Application of Bipartite Graphs in Identifying Strategic Chess Piece Exchanges

Muhammad Edo Raduputu Aprima - 13523096[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*muhammadedo017@gmail.com*, *13523096@std.stei.itb.ac.id*

***Abstract*—The exchange of chess pieces plays a critical role in shaping the dynamic of the game, often determining the strategic advantage of one player over one player. This paper investigates the use of bipartite graphs to model and analyze chess piece exchanges, providing a mathematical framework to evaluate their impact. By representing white and black pieces as two disjoint sets of vertices and their potential interaction as edges, this study employs maximum matching techniques to identify advantageous exchanges. Edge weights are assigned based on piece values and positional significance, reflecting their strategic importance.**

***Keywords*—chess, piece exchanges, bipartite graphs, optimization .**

I. Introduction

Chess is a strategic and calculative game in which players must continuously assess the worth of their pieces and make decisions to enhance their position or reduce their opponent's opportunities. A vital element of chess strategy is the exchange of pieces, particularly during the middlegame. The capacity to assess the benefits of a trade distinguishes novice players from experienced ones. This endeavor is intricate because to the dynamic and multidimensional characteristics of chess positions.

During the middlegame, assessing piece trades requires consideration of both the numerical values attributed to pieces—such as the queen (9), rook (5), bishop (3), knight (3), and pawn (1)—and the strategic ramifications of those exchanges. An ostensibly equivalent material swap may yield long-term ramifications, including enhanced positioning, compromised pawn structures, or augmented king safety. Conventional methods for assessing piece exchanges predominantly depend on static evaluation functions or heuristic guidelines. Although somewhat effective, these methods do not adequately reflect the complex relationships between the pieces on the board and their possible interactions.

This paper introduces an innovative approach to identifying strategic chess piece exchanges using bipartite graphs. Bipartite graphs are a special class of graphs that can effectively model relationships between two disjoint sets of entities. In the context of chess, these sets can represent the pieces of each player, with the edges between nodes signifying possible exchanges and their associated values. By employing this representation, it becomes possible to analyze and quantify the trade-offs in a systematic and structured manner.

The objective of this paper is to explore how bipartite graphs can be applied to chess, focusing on identifying optimal exchanges that maximize strategic advantage. To achieve this, we develop an algorithm that builds a bipartite graph from a given chess position, assigns weights to edges based on the value and strategic impact of exchanges, and computes the most beneficial trade-offs. This study aims to illustrate the utility of bipartite graphs as a decision-making instrument in chess and as an improvement to current approaches for assessing piece swaps. This method offers a unique perspective for evaluating chess positions and establishes a foundation for additional applications of graph theory in artificial intelligence and game theory.

II. Theoritical Foundation

A. Bipartite Graphs

A bipartite graph is a specialized type of graph that is structured into two distinct sets of vertices, U and W, such that every edge in the graph connects a vertex in U to a vertex in W. This ensures that no edges exist within the same set of vertices, making bipartite graphs particularly useful for modeling relationships between two distinct groups. The vertex sets U and W are disjoint

$$(U \cap W = \emptyset)$$

and the union of these sets forms the vertex set of the graph,

$$V = U \cup W$$

Mathematically, a graph $G=(V, E)$ is bipartite if and only if it satisfies the condition that no odd-length cycles exist in the graph. For every edge $e \in E$:

$$e = (u,w), \text{ where } u \in U \text{ and } w \in W$$

This property ensures that all vertices can be divided into two independent sets.

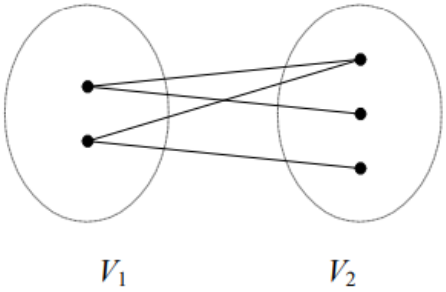


Fig 2.1 Bipartite Graphs

B. Maximum Matching in Bipartite Graphs

One of the fundamental problems in bipartite graph theory is the concept of maximum matching. A matching in a bipartite graph is a set of edges such that no two edges share a vertex. In other words, a matching pairs elements from one vertex set (e.g., U) to another (e.g., W) without overlap. A maximum matching is a matching that contains the largest possible number of edges, maximizing the number of pairings between the two sets. This problem has significant implications in various optimization tasks, such as resource allocation, task scheduling, and pairing problems.

