Algorithmic Pathfinding and Directed Graphs for Optimizing Offensive Passing Sequences in Basketball Strategy

Muhammad Adnan Kurniawan - 13525071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: adnankrniawn@gmail.com , 13525071@std.stei.itb.ac.id

Abstract*—Modern basketball heavily relies on strategic passing to dismantle elite defensive formations. This paper proposes a mathematical framework using algorithmic pathfinding and directed graphs to optimize offensive passing sequences. By modeling players as nodes and passes as directed edges with assigned success probabilities, we evaluate the efficiency of various offensive playbooks. In high-stakes environments such as the NBA Finals, where defensive schemes are highly adaptable and the margin for error is minimal, determining the optimal ball-movement sequence under the 24-second shot clock is a computationally intensive challenge. This study applies combinatorial principles to calculate possible passing permutations and utilizes tree traversal algorithms to identify the path with the highest probability of yielding a high-percentage shot. Experimental implementation via a Python simulation demonstrates the practical application of discrete mathematics in advanced sports analytics. The simulation results demonstrate that a direct one-pass route achieves a 50% success probability, statistically outperforming all multi-pass sequences, which degrade to as low as 27% when routing through congested interior passing lanes. Furthermore, when the model dynamically adapts to an elite interior defensive scheme, the optimal perimeter kick-out route achieves a maximum success probability of 85%, offering a strict, data-driven approach to offensive strategy design.

***Keywords*—directed graphs, combinatorics, pathfinding, basketball analytics, algorithmic optimization**

I. INTRODUCTION

The evolution of professional basketball has transformed the sport from a purely physical contest into a domain of rigorous data analytics [1], [5], [6]. In the contemporary era, particularly at the highest levels of competition such as the NBA Finals, offensive efficiency is heavily dictated by continuous ball movement and spatial manipulation. A critical component of this movement is the passing sequence, which aims to strategically shift elite defensive formations and create optimal scoring opportunities within a tightly constrained timeframe [2].

Mathematically, a basketball offensive possession can be modeled as a complex network system. During a standard 24-second shot clock, the offensive team must rapidly evaluate multiple passing lanes while anticipating the defensive rotation. Relying solely on a coach's predefined playbook or a player's real-time intuition is often insufficient against elite, highly adaptable defensive schemes. As the number of passes increases, the complexity of determining the safest and most lethal route for the ball grows exponentially, evolving into a computationally intensive challenge.

The core challenge for coaching staffs and data analysts lies in determining the most mathematically efficient path through this network. This paper explores the application of discrete mathematics to resolve this spatial optimization problem. By mapping the basketball court into a mathematical graph, assigning probability weights based on defensive pressure, and applying tree traversal algorithms, we can programmatically discover the optimal sequence of passes. This methodology seeks to find the specific offensive play that maximizes the likelihood of an open shot, proving that championship-level execution is fundamentally rooted in discrete mathematical principles [3].

The remainder of this paper is organized as follows: Section II discusses the fundamental mathematical concepts of directed graphs, combinatorial permutations, and algorithmic pathfinding. Section III details the proposed methodology for modeling the basketball network. Section IV presents the algorithmic implementation and the Python simulation setup used to validate the model. Finally, Section V concludes the analysis and discusses potential future work.

II. THEORY

A. Directed Graphs and Weighted Edges

A graph G = (V, E) consists of a nonempty set of vertices V and a set of edges E [1]. In its simplest form, vertices represent objects and edges represent relationships between them. Consider a simple undirected graph with three vertices v_1, v_2, and v_3, where edges connect v_1–v_2 and v_2–v_3. In this case, vertex v_2 is adjacent to both v_1 and v_3, while v_1 and v_3 are not adjacent to each other. The degree of a vertex, deg(v), is the number of edges incident with it; in this example, $\deg(v_2) = 2$ while $\deg(v_1) = \deg(v_3) = 1$.

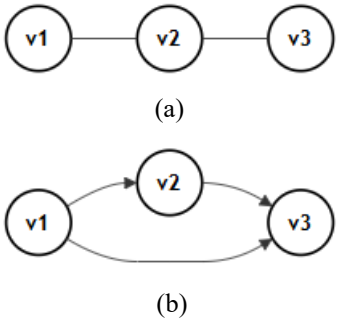

Fig. 1. (a) A simple undirected graph with three vertices. (b) A directed graph (digraph) where edges carry directional orientation represented by arrows. (Source: Author)

Because a pass moves the ball from one specific player to another, the graph modeling a basketball offense must use directed edges, forming a directed graph (digraph) [2]. In a digraph, each edge (u, v) has a defined origin vertex u and a destination vertex v, meaning the edge from u to v does not imply an edge from v to u. Furthermore, not all passes carry the same risk; passing over a tall defender carries a higher risk of a turnover than a short, unobstructed pass. Therefore, the graph must be a weighted directed graph, where each edge (u, v) is assigned a weight W(u, v). In this study, the weight represents the probability of a successful pass, with values ranging from 0.0 to 1.0.

