Optimizing Routes and Resources in Board Game Ticket to Ride: Application of Graph and Combinatorics

Muhammad Kinan Arkansyaddad - 13523152[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]mkinanarkansyaddad@gmail.com, 13523152@std.stei.itb.ac.id

***Abstract*— Optimizing routes and resources in the board game Ticket to Ride involves the application of graph theory and combinatorics. The game board is modeled as a graph, where cities represent nodes and routes correspond to edges, with attributes such as length and color capturing the gameplay mechanics. Combinatorics is used to evaluate probabilities of acquiring specific cards, enabling strategic decision-making under resource constraints. Python-based implementations simulate optimal strategies for completing destination tickets while balancing efficiency and feasibility. Analysis through study cases demonstrates the effectiveness of graph-based modeling and probabilistic evaluation in optimizing route planning, resource management, and decision-making under uncertainty, offering insights into strategic optimization in gaming and real-world applications.**

***Keywords*—Graph, Optimization, Resource Management, Ticket to Ride.**

I. Introduction

Ticket to Ride is a strategic board game where players compete to build the longest and most railway across the map. The game is played on a map featuring cities connected by railway paths and players collect and use train cards to claim these routes corresponding to the cards' color. The main goal of the game is to score the highest points out of all players. Points can be gained by claiming railway routes between cities, completing destination tickets, and bonus from longest continuous path. However, points also can be lost when players cannot complete the destination tickets. The game concludes when a player's train pieces are all used and the player with the highest points wins.

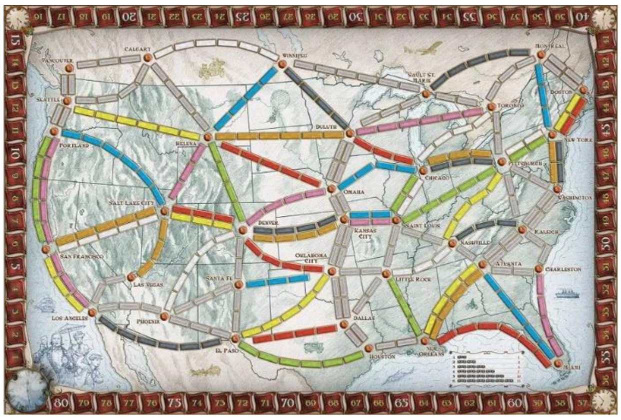


Fig 1.1 Ticket to Ride board map
Source: https://static.wikia.nocookie.net/ticket-to-ride/images/e/e9/TTR.jpg/revision/latest?cb=20160918212901

The mechanic of Ticket to Ride consists of drawing train cards, drawing destination tickets, claiming routes between cities and completing destination tickets. Players have to make decisions about which cities to connect, which destination tickets to prioritize, and when to draw train cards or new destination tickets. This decision-making process reflects real-world resource allocation and route optimization problems. Players must carefully balance their resources such as train cards to maximize their points while also blocking or competing with other players.

This paper explores the optimization of routes and resources in the game Ticket to Ride, applying concepts from graph and combinatorics theory to analyze and improve the strategic elements of the game. The cities and railways can be represented as graph, with cities as nodes and railways as edges, making graph theory is the ideal framework to model the game. The optimization of routes focuses on efficiently connecting cities while minimizing resources use. On the other hand, combinatorics helps players evaluate many possible combinations of routes and the chances of getting the resources that players need, assisting in the decision-making process for optimal gameplay. This paper aims to demonstrate how these mathematical concepts can enhance the understanding of strategy and resource management in Ticket to Ride.

II. Theoretical Basis

A. Graph

Graph is a data structure consisting of nodes (vertices) and edges that connect the nodes. Graphs are used to represent discrete objects and the relationship between them. Graph is formally defined as $G = (V, E)$, where G is graph, V is set of vertices that are not empty, and E is set of edges.

Graphs can be distinguished into two types based on the presence of loops or multiple edges:

1. Simple Graph: A graph that does not have multiple edges or loops.
2. Non-simple Graph: A graph that has multiple edges or loops. It can be further divided into:
    a. Multigraph: A graph containing multiple edges.
    b. Pseudograph: A graph containing loops.

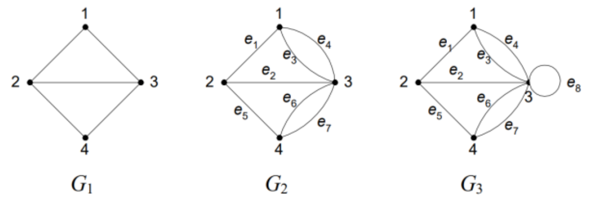


Fig 2.1 G1 is a simple graph, G2 is a multigraph, G3 is a pseudograph
Source: [1]

Furthermore, based on their orientation, graphs can also be classified into two types:

1. Undirected Graph: A graph that does not have directional orientation.
2. Directed Graph: A graph where each edge connecting nodes has direction.

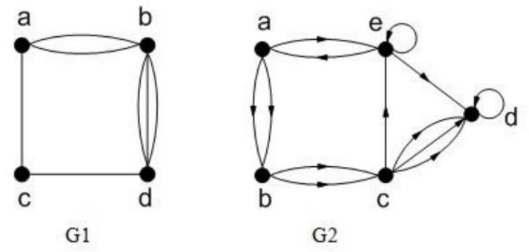


Fig 2.2 G1 is an undirected graph, G2 is a directed graph
Source: [1]

Some terminologies in graph theory:

1. Adjacency: Two nodes in a graph are said to be adjacent if they are connected by at least one edge.
2. Incidency: An edge in a graph is said to be incident to nodes a and b if it connects them.
3. Degree: The degree of a node is the number of edges incident to that node.
4. Path: A path is a sequence of nodes and edges that connects the starting node to the destination node.
5. Cycle: A cycle is a path that starts and ends at the same node.
6. Connected: Two nodes are said to be connected if there is a path connecting them.
7. Weighted Graph: A weighted graph is a graph that has specific values or weights on each edge connecting the nodes.