In the context of chess, maximum matching can be used to optimize piece exchanges between white and black pieces. Each vertex in U represents a white piece, and each vertex in W represents a black piece. An edge $e \in E$ exists between a vertex $u \in U$ and a vertex $w \in W$ if a potential capture is possible, with weights assigned to edges based on the value and strategic impact of the exchange.

The maximum matching in this graph represents the set of optimal exchanges that can be performed. These exchanges are "optimal" in the sense that they maximize the material gain or minimize the material loss for one side, depending on the perspective of the analysis. For example:

- If a white rook (value = 5) can capture a black bishop (value = 3) and a black pawn (value = 1) can counter-capture the white rook, the net material balance of the exchange needs to be carefully evaluated. The maximum matching algorithm identifies whether this trade-off is strategically beneficial.

In chess, not all exchanges are of equal value. A key extension of maximum matching is weighted maximum matching, where each edge in the graph is assigned a weight reflecting the "importance" or "value" of the corresponding exchange. In this case, the goal is to find a matching MMM that maximizes the sum of the weights of the edges in MMM, denoted as:

$$\text{Maximize} \sum_{e \in M} w(e),$$

The weights in a chess bipartite graph can be determined using the following criteria:

1. Material Difference: The difference in value between the capturing and captured pieces (e.g., capturing a queen with a knight has a higher weight than capturing a pawn with a bishop).
2. Positional Factors: Strategic aspects, such as whether the captured piece is undefended or whether the exchange opens up key squares or files.
3. Future Implications: The long-term impact of the exchange, such as weakening pawn structures or exposing the opponent's king.

III. IMPLEMENTATION

The implementation of our bipartite graph approach to analyzing chess exchanges begins with a careful selection of a middlegame position that exemplifies various tactical and strategic exchange opportunities.

A. Approach and Tools

This study introduces the application of bipartite graphs to identify strategic chess piece exchanges during the middlegame phase. In this approach, each chess piece, whether belonging to the white or black side, is represented as a vertex in two disjoint sets. Potential interactions, where one piece can capture another, are represented as edges connecting these vertices. The weights of these edges are calculated based on the material value differences between the pieces involved, considering whether the target piece is defended or undefended. This mathematical framework provides a systematic method to analyze and evaluate exchanges.

To implement this method, several Python libraries and external tools were utilized. The chess library facilitated the representation of the chessboard and its pieces, parsing Portable Game Notation (PGN) files to reconstruct game positions, and identifying legal moves. The networkx library enabled the creation and manipulation of the bipartite graph, simplifying the process of assigning weights to edges, determining connections, and performing graph-related computations. For visualization, the matplotlib library was employed to display the constructed bipartite graph and to illustrate the results. Lastly, the Stockfish Chess Engine was used to evaluate positions after specific exchanges, as it is renowned for its high accuracy in analyzing chess positions.

The implementation of this approach follows several key steps:

1. PGN Parsing
   The chess game is first parsed from a PGN file, which contains the moves, players, and results. Using the chess.pgn module, the program reads the file, extracts the moves, and sequentially applies them to a chessboard object, reconstructing the final board position. This step provides the foundation for identifying active pieces and their possible interactions.
2. Identifying Pieces and Their Values
   After reconstructing the final board position, the program analyzes each square on the chessboard to determine the active pieces. These pieces are categorized by color (white or black), and their standard material values are assigned based on their type: pawn (1), knight (3), bishop (3), rook (5), queen (9), and king (0). This classification forms the basis for building the bipartite graph.
3. Constructing the Bipartite Graph
   Using the networkx library, a bipartite graph is created

where the vertices represent the pieces of both players. White pieces form one set of vertices, while black pieces form the other. For each white piece, all legal captures of black pieces are evaluated using the chess library. If a move is deemed legal, an edge is established between the corresponding vertices. The weight of each edge is calculated based on specific conditions:
    - If the target piece is defended, the weight equals the value of the captured piece minus the capturing piece's value.
    - If the target piece is undefended, the weight is simply the value of the captured piece.
4. Graph Visualization
   Once the bipartite graph is constructed, it is visualized using matplotlib. Each vertex is labeled with the corresponding piece and its position on the board. Edges are drawn with widths proportional to their weights, providing an intuitive representation of the significance of each potential exchange. This visualization aids in understanding the trade-offs and identifying the most advantageous exchanges.