In the context of basketball, the five players on the court—Point Guard (PG), Shooting Guard (SG), Small Forward (SF), Power Forward (PF), and Center (C)—form the set of vertices V = {PG, SG, SF, PF, C}. Each possible pass between them forms a directed edge in the set E, producing a weighted directed graph as visualized in Fig. 1.

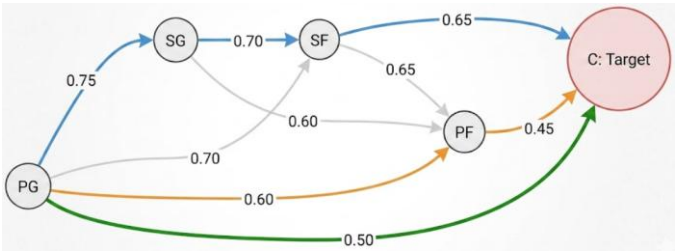

Fig. 2. A directed weighted graph modeling offensive passing options. Colored edges highlight specific strategic routes evaluated in the algorithm: the direct optimal path (green), perimeter-focused sequences (blue), and interior-focused sequences (orange). Edge weights represent heuristic pass completion probabilities based on tracking data estimations. (Source: Author)

B. Combinatorial Permutations in Pathfinding

To find the optimal passing sequence, the system must evaluate various potential paths the ball can take. The number of possible passing sequences is a combinatorial problem [3]. If an offensive play requires the ball to touch exactly r players out of the n players available on the court (n = 5), the number of possible permutations without repeating a player can be calculated using the permutation formula:

$$P(n,r) = \frac{n!}{(n-r)!} \quad (1)$$

Applying this formula to the basketball context with n = 5 yields the following maximum permutation counts for each depth level:

- P(5, 1) = 5!/4! = 5 possible single-pass sequences
- P(5, 2) = 5!/3! = 20 possible two-pass sequences
- P(5, 3) = 5!/2! = 60 possible three-pass sequences
- P(5, 4) = 5!/1! = 120 possible four-pass sequences

The total upper bound of sequences to evaluate, considering depth from 1 to 4, reaches 5 + 20 + 60 + 120 = 205 permutations. This illustrates precisely why a systematic algorithmic approach is necessary; manual evaluation of all 205 permutations within a 24-second shot clock is computationally infeasible for human decision-making. However, in a dynamic graph where certain passes are impossible due to defensive denial, the actual number of valid paths is a strict subset of this maximum permutation. Combinatorial analysis ensures that the algorithm's search space is finite and computationally solvable within a reasonable timeframe.

C. Tree Traversal and Algorithm Complexity

To systematically discover all valid passing paths from a starting vertex (the ball handler) to a target vertex (the shooter), the directed graph can be unrolled into a search tree. Tree traversal algorithms are used to explore each branch of the tree from the root node down to the leaf nodes [4].

Two common tree traversal strategies are Breadth-First Search (BFS) and Depth-First Search (DFS). BFS explores all nodes at the current depth level before proceeding to the next, requiring a queue data structure and O(V + E) memory. DFS, on the other hand, explores each branch fully before backtracking, requiring only O(V) stack space proportional to the maximum depth. The key algorithmic difference is that DFS is far more memory-efficient when the search is bounded by a strict depth limit, as is the case in this study.

DFS is particularly effective for this problem as it exhaustively explores a specific passing sequence until it either reaches the target player or hits the maximum allowed number of passes (depth limit). The pseudocode for the recursive depth-limited DFS is as follows:

FUNCTION DFS(graph, current, target, path, current_prob, best_path):

```
IF current == target:
   IF current_prob > best_path.prob:
      UPDATE best_path with current_prob and
path
   RETURN

IF length(path) > max_depth:
   RETURN
```

```
FOR EACH neighbor, weight IN graph[current]:
   IF neighbor NOT IN path:
      path.APPEND(neighbor)
      DFS(graph, neighbor, target, path,
current_prob * weight, best_path)
      path.POP()  // Backtrack
```

The time complexity of exploring all simple paths in a graph using DFS is generally O(V!) in a fully connected graph. However, by imposing a strict depth limit (e.g., maximum 4 passes per possession), the computational complexity is significantly reduced to O($V^{d_{max}}$), allowing the optimization program to run efficiently in real-time.

