Modeling Basketball Help Defense: A Directed Graph Approach to Execution Timing and Recovery Path Optimization

Ernest Clarence Gunawan - 13525114
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ernestclarenceg@gmail.com , 13525114@std.stei.itb.ac.id

***Abstract*—Basketball help defense requires coordinated rotations and recoveries that must be executed within tight time constraints, yet these actions are typically evaluated through intuition and film study rather than formal quantitative methods. This paper proposes a directed graph framework that models help defense as a weighted directed temporal graph, where vertices represent defensive position-role states and edges represent feasible transitions with time-based weights. A timing model adapted from critical path analysis is introduced to compute earliest and latest feasible arrival times and evaluate defensive feasibility against offensive time thresholds. The recovery problem is formulated as a shortest path problem for single-defender recovery and as a linear assignment problem for multi-defender recovery, with a risk-weighted extension to account for positional danger. A case study demonstrates that the optimization framework converts an infeasible 1.80-second closeout into a feasible 1.45-second recovery, with sensitivity analysis confirming robustness to reasonable parameter variation. The framework offers a formally grounded, computationally tractable tool for analyzing and improving help-defense schemes.**

Keywords—basketball analytics, help defense, directed graph, temporal graph, critical path analysis, recovery optimization.

I. Introduction

At its simplest, basketball is about putting the ball through the hoop. But if that were all that mattered, the team with the best shooters would always win, and that is simply not the case. More often, the outcome of a possession is decided before the shot is even taken, by whether the defense can deny, delay, or disrupt it. A team that consistently makes scoring difficult, forcing rushed attempts, contested looks, or turnovers, controls the flow of the game even when its own shooting numbers are unremarkable. This is why defense is widely regarded by coaches and analysts as one of the most decisive factors in basketball success, regardless of level or league.

Among the various defensive schemes employed by teams, help defense, the act of a defender temporarily leaving their assigned matchup to contain a more immediate offensive threat, stands out as one of the most demanding to execute correctly. A help rotation that arrives a fraction of a second too late allows an open shot; one that recovers along an inefficient path leaves a different offensive player unguarded, creating a chain reaction of defensive breakdowns. Despite the tactical importance of this sequence, the timing and pathing of help-and-recover actions are still evaluated mostly through coach intuition and film study rather than through a formal quantitative framework.

Recent work in basketball analytics has started to look at parts of this problem. Supola, Hoch, and Baca built a machine learning model that can detect defensive openings forming during a drive. Their model can identify these gaps slightly before they actually happen, which shows that defensive gaps can be measured and even predicted from player tracking data. On the other side, some researchers have used network-based approaches to model teams as graphs, where players are nodes and passes or movements are edges. These approaches have mostly been used to study offensive spacing and ball movement. Together, these studies confirm two things: first, timing in defense matters and can be modeled, and second, graph-based representations are useful for basketball. But neither of these approaches directly answers the question of how a sequence of help-and-recovery actions among multiple defenders should be represented as a connected, directional system where both execution timing and recovery efficiency can be modeled and optimized together.

This paper proposes that a directed graph is a natural and mostly unexplored way to represent this problem. If we model defensive assignments as nodes and the possible rotation actions between them as directed, weighted edges, where each edge weight represents time cost, then two practical coaching questions can be turned into solvable graph problems. The first question is: at what point in a rotation sequence must each action be executed to prevent an open shot? The second question is: once a help action has been made, what path allows the helping defender to recover most efficiently to a sound defensive position?

This paper contributes in three ways. First, help defense rotations are formalized as a directed graph with node and edge definitions grounded in basketball concepts. Second, a timing model is introduced to address the time-sensitive execution within the graph, which is largely absent from existing network-based analytics. Third, shortest-path optimization is applied to

define and solve the recovery path problem, illustrated through a case study comparing optimal and naive recovery paths.

II. Graph Theory Foundations

A. Basic Graph Definitions

A graph is a mathematical structure used to represent discrete objects and the relationships between them. Formally, a graph G is defined as

$$G = (V, E)$$

where V is a set of vertices (also called nodes), and E is a set of edges connecting pairs of vertices. A graph must have at least one vertex, but it does not necessarily need to have any edges. The degree of a vertex v, written $d(v)$, is the number of edges connected to it.