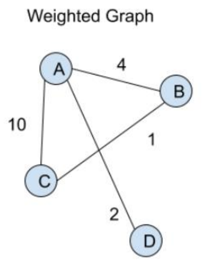


Fig 2.3 Weighted graph example
Source: [1]

B. Combinatorics

Combinatorics is a branch of mathematics that focuses on the study of counting, arrangement, and combination of objects. It provides foundational principles and techniques necessary for solving problems related to discrete structures. Combinatorics is essential in various fields, including computer science, statistics, and optimization, as it helps analyze the possible configurations of a set of items. The primary focus of combinatorics is to determine how many ways a certain arrangement or selection can be made, often without the need to enumerate all possible outcomes explicitly.

A permutation refers to an arrangement of objects in a specific order. This concept is crucial in combinatorics, as it allows us to count the number of ways to arrange a set of distinct items. The total number of permutations of *n* distinct objects is given by *n!*, which is the product of all positive integers up to *n*. For example, if we have three distinct objects, say A, B, and C, the permutations can be listed as follows: ABC, ACB, BAC, BCA, CAB, and CBA. Thus, the number of permutations of these three objects is 3! = 6.

When determining the number of ways to arrange *r* elements selected from a total of *n* distinct elements, the formula for permutations of *r* from *n*, denoted as *P(n,r)*, is used. The formula is given by:

$$P(n,r) = \frac{n!}{(n-r)!}$$

This formula calculates the number of ways to choose *r* elements from *n* and arrange them in order. For instance, if there are 5 distinct books and the goal is to find how many ways 3 of them can be arranged on a shelf, the calculation would be as follows:

$$P(5,3) = \frac{5!}{(5-3)!} = \frac{5!}{2!} = 60.$$

In contrast, a combination refers to a selection of items from a larger set where the order of selection does not matter. This is in contrast to permutations, where the arrangement is significant. The number of ways to choose *r* elements from a set of *n* elements is denoted as *C(n,r)* or $\binom{n}{r}$, and is calculated using the following formula:

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

This formula accounts for the fact that the order of selection is irrelevant by dividing the total permutations by the number of ways to arrange the selected items. For example, if the task is to select 2 fruits from a basket of 5 different fruits, the number of combinations can be calculated as:

$$C(5,2) = \frac{5!}{2!\,(5-2)!} = \frac{5!}{2!\,3!} = 10.$$

C. Hypergeometric Probability Distribution

The hypergeometric probability distribution is a discrete probability distribution that describes the likelihood of drawing a specific number of successes in a sequence of draws from a finite population without replacement. This distribution is particularly useful in scenarios where the population can be divided into two distinct categories, such as success and failure, and where the draws are made without replacement, meaning that the composition of the population changes with each draw. The probability mass function (PMF) of the hypergeometric distribution is given by:

$$P(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

In this formula, *P(X=k)* represents the probability of obtaining exactly *k* successes, while N is the total number of items in the population, K is the total number of successes in the population, n is the number of draws, and k is the number of observed successes.

III. Implementation

A. Limitations

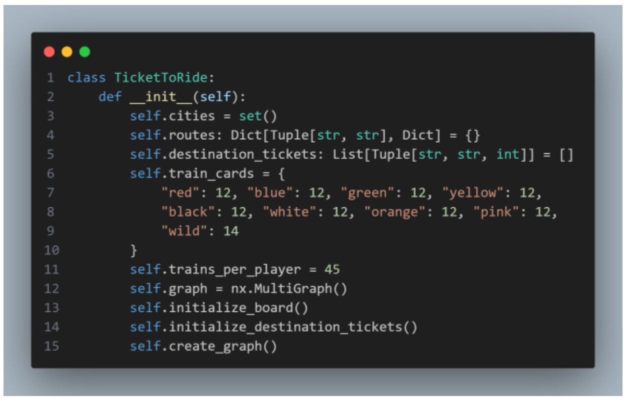
The proposed implementation for optimizing routes and resources in the Ticket to Ride board game effectively models the game mechanic and provides insightful information. However, it operates under certain limitations. First and foremost, the optimization framework assumes a single player environment, disregarding another player's actions and strategies. In a multiplayer game, opponents can claim critical routes and/or block routes which result in altering the availability of optimal routes. The model does not account for this dynamic element which is important to the real-world gameplay experience.

Additionally, the implementation assumes that players will always have the option to draw the exact cards they require over several turns. While the hypergeometric probability distribution provides a statistical estimation of card-drawing probabilities, it simplifies the complex interplay of chance and strategic card selection in the actual game. Furthermore, the framework does not consider the potential benefits of diversifying route options or prioritizing flexible gray routes to mitigate the risks of unforeseen interruptions.

Lastly, the system assumes the availability of sufficient trains to complete all planned routes without considering strategic decisions regarding resource allocation. Players may often need to adapt their plans to optimize points with limited trains. Similarly, the optimization ignores scenarios where completing smaller, high-impact routes may provide a more favorable outcome than pursuing lengthy destination tickets with uncertain probabilities of success.