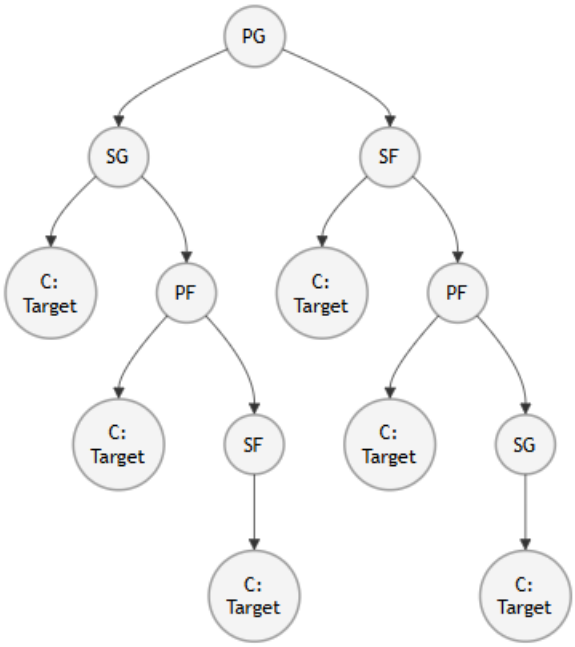

Fig. 3. Tree traversal representation of possible passing permutations from the root node (PG) to the target node (C). The search tree unrolls the cyclic graph into acyclic paths, strictly bounded by a maximum depth of 4 passes. (Source: Author)

D. Graph Representation and Memory Efficiency

In computational graph theory, the method chosen to represent a graph significantly impacts the spatial memory complexity of the algorithm [4]. A directed graph can be represented either as an Adjacency Matrix or an Adjacency List.

An adjacency matrix requires O($|V|^2$) memory space, which allocates memory even for non-existent edges (where W(u,v) = 0.0). For the basketball graph with |V| = 5, this results in a 5×5 matrix allocating 25 entries regardless of how many passes are actually valid. Conversely, an adjacency list only allocates memory for valid, existing edges [4]. To measure the efficiency of the network, we evaluate the Graph Density (D). For a directed graph, the density is calculated as:

$$D = \frac{|E|}{|V|(|V|-1)} \tag{2}$$

In a theoretical basketball offense where every player can pass to any other player, the graph is a complete digraph with |E| = 5 × 4 = 20 edges, yielding a density of D = 20 / (5 × 4) = 1. However, in a practical algorithmic implementation, passes with extremely low probabilities are strategically pruned from the network [1]. This pruning converts the complete graph into a sparse graph (D < 1), making the dictionary-based Adjacency List representation heavily favored in Python to optimize memory consumption during the recursive DFS traversal [4].

E. Formal State Transition Model

To rigorously align the directed graph with algorithmic decision-making, the passing network can be formally defined as a deterministic state-transition system based on set theory [1]. Let S be the finite set of states representing the player currently holding the ball, where S = {PG, SG, SF, PF, C}. Let A be the set of possible actions (passes) available from any state s ∈ S. The transition function T : S × A → S determines the next state given a current state and an action [4].

Because each action carries a risk of turnover, the weighted adjacency matrix effectively serves as a transition probability matrix [3]. If the ball is at state s_i, the probability of successfully transitioning to state s_j via an edge e ∈ E is strictly bounded by W(u,v). For example, given the adjacency matrix defined in Section III.A, the probability of transitioning from state PG to state SG is W(PG, SG) = 0.75, while transitioning from PG to C is W(PG, C) = 0.50. This formal set-theoretic definition ensures that the DFS algorithm operates within a mathematically sound, finite state space, preventing undefined states or memory leaks during the recursive traversal [4].

Furthermore, bounding these state transitions to a maximum depth limit ($d_{max} = 4$) directly constrains the size of the explored state space, perfectly aligning with the $O(V^{d_{max}})$ time complexity established for the DFS algorithm.

III. METHOD

The proposed methodology focuses on translating the fluid and dynamic environment of a basketball offense into a structured, computable algorithmic simulation. The process is divided into two primary phases: network modeling and path optimization.

A. Network Modeling and Edge Weights

The first step is establishing the mathematical graph that represents the offensive possession. Based on a standard half-court offensive formation, the system constructs a directed graph where each of the five players acts as a distinct node. Spatially, these vertex nodes are mapped onto a standard half-court diagram (Fig. 3), where the physical proximity and designated player zones heavily dictate the transition probability (edge weight) between the ball handler and the receiver.