B. Simple and Non-Simple Graphs

Graphs can come in many different forms, which has led to several classifications. One of the first things that distinguishes graphs is whether they have multiple edges or loops.

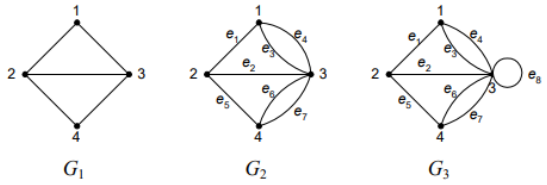

Fig. 2.1. (a) graph with no multiple edges and loop, (b) graph with multiple edges, (c) graph with multiple edges and loop
(Source: https://informatika.stei.itb.ac.id/~rinaldi.munir)

When two edges connect the same pair of vertices, they are referred to as multiple edges or parallel edges. In Figure 2.1, for instance, graph G_2 contains two parallel edges, $e_3 = (1,3)$ and $e_4 = (1,3)$, both joining vertex 1 and vertex 3. Conversely, an edge that starts and ends at the same vertex is known as a loop, such as $e_8 = (3,3)$ in graph G_3 of Figure 2.1, which forms a loop at vertex 3.

Based on the existence of multiple edges or loops, graphs are classified into two categories. If a graph has neither multiple edges nor loops, it is called a simple graph. If it has at least one, it is called a non-simple graph, which is then further split into two more specific types: a multigraph, which contains multiple edges but no loops, and a pseudograph, which contains at least one loop.

For this paper, we restrict our attention to simple graphs. This restriction is appropriate because each defensive transition from one position to another is unique and only needs to be represented once.

C. Directed and Weighted Graphs

In undirected graphs, edges are symmetric: an edge connecting u and v is equivalent in both directions. In directed graphs, however, edges have a direction, so (u, v) and (v, u) are distinct; one does not imply the other.

This difference also affects how we measure degree. Directed graphs use two measures of degree: in-degree (edges entering) and out-degree (edges leaving). A vertex v is reachable from vertex u if a directed path from u to v exists. A directed graph is strongly connected if every pair of vertices satisfies mutual reachability, while it is weakly connected if connectivity holds only when edge directions are ignored.

This distinction is fundamental to our work. Defensive rotations are inherently asymmetric: a rotation from position u to position v does not imply that the reverse rotation is feasible or carries the same cost. The directed graph formalism thus provides a precise representation of the one-directional, sequential structure of defensive assignments.

Regardless of direction, edges may carry a numerical value called a weight, forming a weighted graph. A weighted directed graph combines both properties: a weight function $w : E \rightarrow \mathbb{R}^+$ assigns a positive real number to each directed edge, $w(u, v)$ represents the time required for a defender to execute the transition from position u to position v, incorporating physical distance, running speed, and reaction time as components of a composite cost function developed in Chapter III.

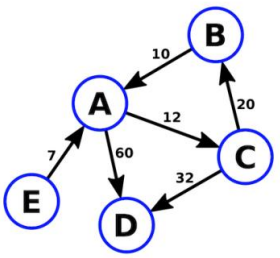

Fig. 2.2. Directed and Weighted Graph
(Source: https://study.com/academy/lesson/weighted-graphs-implementation-dijkstra-algorithm.html)

D. Paths, Cycles, and Weigthed Path Length

A directed path of length n from vertex v_0 to vertex v_n is a sequence of vertices $v_0, v_1, v_2, \ldots, v_n$ such that a directed edge (v_i, v_{i+1}) exists for every consecutive pair. A path that begins and ends at the same vertex is called a cycle.

In a weighted directed graph, the weighted path length (or path cost) is the sum of edge weights along the path:

$$cost(P) = \sum w(v_i, v_{i+1}) \; for \; i = 0 \; to \; n-1$$

In the context of this paper, a path represents a sequence of defensive rotations. For instance, a defender moving from their initial position to a help position and then recovering to their primary assignment. The path cost, therefore, represents the total execution time required to complete this sequence. Minimizing this cost is the central optimization objective addressed in Chapter V.

E. Temporal Graphs (Time-Expanded Graphs)

Classical graph models assume that edge weights are fixed costs that do not depend on when a traversal occurs. This assumption does not work well for modeling basketball defense, where the timing of each rotation — not just the overall cost — is what determines whether the defense is successful. A rotation

that is geometrically efficient can still be ineffective if it arrives too late relative to the offensive action it is supposed to counter.