B. Graph Modeling from Ticket to Ride

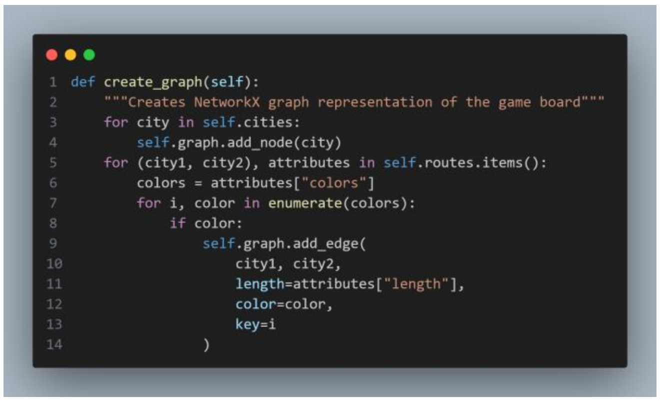
The implementation begins with modeling the Ticket to Ride board game as a graph structure using NetworkX, a Python library for complex network analysis. In the code, each city is represented as a node in the graph, while the train routes between cities are represented as edges. The graph is implemented as a MultiGraph to accommodate parallel routes between the same cities, which is a common feature in the game. The basic structure is initialized in the TicketToRide class:



```python
class TicketToRide:
    def __init__(self):
        self.cities = set()
        self.routes: Dict[Tuple[str, str], Dict] = {}
        self.destination_tickets: List[Tuple[str, str, int]] = []
        self.train_cards = {
            "red": 12, "blue": 12, "green": 12, "yellow": 12,
            "black": 12, "white": 12, "orange": 12, "pink": 12,
            "wild": 14
        }
        self.trains_per_player = 45
        self.graph = nx.MultiGraph()
        self.initialize_board()
        self.initialize_destination_tickets()
        self.create_graph()
```

Fig 3.1 TicketToRide class initialization

The graph construction is handled through the create_graph method, which iterates through the predefined routes and adds them to the NetworkX graph structure. Each edge in the graph contains important attributes such as the route length and color requirements. For parallel routes, a unique key is assigned to distinguish between multiple edges connecting the same pair of cities:

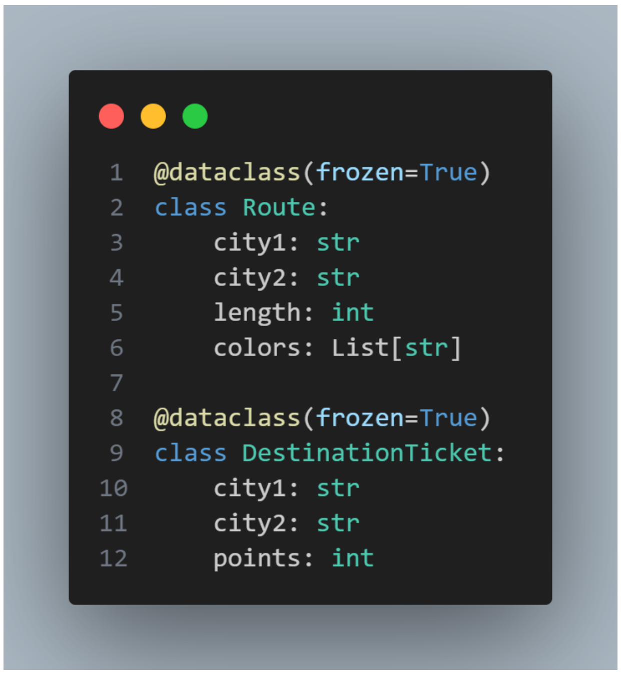


```python
def create_graph(self):
    """Creates NetworkX graph representation of the game board"""
    for city in self.cities:
        self.graph.add_node(city)
    for (city1, city2), attributes in self.routes.items():
        colors = attributes["colors"]
        for i, color in enumerate(colors):
            if color:
                self.graph.add_edge(
                    city1, city2,
                    length=attributes["length"],
                    color=color,
                    key=i
                )
```

Fig 3.2 create_graph function

C. Game Modelling in Python

The game implementation models several key components of Ticket to Ride using Python's data structures and object-oriented programming principles. At the core of the implementation is the TicketToRide class, which encapsulates all game mechanics and state management. The game state is primarily modeled through dataclasses and dictionaries that represent the essential game elements:

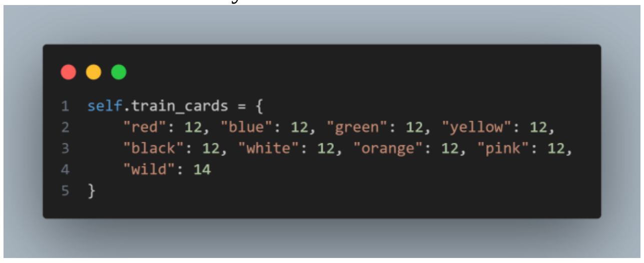


```python
@dataclass(frozen=True)
class Route:
    city1: str
    city2: str
    length: int
    colors: List[str]

@dataclass(frozen=True)
class DestinationTicket:
    city1: str
    city2: str
    points: int
```

Fig 3.3 Route and destination ticket data structures

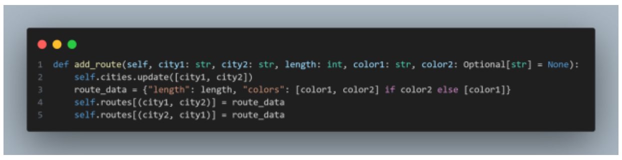
The train cards, which represent the game's primary resource, are modeled using a dictionary that tracks the quantity of each color card remaining in the deck. This implementation accounts for all card colors available in the game, including the wild cards that can substitute for any color:



```python
self.train_cards = {
    "red": 12, "blue": 12, "green": 12, "yellow": 12,
    "black": 12, "white": 12, "orange": 12, "pink": 12,
    "wild": 14
}
```