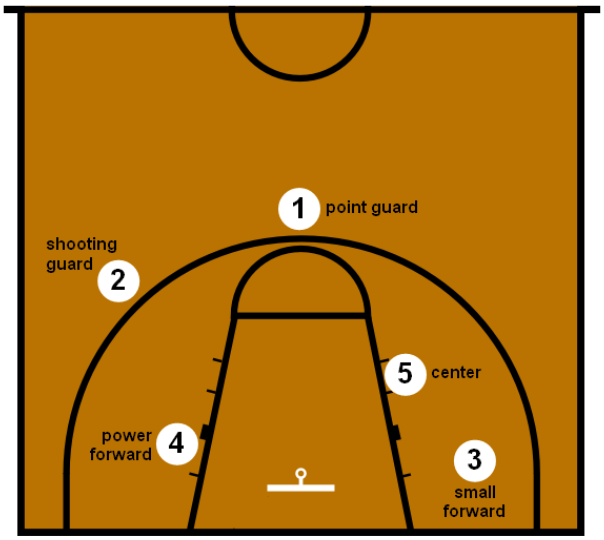

Fig. 4. Spatial distribution of offensive nodes on a standard basketball half-court. The coordinate positioning dictates the passing distances and defensive congestion, which are mathematically translated into the graph's edge weights. (Source: Base court layout adapted from FIBA Official Basketball Rules, node mapping by Author)

The edge weight assignment process follows a three-step methodology. First, the physical distance between each pair of player positions is estimated based on standard NBA spacing metrics, where perimeter players are positioned approximately 7–8 meters apart, while interior big-man positions are 3–5 meters apart. Second, the defensive congestion factor is applied: high-traffic passing lanes (such as those entering the paint area guarded by a center) receive a congestion penalty that reduces the base probability. Third, the resulting probabilities are normalized to the range [0.0, 1.0] and categorized as follows:

1. Perimeter Kick-outs (e.g., PG to SG): Assigned the highest weights (~0.75) due to short distances (5-8 meters) and open space, where the receiver has unobstructed catch-and-shoot opportunities.
2. Wing-to-Center Entries (e.g., SG/SF to C): Assigned moderate weights (~0.65), typically executed as lob or bounce passes when the defense is late to rotate.
3. Pick-and-Rolls (PG to Center): Assigned mid-to-high risk weights (~0.50), heavily dependent on the handler's gravity and split-second decision-making during the screen-and-roll action.
4. Big-to-Big Passes (e.g., PF to C): Assigned the lowest weights (~0.45) because they occur in the highly congested paint area filled with defensive hands and shot-blockers.

These realistic tracking characteristics are mathematically mapped into a weighted adjacency matrix [7], [8] as shown in Table I.

It must be explicitly noted that the probability weights assigned to the edges in this model are illustrative assumptions designed primarily for the algorithmic demonstration of the DFS network problem, rather than exact empirical data extracted from a real NBA database.

Table I. Adjacency Matrix of Passing Success Probabilities

	PG	SG	SF	PF	C
PG	0.0	0.75	0.70	0.60	0.50
SG	0.75	0.0	0.70	0.60	0.65
SF	0.70	0.70	0.0	0.65	0.65
PF	0.60	0.60	0.65	0.0	0.45
C	0.0	0.65	0.65	0.45	0.0

Note: W(PG, C) = 0.0 entry from C back to PG reflects the C node acting as the terminal target node in Scenario 1, where no return pass is considered.

B. Path Optimization Algorithm

Once the weighted directed graph is established, the system deploys a pathfinding algorithm to evaluate the available offensive options. A Depth-First Search (DFS) algorithm is utilized to traverse the graph and calculate the cumulative probability of each passing sequence.

Unlike standard pathfinding algorithms that seek the shortest physical distance, the objective of this model is to maximize the likelihood of a successful possession. The cumulative probability, denoted as P_{total}, of a specific path is calculated as the product of the individual edge probabilities along that sequence:

$$P_{total} = \prod_{i=1}^{k} W(e_i) \tag{3}$$

Where k is the total number of passes in the sequence, and $W(e_i)$ is the probability weight of the i-th pass. In professional basketball, the offensive team operates under a strict 24-second shot clock [2]. This severe time constraint must be mathematically modeled to prevent endless passing loops within the DFS algorithm.