To address this, we introduce the temporal graph (also known as a time-expanded graph). A temporal graph is a directed graph where each edge (u, v) is labeled with a time interval or a departure time constraint that specifies when the traversal is allowed. Formally:

$$G_t = (V, E, \lambda)$$

where V and E are the usual vertex and edge sets, and λ : E → T assigns each edge to the time step or interval during which it is valid. A time-respecting path from *u* to *v* is a directed path where the time labels of consecutive edges are non-decreasing, meaning a traversal cannot start before the previous one has finished.

In the time-expanded graph approach, the graph is replicated across discrete time steps. Each vertex *v* at time *t* becomes a distinct node (*v*, *t*), and edges are added from (*u*, *t*) to (*v*, *t'*) only when the transition from *u* to *v* is feasible from time *t* to *t'*. This converts a temporal routing problem into a standard shortest path problem on a larger static graph, allowing classical algorithms to be applied directly.

F. Shortest Path Algorithms

The shortest path problem seeks the minimum-weight directed path between two vertices in a weighted directed graph. This problem forms the computational foundation for the recovery path optimization discussed in Chapter V.

Dijkstra's Algorithm

Dijkstra's algorithm is designed to solve the single-source shortest path problem in graphs where all edge weights are non-negative. Starting from a source vertex *s*, it keeps a list of the best distances found so far, always picks the closest unprocessed vertex, finalizes its distance, and checks its neighbors. The process ends once all reachable vertices have been finalized.

When implemented with a binary heap, the time complexity of Dijkstra's algorithm is O((V + E) log V). Since all edge weights in our model represent execution times, which are strictly positive, this algorithm is directly applicable and serves as the main algorithm for computing optimal recovery paths in Chapter V.

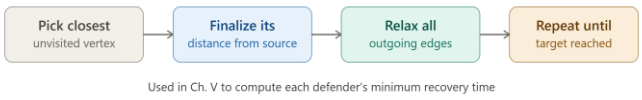

Figure 2.3. Dijkstra Flow

Bellman-Ford Algorithm

The Bellman-Ford algorithm solves the single-source shortest path problem on graphs that may contain negative-weight edges. The algorithm will repeatedly scans all edges, updating the distance to each vertex whenever a better path is discovered. This process is repeated V − 1 times. After that, one additional scan is performed to detect negative cycles. Its time complexity is O(VE).

While the non-negative timing weights of our primary model do not require Bellman-Ford, the algorithm is still relevant as a theoretical reference. It becomes useful in extensions of the model where edge weights represent net defensive advantage, a value that could be negative if a rotation leaves a more dangerous position exposed than the one it covers. Furthermore, Bellman-Ford provides the theoretical foundation for the critical path method discussed in Chapter IV.

III. Modeling Help Defense as a Directed Graph

With the graph-theoretic foundations established in Chapter II, this chapter now translates the structure of basketball help defense into a formal directed graph model. We define the graph components — vertices, edges, and weights — in terms of their defensive meanings, introduce the constraints that dictate valid defensive configurations, and present the complete formal model that will be used throughout the rest of the paper.

Defensive Context and Motivation

Help defense in basketball is the coordinated movement and positional adjustment of off-ball defenders when a ball-handler drives toward the basket or beats their primary defender. Effective help defense demands more than just arriving at the right spot — the helper must also arrive within a specific time window, before the offensive player can take advantage of the opening. Once a defender commits to helping, the entire defensive structure shifts: adjacent defenders must rotate to cover the helper's original assignment, and the helper must eventually recover back to their own primary responsibility.

This process is inherently sequential, directional, and time-sensitive. A rotation from position A to position B is not the same as a rotation from B to A; the cost of each transition is determined by physical distance and player speed; and whether a rotation succeeds depends on when it happens relative to the offensive action. These characteristics align naturally with the structure of a weighted directed temporal graph, which we formalize below.