Fig 3.4 Train cards initialization

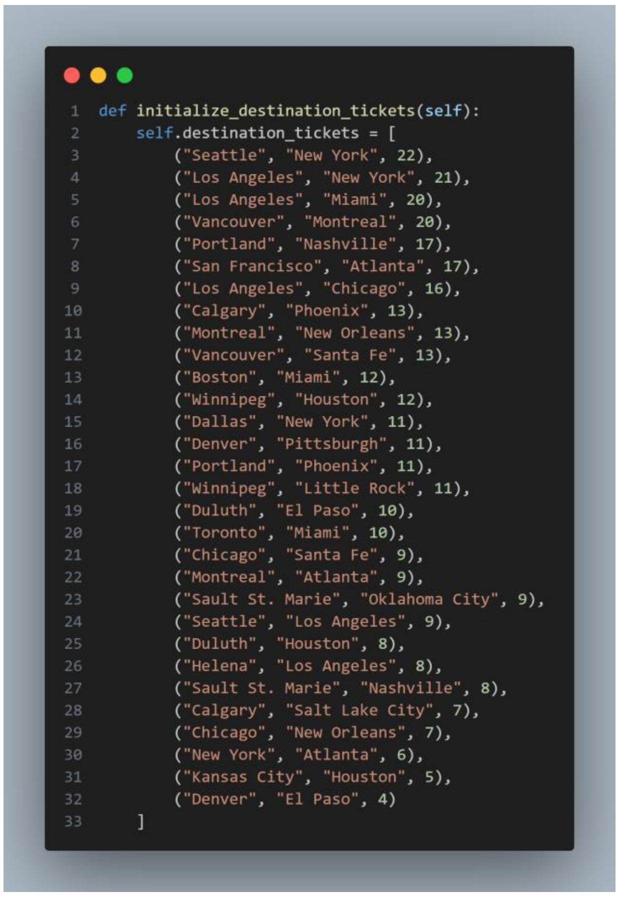
The game board's routes are initialized through the initialize_board method, which establishes all possible connections between cities. Each route is defined with its length and color requirements, including special cases for gray routes (which can be claimed using any single-color set) and parallel routes (where two separate connections exist between the same cities). The route data is stored in a bidirectional format, allowing for efficient lookup from either city:



```python
def add_route(self, city1: str, city2: str, length: int, color1: str, color2: Optional[str] = None):
    self.cities.update([city1, city2])
    route_data = {"length": length, "colors": [color1, color2] if color2 else [color1]}
    self.routes[(city1, city2)] = route_data
    self.routes[(city2, city1)] = route_data
```

Fig 3.5 add_route method

Destination tickets, which define the scoring objectives for players, are modeled as a collection of city pairs with associated point values. These tickets are stored with their respective point values and can be used to evaluate different route completion strategies. The implementation maintains this information in a structured format that facilitates route planning and scoring calculations:



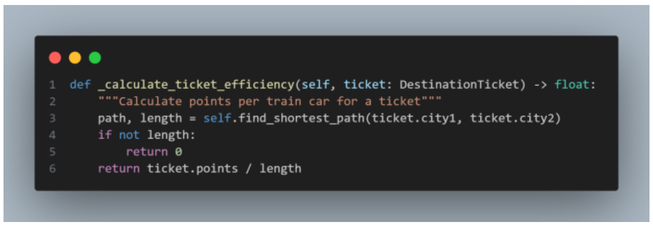
```python
def initialize_destination_tickets(self):
    self.destination_tickets = [
        ("Seattle", "New York", 22),
        ("Los Angeles", "New York", 21),
        ("Los Angeles", "Miami", 20),
        ("Vancouver", "Montreal", 20),
        ("Portland", "Nashville", 17),
        ("San Francisco", "Atlanta", 17),
        ("Los Angeles", "Chicago", 16),
        ("Calgary", "Phoenix", 13),
        ("Montreal", "New Orleans", 13),
        ("Vancouver", "Santa Fe", 13),
        ("Boston", "Miami", 12),
        ("Winnipeg", "Houston", 12),
        ("Dallas", "New York", 11),
        ("Denver", "Pittsburgh", 11),
        ("Portland", "Phoenix", 11),
        ("Winnipeg", "Little Rock", 11),
        ("Duluth", "El Paso", 10),
        ("Toronto", "Miami", 10),
        ("Chicago", "Santa Fe", 9),
        ("Montreal", "Atlanta", 9),
        ("Sault St. Marie", "Oklahoma City", 9),
        ("Seattle", "Los Angeles", 9),
        ("Duluth", "Houston", 8),
        ("Helena", "Los Angeles", 8),
        ("Sault St. Marie", "Nashville", 8),
        ("Calgary", "Salt Lake City", 7),
        ("Chicago", "New Orleans", 7),
        ("New York", "Atlanta", 6),
        ("Kansas City", "Houston", 5),
        ("Denver", "El Paso", 4)
    ]
```

Fig 3.6 initialize_destination_tickets method

The model incorporates game rules and constraints through various methods and attributes. For instance, the trains_per_player attribute enforces the game's limitation on the number of train pieces each player can use, while the route color requirements ensure that players must collect appropriate sets of cards to claim routes. This comprehensive modeling approach provides a foundation for implementing optimization strategies while maintaining the game's core mechanics and rules.