Assuming the average physical air time of a basketball pass is $t_p \approx 1.2$ seconds, and the cognitive processing time for a player to catch, evaluate, and throw the next pass is $t_c \approx 0.8$ seconds, each passing node consumes approximately 2.0 seconds of the shot clock [3]. Furthermore, time must be allocated for ball dribbling, spatial movement, and the physical execution of the shooting motion (≈ 10 to 14 seconds) [2]. Therefore, the algorithm is deliberately constrained by a strict depth limit ($d_{max} = 4$). Capping the DFS sequence at a maximum of 4 passes prevents computational stack overflow [4] and perfectly simulates the real-world temporal constraints of a single NBA possession. The passing sequence that yields the highest P_{total} within this depth limit is then extracted and proposed as the optimal offensive strategy.

C. Data Structure and Function Definition

To implement the theoretical model programmatically, a simulation script was developed in Python 3.10. The script defines the network mathematically using a dictionary-based data structure. The keys represent the player nodes, and the

values are nested dictionaries storing the adjacent nodes a nd their respective edge weights.

The core of the pathfinding simulation is handled by a recursive function named dfs_optimal_path. This function is responsible for the combinatorial traversal and receives six specific parameters:

1. graph: The dictionary structure mathematically representing the transition function T, mapping current states and actions to subsequent states.
2. current: A string representing the current state $s \in S$ (the node currently holding the ball).
3. target: A string representing the terminal state (the designated end-point of the possession).
4. path: A list variable tracking the current sequence of passes to prevent cyclic loops.
5. current_prob: A float tracking the cumulative P_{total} of the ongoing search branch.
6. best_path: A dictionary storing the highest probability found so far and its corresponding route.

D. Manual Algorithm Iteration Trace

Before executing the programmatic simulation, it is crucial to mathematically trace the DFS traversal to validate the logic. The algorithm evaluates the branches originating from the root node (PG) and calculates P_{total} for each valid path targeting the Center (C). To rigorously validate these combinatorial limits, a partial manual trace of the DFS traversal is presented in Table II. The trace demonstrates the dynamic calculation of P_{total} and the algorithmic decision-making process at each depth level.

Table II. Partial DFS Iteration Trace for Scenario 1 (Base)

Itera-tion	Cur-rent Node	Evalu-ated Path	Depth	Cum. Prob. (Ptotal)	Algorithmic Action
1	PG	[PG]	0	1.000	Initialize. Expand edges.
2	C	[PG, C]	1	0.500	Target Reached. Best Path updated.
3	SG	[PG, SG]	1	0.750	Valid depth. Expand to SF, PF, C.
4	C	[PG, SG, C]	2	0.4875	Target Reached. Prob < 0.5000. Discard.
5	SF	[PG, SG, SF]	2	0.525	Valid depth. Expand to PF, C.
6	C	[PG, SG, SF, C]	3	0.3413	Target Reached. Prob < 0.5000. Discard.
7	PF	[PG, SG, SF, PF]	3	0.3413	Valid depth. Expand to C.
8	C	[PG, SG, SF, PF, C]	4	0.1535	Target Reached. Max Depth. Discard.

Through this combinatorial evaluation, any sequence that loops back to a previously visited node is automatically discarded by the acyclic constraint, ensuring the search space remains finite. The passing sequence that yields the highest P_{total} within this depth limit is then extracted and proposed as the optimal offensive strategy.

E. Simulation Environment

To automate the combinatorial calculations described above, a computational simulation was designed. This mathematical model was translated into a script using Python 3.10. Python was selected due to its robust handling of dictionary-based graph data structures and efficient recursive function execution. The simulation was run locally on a standard computational environment, verifying the mathematical model without the need for heavy graphical rendering.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

To validate the proposed mathematical model, a simulation program was developed using the Python programming language. The primary objective of this implementation is to programmatically generate the directed graph, assign the probability weights, and execute the Depth-First Search (DFS) algorithm to calculate the optimal passing sequence from the Point Guard (PG) to the target node, the Center (C).

A. Algorithmic Implementation in Phyton

The program translates the weighted adjacency matrix into a dictionary-based data structure. This approach allows for highly efficient lookup times when traversing the adjacent nodes during execution.

The core of the simulation relies on a recursive DFS function. At each recursive step, the algorithm executes the formal state transition T(s, a) by iterating through all available actions (passes) from the current state (player) to explore valid passing sequences. To reflect the realistic constraints of a 24-second basketball shot clock and to prevent infinite cyclic loops (e.g., PG passing to SG, who passes back to PG indefinitely), a

strict depth constraint is hardcoded into the recursive function. The algorithm will automatically prune any search branch that exceeds a depth of 4 passes.