B. Steps in Building Bipartite Graph

Number 1. Parsing PGN Files A chess game is commonly stored in the Portable Game Notation (PGN) format, which contains details about the players, moves, and results. The program begins by importing and parsing the PGN file to reconstruct the final board position:

- Input: The user provides the file path to a PGN file.
- Parsing: Using the chess.pgn module, the program reads the file and extracts the moves played.
- Reconstruction: Each move is applied sequentially to a chessboard object, resulting in the final board position.

2. Identifying Pieces and Their Values The final board position is analyzed to determine the active pieces:

- Each square on the chessboard is checked for the presence of a piece.
- Pieces are categorized by color (white or black), and their values are assigned based on the following standard:
    - Pawn = 1
    - Knight = 3
    - Bishop = 3
    - Rook = 5
    - Queen = 9
    - King = 0

```
White Pieces: {'a1': 5, 'c1': 3, 'f1': 5, 'g1': 0, 'a2': 1, 'b2'
 1, 'c2': 1, 'f2': 1, 'g2': 1, 'h2': 1, 'c3': 3, 'd3': 1, 'f3': 
, 'c4': 3, 'e4': 1, 'e5': 3}

Black Pieces: {'g4': 3, 'c5': 1, 'd5': 1, 'c6': 3, 'e6': 1, 'f6'
 3, 'a7': 1, 'b7': 1, 'f7': 1, 'g7': 1, 'h7': 1, 'a8': 5, 'c8': 
, 'd8': 9, 'e8': 0, 'h8': 5}
```

Fig 3.1

3. Constructing the Bipartite Graph The bipartite graph is constructed using the networkx library:

- Vertex Creation:
    - White pieces are assigned to one set of vertices.
    - Black pieces are assigned to the other set.
- Edge Creation:
    - For each white piece, all potential captures of black pieces are evaluated using the chess library to verify legality.
    - If a move is legal, an edge is created between the corresponding vertices in the graph.
    - The edge weight is calculated based on the conditions:
        - If the target piece is defended, the weight is the value of the captured piece minus the capturing piece.
        - If the target piece is undefended, the weight is equal to the value of the captured piece.

4. Graph Visualization Once the graph is constructed, it is visualized using matplotlib:

- Each vertex is labeled with the corresponding piece and position.
- Edges are drawn with widths proportional to their weights, representing the significance of each exchange.

C. Implementation to Code

To analyze potential piece exchanges in a chess game, the initial step involves parsing a Portable Game Notation (PGN) file, which contains all moves played during the game. This process reconstructs the final board position after all moves are sequentially applied.

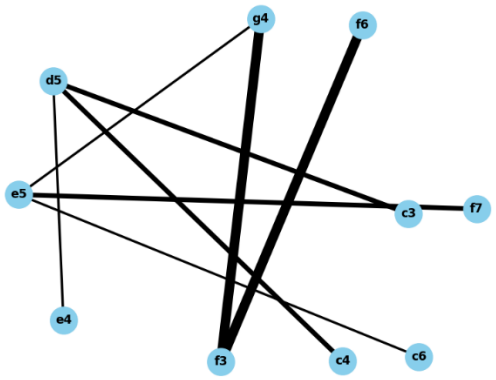
```python
file_path = input("Masukkan file .pgn: ")

game = load_pgn(file_path)
if game is None:
    return

print("Membaca papan...")
time.sleep(0.8)

board = game.board()

print(f"Posisi papan setelah langkah-langkah PGN:")
for move in game.mainline_moves():
    board.push(move)

print(board)

white_pieces = {}
black_pieces = {}

for square in chess.SQUARES:
    piece = board.piece_at(square)
    if piece:
        piece_value = piece_values[piece.piece_type]
        if piece.color == chess.WHITE:
            white_pieces[square] = piece_value
        else:
            black_pieces[square] = piece_value

print("\nWhite Pieces:", {chess.square_name(k): v for k, v
in white_pieces.items()})
print("\nBlack Pieces:", {chess.square_name(k): v for k, v
in black_pieces.items()})
```