D. Implementation of Optimization Calculation

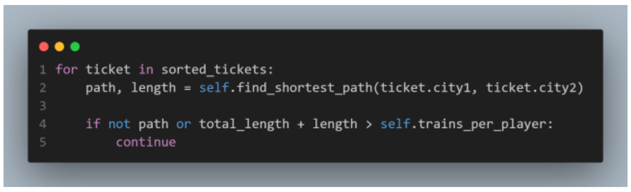
The optimization implementation focuses on maximizing point gains while managing the limited resources available to players. The core optimization logic is implemented in the optimize_ticket_completion method, which employs a greedy approach to select and complete destination tickets. The method first calculates a ticket efficiency metric (points per train car required) to prioritize tickets that offer the best point-to-resource ratio:



```python
def _calculate_ticket_efficiency(self, ticket: DestinationTicket) -> float:
    """Calculate points per train car for a ticket"""
    path, length = self.find_shortest_path(ticket.city1, ticket.city2)
    if not length:
        return 0
    return ticket.points / length
```

Fig 3.7 calculate_ticket_efficiency function

The optimization process considers several key constraints. First, the total number of train cars used cannot exceed the player's limit (45 trains). Second, the algorithm must account for the shared routes between different destination tickets to maximize efficiency. The implementation handles these constraints by maintaining a running total of used train cars and tracking which routes have been claimed:

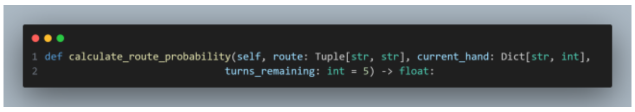


```python
for ticket in sorted_tickets:
    path, length = self.find_shortest_path(ticket.city1, ticket.city2)

    if not path or total_length + length > self.trains_per_player:
        continue
```

Fig 3.8 Snap code of the optimization process

Resource management is further enhanced by the calculate_route_probability method, which uses hypergeometric probability distribution to estimate the likelihood of collecting required train cards for specific routes. This method considers both the current hand composition and the remaining cards in the deck:



```python
def calculate_route_probability(self, route: Tuple[str, str], current_hand: Dict[str, int],
                                turns_remaining: int = 5) -> float:
```

Fig 3.9 calculate_route_probability function

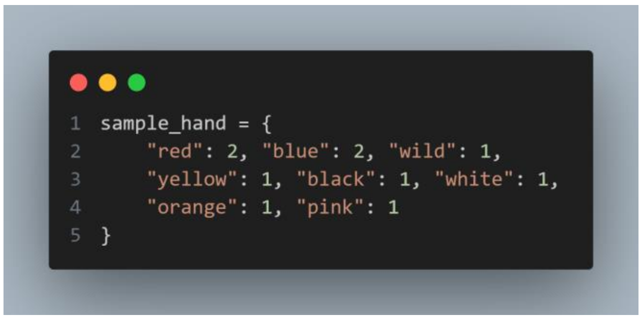
One notable limitation of the current implementation is that it does not consider opponent actions, which could potentially block critical routes in the actual game. Additionally, the optimization assumes perfect information about the destination tickets, whereas in a real game, players must make decisions with incomplete information about future ticket draws. The greedy approach used in route selection, while computationally efficient, may not always find the globally optimal solution, particularly in complex scenarios where multiple interdependent routes exist.

The implementation includes visualization capabilities through matplotlib, allowing for visual verification of the optimized routes. This is particularly useful for debugging and understanding the spatial relationships between selected routes. These visualizations help validate the optimization results and provide insights into the geographical distribution of selected routes, though they are not used in the actual optimization calculations.

IV. Experiments

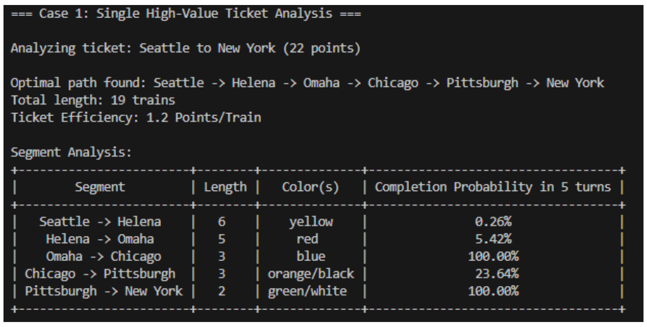
Optimizing strategies in Ticket to Ride requires a blend of mathematical precision and strategic foresight. By modeling the game's mechanics using graph theory, probability, and resource optimization techniques, it becomes possible to evaluate and implement efficient gameplay strategies. Through carefully constructed study cases, various aspects of route planning, resource allocation, and probabilistic decision-making are analyzed. Each case highlights unique challenges and insights, offering a detailed perspective on how theoretical models translate into practical outcomes.

1. Case 1: Single High-Value Destination Ticket
   This case examines the completion of a single high-value destination ticket, such as "Seattle to New York" (22 points). The model identifies the shortest path to minimize train usage. The selected path spans a geographically extensive route, requiring 19 trains to complete. The results indicate that the optimization model effectively handles straightforward objectives, ensuring efficient resource allocation and high ticket completion rates. This scenario demonstrates the model's ability to simplify complex decision-making while achieving optimal results.



```python
sample_hand = {
    "red": 2, "blue": 2, "wild": 1,
    "yellow": 1, "black": 1, "white": 1,
    "orange": 1, "pink": 1
}
```