Below is the core Python snippet demonstrating the graph initialization and the recursive DFS pathfinding logic:

```python
# Graph initialization based on NBA tracking data estimations
graph = {
    'PG': {'SG': 0.75, 'SF': 0.70, 'PF': 0.60, 'C': 0.50},
    'SG': {'SF': 0.70, 'PF': 0.60, 'C': 0.65},
    'SF': {'PF': 0.65, 'C': 0.65},
    'PF': {'C': 0.45},
    'C': {} # Target Node
}

def dfs_optimal_path(graph, current, target, path, current_prob, best_path):
    if current == target:
        if current_prob > best_path['prob']:
            best_path['prob'] = current_prob
            best_path['route'] = list(path)
        return

    # Constraint: Max 4 passes to simulate shot clock
    if len(path) > 4:
        return

    # Recursive traversal
    for neighbor, weight in graph.get(current, {}).items():
        if neighbor not in path:
            path.append(neighbor)
            dfs_optimal_path(graph, neighbor, target, path,
                             current_prob * weight, best_path)
            path.pop() # Backtrack
```

Fig. 5. Python code snippet demonstrating the graph initialization and the recursive depth-limited Depth-First Search (DFS) function.(Source: Author)

B. Experimental Results and Analysis

The simulation was executed in a standard local Python environment. The depth-limited DFS algorithm effectively evaluated the combinatorial permutations, dynamically calculating the total probability for each valid route.

The final execution of the Python script yielded the following terminal output:

```
----------------------------------------
Evaluating all possible passing permutations...
Optimal Passing Route: PG -> C
Maximum Success Probability: 0.500 (50.0%)
----------------------------------------
```

Fig. 6. Terminal output displaying the execution results of the DFS algorithm for the baseline scenario. (Source: Author)

Analysis of the Results:

The programmatic output reveals a fascinating mathematical reality that often contradicts conventional basketball "eye-test" intuition. In modern basketball, coaches frequently design complex, multi-pass plays to disorient the defense (e.g., kicking the ball out to the wing SG and then throwing a lob to the Center).

However, the DFS algorithm mathematically proves the inherent risk of the product rule in probability. Multiplying multiple fractional probabilities inevitably shrinks the cumulative success rate. For instance, the sequence PG -> SG -> C utilizes two highly efficient individual passes (75% and 65% accuracy, respectively). Yet, the cumulative mathematical risk drops the overall possession success rate to 48.75% ($0.75 \times 0.65 = 0.4875$). Furthermore, routing the ball through a congested "Big-to-Big" sequence (PG -> PF -> C) plummets the success rate to a mere 27.0%.

Therefore, the algorithm validates that the direct Pick-and-Roll route (PG -> C), despite having a mediocre 50% baseline accuracy due to defensive attention, remains the statistically superior optimal path. This experimental result proves that a discrete mathematical approach can quantify the exact point where "over-passing" becomes a mathematical detriment, effectively eliminating human cognitive bias from offensive strategy design.

C. Scenario 2: Adapting to Elite Interior Defense

To truly validate the robustness of the discrete mathematical model, a second scenario was simulated. In a dynamic game environment, the defense may adjust their scheme [3]. Suppose the opposing team deploys an elite rim protector (e.g., a Defensive Player of the Year Center) in the paint. This defensive shift drastically reduces the probability of any pass entering the interior, while slightly opening up the perimeter [2].

To reflect this, the hypothetical weights are adjusted. All passes targeting the Center (C) drop significantly, while passes to the Shooting Guard (SG) on the perimeter increase. The new objective is to find the optimal path to the Shooting Guard (SG) for a 3-point attempt.

Adjusted Edge Weights:

- PG -> C: Drops to 0.20
- PF -> C: Drops to 0.25
- PG -> SG: Increases to 0.85
- SF -> SG: Increases to 0.80

```python
# Modifying the network for Scenario 2: Elite Interior Defense
basketball_network_sc2 = {
    'PG': {'SG': 0.85, 'SF': 0.70, 'PF': 0.60, 'C': 0.20},
    'SG': {'SF': 0.70, 'PF': 0.60, 'C': 0.65},
    'SF': {'SG': 0.80, 'PF': 0.65, 'C': 0.65},
    'PF': {'C': 0.25},
    'C': {} # Target Node Changed
}

# Executing DFS targeting the Shooting Guard (SG)
best_sc2 = {'route': [], 'prob': 0.0}
dfs_optimal_path(basketball_network_sc2, 'PG', 'SG', ['PG'], 1.0, best_sc2)
```