The reconstructed position forms the basis for identifying active pieces and their interactions, as illustrated in the diagram below.

```
Masukkan file .pgn: tes3.pgn
Membaca papan...
Posisi papan setelah langkah-langkah PGN:
r . b q k . . r
p p . . . p p p
. . n . p n . .
. . p p N . . .
. . B . P . b .
. . N P . Q . .
P P P . . P P P
R . B . . R K .
```

The above figure displays the final chessboard position after parsing the PGN file and applying all moves. Each piece's type and position are explicitly shown, providing a clear starting point for building the bipartite graph. This visualization ensures that the reconstructed position accurately reflects the state of the game, facilitating further analysis of possible exchanges.

Once the final chessboard position has been established, a bipartite graph is constructed to model potential exchanges between pieces. White pieces are represented as one set of vertices, while black pieces form the other set. Edges between vertices signify legal captures, with weights indicating the value of the exchanges. The following figure illustrates this representation.



The figure above demonstrates how a bipartite graph effectively models chess exchanges. Each vertex corresponds to a chess piece, labeled with its type and position. Edges are drawn between vertices representing legal captures, and their weights reflect the calculated value of each potential exchange. This graph allows for systematic identification of advantageous trades based on both material values and positional factors.

```python
exchange_pairs = []
for white_square, white_value in white_pieces.items():
    for black_square, black_value in black_pieces.items():
        move = chess.Move(white_square, black_square)

        if move in board.legal_moves:
            if is_protected(board, black_square, chess.BLACK):
                value_difference = black_value - white_value
            else:
                value_difference = black_value

            exchange_pairs.append(
                (chess.square_name(white_square), chess.square_name(black_square), value_difference)
            )

print("\nPasangan potensial untuk pertukaran:")
for pair in exchange_pairs:
    print(f"White {pair[0]} vs Black {pair[1]} -> Selisih nilai: {pair[2]}")

G = build_bipartite_graph_from_exchange_pairs(exchange_pairs)

pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=500, font_size=10, font_weight='bold')

edge_weights = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edges(G, pos, edgelist=edge_weights.keys(),width=[2 + abs(weight) for weight in edge_weights.values()])

plt.title("Bipartite Graph for Chess Piece Exchange")
plt.show()

if exchange_pairs:
    maxi = max(exchange_pairs, key=lambda pair: pair[2])

    print("\nExchange Pair dengan Value Difference Tertinggi:")
    print(f"White {maxi[0]} vs Black {maxi[1]} -> Selisih nilai: {maxi[2]}")
else:
    print("\nTidak ada pasangan pertukaran yang ditemukan.")
```

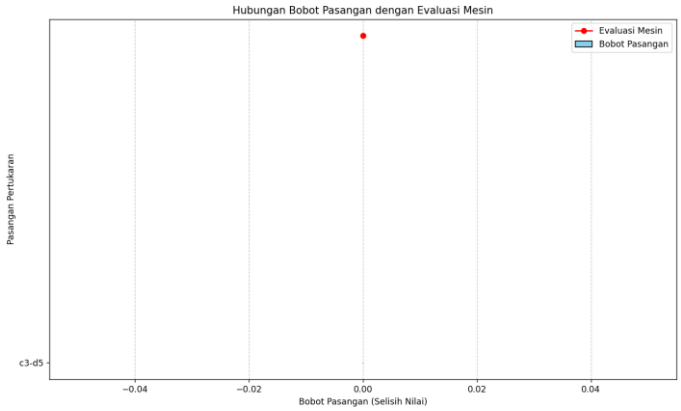
A critical aspect of this implementation is the assignment of weights to edges in the bipartite graph. Edge weights depend on whether the target piece is defended or undefended and are calculated based on the difference in material values. The following figure highlights this weight assignment process.

```
Pasangan potensial untuk pertukaran:
White c3 vs Black d5 -> Selisih nilai: -2
White f3 vs Black g4 -> Selisih nilai: -6
White f3 vs Black f6 -> Selisih nilai: -6
White c4 vs Black d5 -> Selisih nilai: -2
White e4 vs Black d5 -> Selisih nilai: 0
White e5 vs Black g4 -> Selisih nilai: 0
White e5 vs Black c6 -> Selisih nilai: 0
White e5 vs Black f7 -> Selisih nilai: -2

Exchange Pair dengan Value Difference Tertinggi:
White e4 vs Black d5 -> Selisih nilai: 0

Evaluasi mesin catur:
Pasangan: White e4 vs Black d5
Evaluasi mesin: 518
```

In the figure above, edges between vertices are labeled with their calculated weights. For instance, if a defended queen is captured by a rook, the weight reflects the material difference reduced by the risk involved. In contrast, an undefended target's weight equals its full value. This detailed representation underscores the strategic significance of each potential exchange.