Fig 5.1 Sample hand case 1



```
=== Case 1: Single High-Value Ticket Analysis ===

Analyzing ticket: Seattle to New York (22 points)

Optimal path found: Seattle -> Helena -> Omaha -> Chicago -> Pittsburgh -> New York
Total length: 19 trains
Ticket Efficiency: 1.2 Points/Train

Segment Analysis:
+-------------------------+--------+-------------+-----------------------------------+
|         Segment         | Length |   Color(s)  | Completion Probability in 5 turns |
+-------------------------+--------+-------------+-----------------------------------+
|   Seattle -> Helena     |   6    |   yellow    |               0.26%               |
|    Helena -> Omaha      |   5    |    red      |               5.42%               |
|    Omaha -> Chicago     |   3    |    blue     |              100.00%              |
| Chicago -> Pittsburgh   |   3    | orange/black |              23.64%              |
| Pittsburgh -> New York  |   2    | green/white |              100.00%              |
+-------------------------+--------+-------------+-----------------------------------+
```

Fig 5.2 Case 1 results

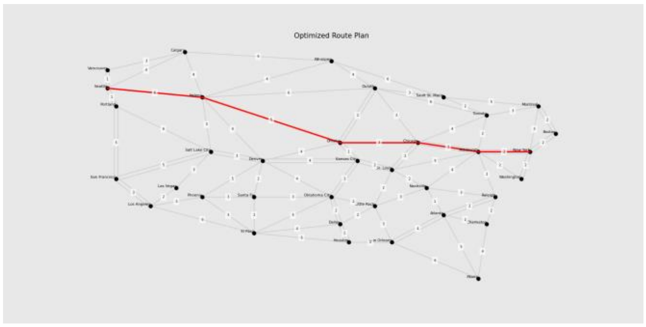
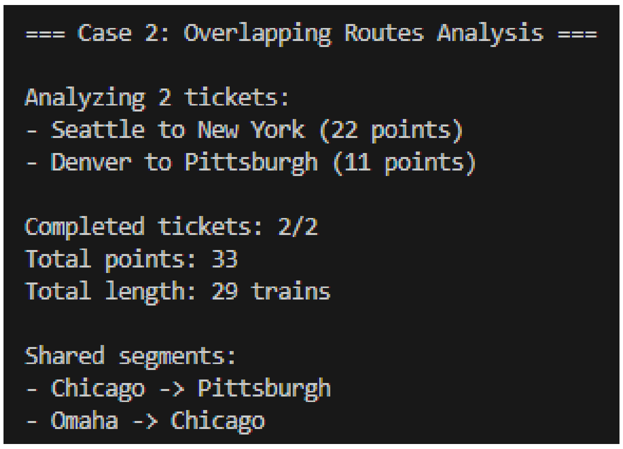


Fig 5.3 Visualization of optimal routes from Seattle to New York

2. Case 2: Completing Tickets with Overlapping Routes
   In this scenario, the model is tasked with optimizing the completion of two tickets: "Seattle to New York" (22 points) and "Denver to Pittsburgh" (11 points). These tickets share overlapping segments, such as the route between "Chicago and Pittsburgh." The optimization prioritizes overlapping routes to conserve resources,

reducing redundancy and improving overall efficiency. The final plan completes both tickets using 29 trains, achieving a combined score of 33 points with an efficiency of 1.14 points per train. This case highlights the model's ability to manage multiple objectives, emphasizing the value of detecting and leveraging route overlaps in complex scenarios.


```
=== Case 2: Overlapping Routes Analysis ===

Analyzing 2 tickets:
- Seattle to New York (22 points)
- Denver to Pittsburgh (11 points)

Completed tickets: 2/2
Total points: 33
Total length: 29 trains

Shared segments:
- Chicago -> Pittsburgh
- Omaha -> Chicago
```

Fig 5.4 Case 2 results

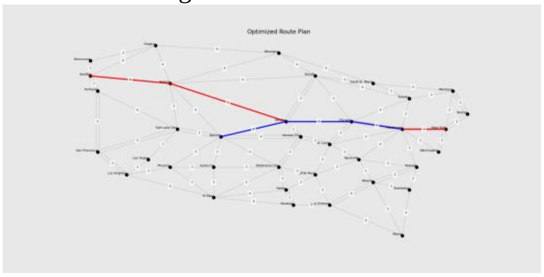


Fig 5.3 Visualization of optimal routes from Seattle to New York

3. Case 3: Probability Challenges in Gray Routes
This case focuses on evaluating the feasibility of completing a gray route, such as "Seattle to Calgary" (4 train), with a limited card hand. Gray routes allow flexibility in card colors but often present probabilistic challenges. For longer gray routes, the success probability diminishes, making strategic card draws or wild card reserves essential. The analysis highlights the importance of probabilistic modeling in decision-making, demonstrating how the model helps assess risks and prioritize routes based on success likelihood.

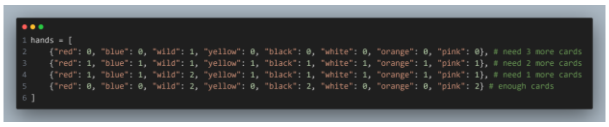
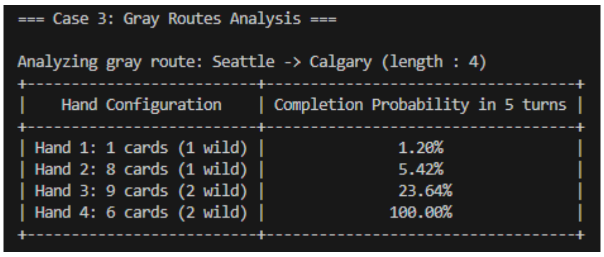

```
hands = [
    {"red": 0, "blue": 0, "wild": 1, "yellow": 0, "black": 0, "white": 0, "orange": 0, "pink": 0}, # need 3 more cards
    {"red": 1, "blue": 1, "wild": 1, "yellow": 1, "black": 1, "white": 1, "orange": 1, "pink": 1}, # need 2 more cards
    {"red": 1, "blue": 1, "wild": 2, "yellow": 1, "black": 1, "white": 1, "orange": 1, "pink": 1}, # need 1 more cards
    {"red": 0, "blue": 0, "wild": 2, "yellow": 0, "black": 2, "white": 0, "orange": 0, "pink": 2} # enough cards
]
```