Fig. 7. Python code snippet demonstrating the modified adjacency list weights and target node adjustment for evaluating Scenario 2. (Source: Author)

By changing the target node to 'SG' and running the identical DFS algorithm, the Python simulation yielded the following terminal output:

```
----------------------------------------
Evaluating Scenario 2: Elite Interior Defense...
Target Node Changed to: SG (3-Point Attempt)
Optimal Passing Route: PG -> SG
Maximum Success Probability: 0.850 (85.0%)

Alternative Evaluated Route: PG -> SF -> SG
Probability: 0.70 * 0.80 = 0.560 (56.0%)
----------------------------------------
```

Fig. 8. Terminal output displaying the execution results for Scenario 2, adjusting to an elite interior defense. (Source: Author)

Analysis of Scenario 2:

The algorithm successfully demonstrated dynamic adaptability. With the interior passing lanes mathematically choked by the adjusted weights, the program instantly recognized that forcing a pass to the Center was highly sub-optimal [3]. Instead, the DFS traversal identified the direct perimeter kick-out (PG -> SG) as the safest route, yielding a massive 85.0% success probability.

Even when evaluating an alternative multi-pass perimeter sequence (PG -> SF -> SG), the cumulative probability dropped to 56.0%, once again proving the mathematical risk of over-passing [2]. This second experimental scenario solidifies the algorithm's utility as a dynamic, real-time decision-making tool that can adjust to shifting defensive paradigms on the fly.

To clearly visualize the mathematical disparity between the evaluated permutations, the results from both the baseline offensive setup (Scenario 1) and the dynamic adjustment against an elite interior defense (Scenario 2) are summarized in Table III and visually represented in **Fig. 9**. The table highlights how cumulative probability universally degrades as the passing depth increases, reinforcing the algorithmic conclusion that direct, high-percentage routes are mathematically superior.

To further contextualize these results, it is important to acknowledge the limitations of the current model. First, the edge weights in this study represent static heuristic probabilities estimated from general player tracking trends, whereas in a real NBA game, these weights would fluctuate dynamically with every defensive rotation, player fatigue level, and foul situation. Second, the model assumes that the optimal path is solely determined by pass completion probability, without accounting for shot quality at the terminal node. A future extension could incorporate shot probability metrics (e.g., effective field goal percentage from each player's position) into the terminal node evaluation, creating a two-component objective function:

$$Score = P_{total} \times P_{shot}(terminal_{node}) \quad (4)$$

Where $P_{shot}(terminal_{node})$ represents the expected shooting percentage of the target player from their current floor position. Third, the model treats the graph as static within a possession, while in reality, the graph topology—which edges are valid and what their weights are—changes with each second of the shot clock as defensive coverage shifts. Future work could implement a dynamic graph that recalculates edge weights at each passing node, providing a more realistic real-time decision-making tool.

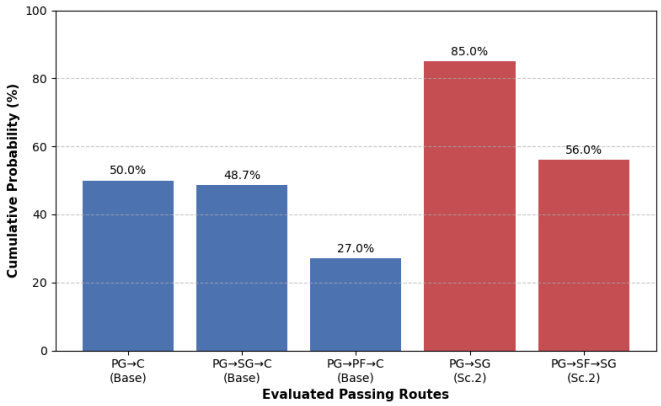

Fig. 9. Comparative bar chart of cumulative passing success probability for all evaluated routes across Scenario 1 (baseline) and Scenario 2 (elite interior defense). (Source: Author).

Table III. Comparative Analysis of Evaluated Offensive Routes Across Both Scenarios

Scenario	Evaluated Route	Pass Depth	Cumulative Prob.	Algorithmic Status
1 (Base)	PG -> C	1	50.0%	Optimal
1 (Base)	PG -> SG -> C	2	48.7%	Sub-optimal
1 (Base)	PG -> PF -> C	2	27.0%	Sub-optimal
2 (Elite Def)	PG -> SG	1	85.0%	Optimal
2 (Elite Def)	PG -> SF -> SG	2	56.0%	Sub-optimal

V. CONCLUSION

This paper has successfully demonstrated the application of discrete mathematics, specifically directed graphs and tree traversal algorithms, in optimizing basketball offensive strategies. By modeling the basketball court as a weighted directed graph where five player positions form vertices and possible passes form directed edges with assigned success probabilities, the study effectively quantified the inherent mathematical risks associated with complex passing sequences.