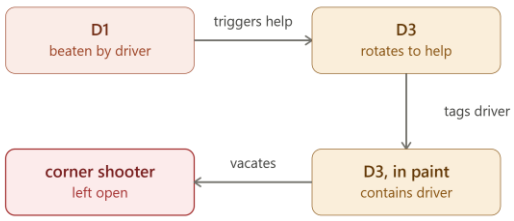

Figure 3.1. Help Defense Structure

A. Vertex Definition

Each vertex in the graph represents a defensive position-role state, which is a combination of: a spatial zone on the court, a defensive role assigned to that zone (primary defender, helper, or recoverer), and a discrete time step at which that state is occupied. Formally:

$$V = \{(p, r, t) \mid p \in P, r \in R, t \in T\}$$

where P is the set of spatial positions (for example, strong-side block, weak-side elbow, paint center, or three-point corner), R = {primary, helper, recoverer} is the set of defensive roles, and T is the discrete set of time steps that make up the defensive sequence.

In practice, not every combination of (p, r, t) is occupied at the same time. The set of active vertices at any given time step t is limited by the number of defenders on the court and the specific defensive scheme being used. For the sake of clarity, the examples in this paper use a five-defender setup, with positions labeled P1 through P5 corresponding to the standard help-defense zones.

B. Edge Definition

A directed edge $(u, v) \in E$ from vertex $u = (p_1, r_1, t_1)$ to vertex $v = (p_2, r_2, t_2)$ represents a feasible defensive transition. This means a defender currently in state u moves to state v by executing a rotation, switch, or recovery action. Such an edge exists in the graph if and only if all of the following conditions are met:

- The transition from position p_1 to position p_2 is physically reachable within the time interval $[t_1, t_2]$ given the defender's movement speed.
- The change in role from r_1 to r_2 must be allowed under the defensive scheme. For instance, a primary defender may become a helper, but two defenders cannot both occupy the helper role for the same assignment at the same time.
- No other defender is already at p_2 at time t_2, unless the scheme allows zone overlap.

These rules ensure the graph only includes valid defensive actions, not all possible movements.

C. Edge Weight: The Cost Function

Each directed edge (u, v) carries a weight $w(u, v) \in \mathbb{R}^+$ representing the total execution cost of the transition. The weight is modeled as a composite function:

$$w(u, v) = \alpha \cdot \frac{d(p_1, p_2)}{s} + \beta \cdot \tau_{react} + \gamma \cdot \tau_{read}$$

where $d(p_1, p_2)$ is the Euclidean distance between p_1 and p_2, s is the defender's average movement speed for that transition type (lateral slide, sprint, or closeout), τ_{react} is the defender's reaction time from the moment the offensive trigger occurs, τ_{read} is the time needed to process the play and choose the correct rotation; and α, β, γ are non-negative scaling factors that can be adjusted to fit different defensive schemes and player abilities.

This structure separates the physical part of the transition cost (distance and speed) from the cognitive parts (reaction and decision time). This separation makes the model flexible, so it can be adapted to different players and systems without the need to change the graph structure.

D. Complete Formal Model

The complete help defense model takes the form of a weighted directed temporal graph:

$$G_D = (V, E, w, \lambda)$$

where V is the set of vertices defined in Section A, E is the set of edges defined in Section B, $w : E \to \mathbb{R}^+$ is the composite cost function defined in Section C, and $\lambda : E \to T$ is a function that assigns each edge to the time step at which the transition begins.

IV. Execution Timing Model

The directed graph model defined in Chapter III establishes which defensive transitions are structurally possible and how costly each one is in terms of execution time. This chapter now focuses on the timing aspects of the model: how to interpret path cost as a total rotation time, how that total is compared against the time limit, and how earliest and latest feasible arrival times are computed using critical path analysis.

A. Total Rotation Time and Critical Treshold

Given a time-respecting path $P = (v_0, v_1, \ldots, v_n)$ in the defensive graph G_D, the total rotation time is the sum of edge weights along the path:

$$T_{rot}(P) = \sum_{i=0}^{n-1} w(v_i, v_{i+1})$$

This represents the elapsed time from when the defensive rotation is triggered to when the last defender arrives at their target position.

Each defensive scenario also has critical time thresholds that define the latest acceptable arrival time for each vertex. Let $t(v)$ be the critical threshold for vertex v. A path P is feasible if and only if:

$$t_a(v_i) \leq t(v_i) \; for \; every \; step \; v_i \; in \; the \; path$$

where $t_a(v_i)$ is the time elapsed from the start of the sequence to the arrival at vertex v_i.