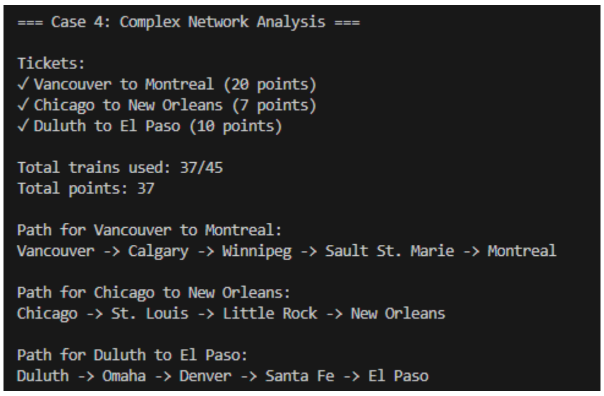
Fig 5.4 Case 3 hands configuration


```
=== Case 3: Gray Routes Analysis ===

Analyzing gray route: Seattle -> Calgary (length : 4)
+------------------------+-----------------------------------+
|   Hand Configuration   | Completion Probability in 5 turns |
+------------------------+-----------------------------------+
| Hand 1: 1 cards (1 wild) |              1.20%              |
| Hand 2: 8 cards (1 wild) |              5.42%              |
| Hand 3: 9 cards (2 wild) |             23.64%              |
| Hand 4: 6 cards (2 wild) |            100.00%              |
+------------------------+-----------------------------------+
```

Fig 5.5 Case 3 results

4. Case 4: Resource-Constrained Multi-Ticket Optimization
In this case, the model attempts to complete three tickets—"Vancouver to Montreal" (20 points), "Chicago to New Orleans" (7 points), and "Duluth to El Paso" (10 points)—under a resource constraint of 45 trains. The optimization prioritizes tickets based on their points-to-length ratio, ensuring that high-efficiency tickets are completed first. The final plan uses 37 trains to complete all three tickets, leaving 8 trains available for future connections. The optimized routes avoid unnecessary detours, maximizing the total score to 37 points with an efficiency of 1 point per train. This case illustrates the model's effectiveness in balancing multiple objectives within strict resource constraints.


```
=== Case 4: Complex Network Analysis ===

Tickets:
✓ Vancouver to Montreal (20 points)
✓ Chicago to New Orleans (7 points)
✓ Duluth to El Paso (10 points)

Total trains used: 37/45
Total points: 37

Path for Vancouver to Montreal:
Vancouver -> Calgary -> Winnipeg -> Sault St. Marie -> Montreal

Path for Chicago to New Orleans:
Chicago -> St. Louis -> Little Rock -> New Orleans

Path for Duluth to El Paso:
Duluth -> Omaha -> Denver -> Santa Fe -> El Paso
```

Fig 5.6 Case 4 results

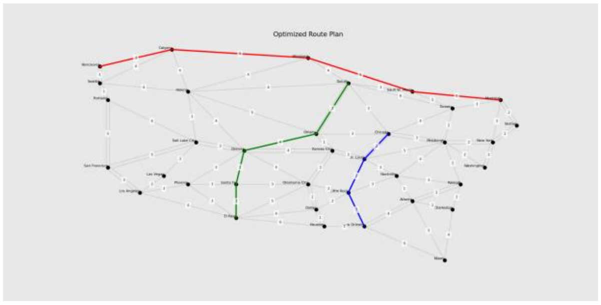
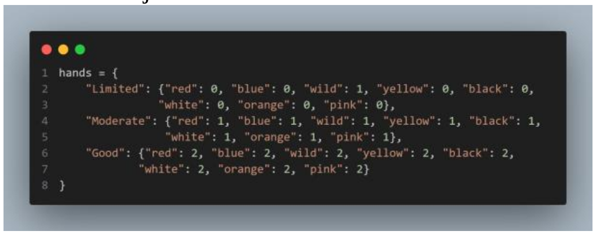


Fig 5.7 Visualization of optimal routes for case 4

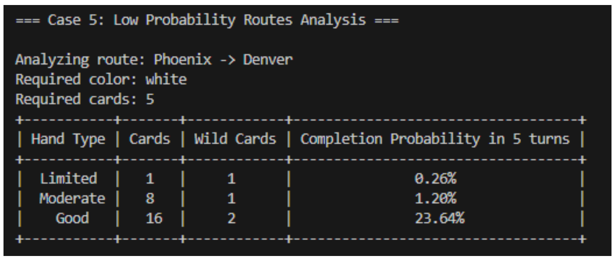
5. Case 5: Low Probability Routes Analysis
This scenario assesses the feasibility of completing a low-probability ticket, such as "Phoenix to Denver" (5 points), which requires five white cards. Due to the low likelihood of acquiring the required cards, the model deprioritizes this ticket in favor of higher-probability routes. This analysis demonstrates the importance of

integrating probabilistic modeling into decision-making, enabling the player to avoid risky strategies and focus on achievable objectives.



```
hands = {
    "Limited": {"red": 0, "blue": 0, "wild": 1, "yellow": 0, "black": 0,
                "white": 0, "orange": 0, "pink": 0},
    "Moderate": {"red": 1, "blue": 1, "wild": 1, "yellow": 1, "black": 1,
                 "white": 1, "orange": 1, "pink": 1},
    "Good": {"red": 2, "blue": 2, "wild": 2, "yellow": 2, "black": 2,
             "white": 2, "orange": 2, "pink": 2}
}
```

