['eigenvector'])
```

The calculation done by program is



```
Matrix:
['  0.0000', ' 70.0000', '100.0000', '  0.0000', ' 10.0000', ' 10.0000', '  0.0000']
['  0.0000', '  0.0000', ' 50.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000']
['  0.0000', '  0.0000', '  0.0000', ' 40.0000', '  0.0000', '  0.0000', '  0.0000']
['  0.0000', '  0.0000', '  0.0000', '  0.0000', ' 30.0000', '  0.0000', '  0.0000']
['  0.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000', ' 10.0000', ' 20.0000']
['  0.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000', ' 10.0000']
['273.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000', '  0.0000']

Results:

Eigenvalue: -30.0403
Eigenvector: [' -0.0565', '  0.6308', ' -0.3790', '  0.2846', ' -0.2850', ' -0.1710', '  0.5136']
Error: 2.712945e-14

Eigenvalue:  -5.0009
Eigenvector: [' -0.0081', '  0.1481', ' -0.0148', '  0.0019', ' -0.0003', ' -0.8844', '  0.4423']
Error: 2.533262e-11

Eigenvalue:  68.0856
Eigenvector: ['  0.2265', '  0.0544', '  0.0741', '  0.1262', '  0.2863', '  0.1334', '  0.9081']
Error: 2.156006e-14
```

Figure 1. Calculation Program Result
Source: https://github.com/Incheon21/AlgeoMakalah

The biggest eigenvalue is 68.05 and the eigenvector is

$$x = \begin{bmatrix} 0.2265 \\ 0.0544 \\ 0.0741 \\ 0.1262 \\ 0.2863 \\ 0.1334 \\ 0.9081 \end{bmatrix}$$

That eigenvector is also the centrality scores of the adjacency matrix because that is a principal eigenvector and is already normalized The centrality score showed that Money Launderer (Z) or node 7 has the biggest centrality score so it means that it is the most influential individual.

Though H or L might be seen as a dominant individual because they initiate and start the corruption, the Money Launderer (Z) has the biggest centrality score. This suggests that targeting the Money Launderer (Z) could give the greatest impact in disrupting the corruption network. This also indicates the network's resilience hinges on the operational efficiency of the launderer.

By findings focusing on the individual with high centrality scores, anti-corruption efforts can become more systematic and data-driven and result in more effective interventions and better outcomes.

IV. Some Common Mistakes

The analysis on corruption networks using mathematical approaches may result in different or inaccurate conclusions because of several common mistakes that occur in the process of analysis. The data representation has to be complete data so there is no missing actions/transactions/actors that may lead to an incomplete adjacency matrix that affects the accuracy of eigenvalue and centrality calculations.

Treating all connections as equally important can oversimplify the network. That is why weighted graph is used in the implementation as it reflects the magnitude or significance of transactions to ensure a more realistic analysis.

Improper normalization and misunderstanding the meaning of eigenvector centrality or other metrices can also lead to misguided conclusions.

By addressing these common mistakes, the findings on the analysis can be robust and insightful for stopping corruption networks.

V. Conclusion

This study demonstrates the application of linear algebra, particularly matrix eigenvalue, and eigenvector analysis, in covering patterns and key actors within corruption networks. By using and analyzing adjacency matrices, the centrality score of nodes can be determined and it helps to identify the influential individuals and critical connections.

In the implementation part, the simple corruption case showed that eigenvector centrality successfully highlighted the primary role of Person A, proofing the method's effectiveness in identifying dominant actors in small networks. In the more complex case, the analysis identified the Money Launderer (Z) as the most influential individual, emphasizing the necessity of targeting such actors to disrupt a big systemic corruption.

These findings emphasize that mathematical tools can provide a structured and objective approach to analyzing corruption networks. It offers insights that can guide anti-corruption party to ensure the interventions are focused on the most impactful person (node). Future work and deeper analysis could explore extending these methods to dynamic or evolving networks, integrating time-based analysis to track changes in network structures over time. By using these techniques, authorities and researchers can work together to address corruption systematically, paving the way for more transparent and clean systems.

VI. Appendix

1. Github Repository: https://github.com/Incheon21/AlgeoMakalah

VII. Acknowledgment

The author expresses heartfelt gratitude to Almighty God for the blessings and guidance during the writing of this paper. Special thanks are extended to Ir. Rila Mandala, M.Eng., Ph.D., for the role as lecturer in the IF2123 Linear and Geometry Algebra course and to Dr. Ir. Rinaldi Munir, M.T., for making the lecture materials available on the course website, which supported the research process. The author also wishes to acknowledge the unwavering support from family and friends that were invaluable in completing this paper.

References

[1] Munir, Rinaldi. 2023. “Nilai Eigen dan Vektor Eigen (Bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf [Accessed: Jan, 2, 2025]

[2] Balakrishnan, V. K. (1997). *Graph Theory* (1st ed.). McGraw-Hill. ISBN 978-0-07-005489-9 [Accessed: Jan, 1, 2025]

[3] Jiang, Y. (2022). Study on eigenvalue and eigenvector introduction. *Journal of Physics Conference Series*, *2282*(1), 012004. https://doi.org/10.1088/1742-6596/2282/1/012004 [Accessed: Jan, 1, 2025]

[4] *Paul Minogue*. (n.d.). https://paulminogue.com/posts/8cdb1f03-f215-4060-908f-d21d403bf9e5 [Accessed: Jan, 1, 2025]

Statement of Orginality

I hereby declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not plagiarism..

Bandung, 2nd Desember 2024

[illegible]

Alvin Christopher Santausa
13523033