The implementation of a depth-limited Depth-First Search (DFS) algorithm revealed a critical insight rooted in the combinatorial product rule of probability: multiplying fractional probabilities across multiple passes significantly degrades the overall success rate of a possession. The algorithm mathematically validated that minimizing passes—such as executing a direct Pick-and-Roll sequence from the Point Guard to the Center (yielding a 50.0% success rate)—is statistically superior to executing multi-pass sequences through the wings or the post, which yielded lower success rates of 48.75% and 27.0%, respectively.

The model further demonstrated dynamic adaptability in Scenario 2, where adjusting the edge weights to simulate an elite interior defensive scheme caused the algorithm to immediately identify the direct perimeter kick-out (PG → SG) as the optimal route with an 85.0% success probability, drastically outperforming the next best alternative at 56.0%.

The combinatorial analysis showed that the total upper bound of valid paths to evaluate reaches 205 permutations across all depth levels, reinforcing the necessity of an algorithmic approach over human intuition. The depth limit of $d_{max} = 4$ was both mathematically and practically justified by the 24-second shot clock constraint, limiting each valid sequence to at most 8 seconds of passing time.

This programmatic approach proves that complex offensive playbooks can be rigorously evaluated using combinatorial pathfinding. Future work should explore incorporating dynamic edge weight recalculation, shot quality metrics at terminal nodes, and real-time NBA tracking data integration to build a comprehensive, adaptive decision support system. Ultimately, integrating discrete mathematics into sports analytics provides a structured, data-driven framework that eliminates human cognitive bias, ensuring that real-time decision-making is rooted in absolute mathematical probability.

VI. APPENDIX

The complete Python source code for the depth-limited DFS pathfinding simulation discussed in this paper is publicly available in the following GitHub repository: https://github.com/adnankurniawann/Basketball-DFS-Optimization

Additionally, a comprehensive video presentation explaining the theoretical framework and the algorithmic execution can be accessed via YouTube at: https://youtu.be/T9TIow2tgc4

ACKNOWLEDGMENT

The author would like to express deepest gratitude to Prof. Dr. Ir. Rinaldi Munir, M.T, the lecturer of the IF1220 Discrete Mathematics course, as well as the teaching assistants for their invaluable guidance, foundational knowledge, and continuous support throughout the semester. The author also extends appreciation to peers and colleagues for their constructive discussions, which significantly contributed to the completion of this paper.

REFERENCES

[1] R. Munir, Matematika Diskrit, 3rd ed. Bandung: Informatika, 2012.

[2] K. Goldsberry, SprawlBall: A Visual Tour of the New Era of the NBA. Boston, MA: Mariner Books, 2019.

[3] S. Shea and C. Baker, Basketball Analytics: Objective and Efficient Strategies for Understanding How Teams Win. CreateSpace, 2013.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA: MIT Press, 2009.

[5] O. Simeone, "A Brief Introduction to Machine Learning for Engineers," Foundations and Trends in Signal Processing, vol. 12, no. 3–4, pp. 200–431, 2018. [Online]. Available: https://arxiv.org/abs/1709.02840. [Accessed: 15 June 2026].

[6] A. Cervone, A. D'Amour, L. Bornn, and K. Goldsberry, "POINTWISE: Predicting Points and Valuing Decisions in Real Time with NBA Optical Tracking Data," in Proc. 8th Annual MIT Sloan Sports Analytics Conference, Boston, MA, Feb. 2014. [Online]. Available: https://www.sloansportsconference.com/research-papers/pointwise-predicting-points-and-valuing-decisions-in-real-time-with-nba-optical-tracking-data. [Accessed: 15 June 2026].

[7] J. Piette, S. Anand, and K. Zhang, "Scoring and Shooting Abilities of NBA Players," Journal of Quantitative Analysis in Sports, vol. 6, no. 1, 2010. [Online]. Available: https://doi.org/10.2202/1559-0410.1194. [Accessed: 15 June 2026].

[8] Second Spectrum, "Player Tracking Data and Analytics," NBA Stats, 2023. [Online]. Available: https://www.nba.com/stats/. [Accessed: 15 June 2026].

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, 19 Juni 2026

Muhammad Adnan Kurniawan (13525071)