D. Result

The analysis aims to compare the calculated pair weights derived from the bipartite graph model with the evaluation scores provided by the chess engine. The pair weights are computed based on the value difference and positional importance of exchanges, while the engine's evaluation reflects the strategic impact of the exchange on the overall game position. The figure below demonstrates the relationship between these two parameters for selected piece exchange pairs.



The figure above illustrates a comparison between the computed pair weights (represented in blue bars) and the engine's evaluation scores (represented as red points). For example, the exchange at position c3-d5 shows a minor positive correlation, indicating that both metrics align in identifying the exchange as slightly advantageous. This result validates the effectiveness of the bipartite graph model in approximating the strategic value of exchanges, although discrepancies in more complex positions may require further refinement.

IV. CONCLUSION

This study demonstrates the utility of bipartite graphs in analyzing strategic chess piece exchanges during the middlegame. By modeling the potential interactions between white and black pieces as a bipartite graph, and assigning edge weights based on material values and positional considerations, the proposed method provides a structured approach to evaluating exchanges. The integration of tools such as the chess library, networkx, and the Stockfish chess engine further enhances the accuracy and applicability of this approach.

The results indicate that the computed pair weights derived from the bipartite graph model show reasonable alignment with the evaluations provided by the chess engine. This suggests that the proposed method effectively approximates the strategic importance of exchanges, making it a valuable tool for decision-making in chess analysis.

However, certain discrepancies between the model's weights and the engine's evaluations highlight the need for further refinement. Future work could explore additional factors, such

as dynamic positional considerations and long-term strategic implications, to improve the accuracy of the model. The findings of this study also open avenues for applying graph theory in broader domains, such as artificial intelligence and game theory, particularly in problems involving complex interactions between entities.

In conclusion, the bipartite graph approach offers a novel and systematic perspective for evaluating chess exchanges, contributing to the intersection of mathematics, computer science, and strategic gameplay.

V. Suggestions

This paper presents a promising approach to analyzing chess piece exchanges using bipartite graphs, providing a mathematical framework that enhances strategic decision-making. However, several limitations and areas for improvement remain that warrant further exploration. First, the model primarily focuses on material values and basic positional factors, which, while effective, do not fully capture the dynamic complexity of chess positions, such as king safety, control of critical squares, or the psychological factors that influence human gameplay. Additionally, the current implementation is limited to analyzing individual exchanges rather than multi-move sequences, which are often crucial in determining long-term strategic advantages. The paper also lacks real-time applicability, as the algorithms are computationally intensive and better suited for post-game analysis rather than live scenarios. Visualization methods, while functional, could be made more intuitive and engaging to appeal to a broader audience, such as by integrating interactive board representations. Furthermore, the study would benefit from feedback or validation from expert players to ensure that the results align with practical chess understanding. Lastly, while the paper demonstrates the potential of bipartite graphs in chess, it does not fully explore the broader applications of this method in other domains, such as game theory, artificial intelligence, or resource allocation problems. Addressing these limitations and suggestions in future work could make this approach not only more comprehensive for chess analysis but also a versatile tool for solving complex decision-making problems across various fields.

VI. Acknowledgment

All praise to Allah Subhanahu wa Ta'ala, for His blessing, which have guided me to complete this paper succesfully. I would also like to thank to lecturer for IF1220 Linear and Geometric Algebra, Ir. Rila Mandala, M.Sc., Ph.D. for the motivation, knowledge, and guidance throughout the course. I hope that this paper will provide valuable insight and benefits to its readers.

References

[1] Munir, R. (2024). *Graf Bagian 1*. Informatika STEI ITB. Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf. [Accessed: Jan. 8, 2025]

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 January 2025

Muhammad Edo Raduputu Aprima - 13523096