Fig 5.8 Case 5 hands configuration

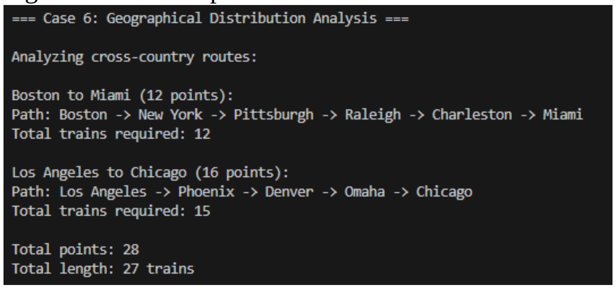


```
=== Case 5: Low Probability Routes Analysis ===

Analyzing route: Phoenix -> Denver
Required color: white
Required cards: 5
+-----------+-------+------------+--------------------------------+
| Hand Type | Cards | Wild Cards | Completion Probability in 5 turns |
+-----------+-------+------------+--------------------------------+
|  Limited  |   1   |     1      |             0.26%              |
|  Moderate |   8   |     1      |             1.20%              |
|    Good   |  16   |     2      |             23.64%             |
+-----------+-------+------------+--------------------------------+
```

Fig 5.9 Case 5 results

6. Case 6: Geographical Distribution and Complexity
This case evaluates the completion of geographically dispersed tickets, such as "Boston to Miami" (12 points) and "Los Angeles to Chicago" (16 points). The optimization balances resource allocation across regions, identifying critical nodes like Chicago as hubs for multiple connections. Visualization reveals efficient routing strategies that minimize detours while ensuring connectivity between distant regions. The final plan uses 27 trains to complete both tickets, achieving a score of 28 points with an efficiency of 1.04 points per train. This case highlights the value of geographical visualization in managing complex route networks and optimizing cross-regional ticket completion.



```
=== Case 6: Geographical Distribution Analysis ===

Analyzing cross-country routes:

Boston to Miami (12 points):
Path: Boston -> New York -> Pittsburgh -> Raleigh -> Charleston -> Miami
Total trains required: 12

Los Angeles to Chicago (16 points):
Path: Los Angeles -> Phoenix -> Denver -> Omaha -> Chicago
Total trains required: 15

Total points: 28
Total length: 27 trains
```

Fig 5.10 Case 6 results

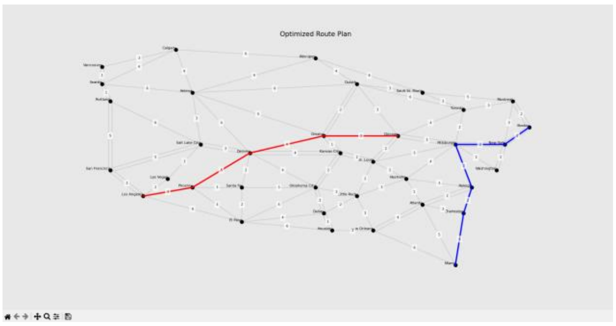


Fig 5.11 Visualization of optimal routes for case 6

V. Conclusion

The study successfully demonstrates how graph theory and combinatorics can be applied to optimize strategic decision-making in the board game Ticket to Ride. By modeling the game as a graph, the framework provides a clear representation of the board, enabling efficient computation of shortest paths and resource allocation. Combinatorics, particularly the use of hypergeometric probability distributions, enhances the evaluation of route feasibility under various constraints, helping players make informed decisions about ticket prioritization and resource usage.

Through diverse study cases, the model proves its versatility in handling single and multi-ticket scenarios, addressing both deterministic and probabilistic challenges. Key strengths include its ability to detect overlapping routes, prioritize high-efficiency tickets, and assess risk in low-probability routes. However, the model's assumptions, such as the absence of opponent interference and perfect information about resources, limit its applicability in real-world multiplayer scenarios.

Future extensions could incorporate adversarial dynamics, real-time decision-making, and machine learning to adapt strategies dynamically. Despite its limitations, the framework offers valuable insights into resource optimization and strategic planning, with potential applications in other domains requiring efficient allocation of limited resources.

VI. Appendix

Project source code: https://github.com/kin-ark/optimizing-ticket-to-ride

VII. Acknowledgment

The author expresses gratitude to all parties who have assisted in the making of this paper, especially to:

1. Allah Swt.
2. Both parents, for providing moral and material support.
3. Friends who have encouraged and aided in the completion of this paper.
4. Arrival Dwi Sentosa, S.Kom, M.T. as the lecturers for the IF1220 Discrete Mathematics course, for his invaluable guidance and support throughout the semester.
5. Dr. Ir. Rinaldi Munir, MT., for providing learning resources materials.

The author deeply appreciates all the assistance, encouragement, and kindness received from these individuals and groups, without which the completion of this paper would not have been possible.

References

[1] Munir, Rinaldi. 2024. “Graf (bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf (accessed on 8 January 2025).
[2] Munir, Rinaldi. 2024. “Kombinatorika (bagian 1)”. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf (accessed on 8 January 2025).

[3] Pollard, D. (2010). Symmetry [Polya Urn]. Yale University. https://www.stat.yale.edu/~pollard/Courses/600.spring2010/Handouts/Symmetry%5BPolyaUrn%5D.pdf (accessed on 8 January 2025)

STATEMENT

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, 5 January 2024

[illegible]

Muhammad Kinan Arkansyaddad
13523152