These thresholds depend on the offense: for help defense, it corresponds with the time before the ball-handler scores; for closeouts, it corresponds with the time before the shooter releases the ball. We treat these as inputs to the model, based on real game data.

B. Earliest and Latest Arrival Times

To identify time-critical transitions, we compute two quantities for each vertex using critical path method (CPM) analysis:

- Earliest Arrival Time (EAT): The minimum time at which vertex v can be reached, computed by a forward pass where u is all predecessor of v:

  $$EAT(v) = min_u[EAT(u) + w(u, v)]$$

  with $EAT(s) = 0$ for the source vertex s
- Latest Arrival Time (LAT): The maximum time at which vertex v can be reached without violating downstream thresholds, computed by a backward pass where v is all predecessor of w:

  $$LAT(v) = max_w[LAT(w) - w(v, w)]$$

with $LAT(sink) = t(sink)$ for the final target vertex

C. Slack and Critical Transitions

The slack of a vertex v is defined as:

$$slack(v) = LAT(v) - EAT(v)$$

A vertex with zero slack is called a critical vertex, any delay at this point will carry forward and cause a failure somewhere downstream. The sequence of critical vertices forms the critical path, which represents the chain of transitions that must be executed perfectly, with no room for error.

Finding the critical path has real practical value for coaches. It shows exactly which rotations are most vulnerable to timing mistakes and thus need the most attention in practice. Rotations that are not on the critical path have positive slack, meaning they can afford some delay without breaking the defense. This helps coaches focus their training time on the most time-sensitive parts of the scheme.

D. Feasibility Condition

A complete defensive sequence is feasible if and only if the earliest arrival time at every vertex stays within its critical threshold:

$$EAT(v) \leq t^*(v) \; for \; all \; v \in V$$

If this condition is not met at any vertex, it means that the defensive scheme is just not enough for that offensive situation with the players available. This tells coaches what went wrong, so they can decide whether to change players or adjust the defensive scheme.

V. Recovery Path Optimization

So far, we've built a model of help defense using a directed graph with timing rules. Now, in this chapter, we turn to the second major problem of this paper: after a defender has rotated to help and the immediate threat has been stopped, how should that defender, and possibly others, return to their original assignments in a way that minimizes both the total recovery time and the defensive vulnerability left behind?

The Recovery Problem

After a help rotation is executed, the defending team is left in a transitional state: the helper *h* is displaced from their primary assignment p_h, and the defensive structure has one or more temporarily uncovered positions. The recovery problem is to find a path in the defensive graph from the post-help vertex of *h* back to their primary assignment vertex, or in the multi-defender case, to find an assignment of defenders to positions that minimizes total recovery cost across the team.

This problem is formulated as a shortest path problem in the single-defender case and as an assignment problem in the multi-defender case, as detailed in the sections below.

A. Single-Defender Recovery as Shortest Path

Suppose *h* is the help defender. After helping, they are at position $p_{h'}$ at time t_h. They need to return to their own assignment at position p_h before the recovery time limit runs out.

To find the best way back, we look for the path that has the smallest total cost while still following all the rules. Path cost is the sum of edge weights along the route.

This is a classic shortest path problem, which we can solve using Dijkstra's algorithm. The algorithm tells us both the fastest recovery time and exactly which moves the defender should make.

B. Multiple-Defender Recovery as Assignment Problem

In many help defense situations, more than one defender ends up moving. When the helper rotates, the defenders next to them also shift to cover the gaps, which creates a chain of movements. In this case, the recovery problem is no longer about getting just one defender back to their spot. Instead, we need to reassign all the defenders who moved to their correct positions in a way that minimizes the total cost for the whole team.

Let $D = \{d_1, d_2, \ldots, d_k\}$ be the set of displaced defenders and $A = \{a_1, a_2, \ldots, a_k\}$ be the set of unoccupied primary assignments that must be recovered. Define $c(d_i, a_j)$ as the minimum recovery cost for defender d_i to recover to assignment a_j, computed by the single-defender shortest path method of Section A Chapter V. The multi-defender recovery problem is then:

$$min \sum_i c\big(d_i, a_\sigma(i)\big) \; over \; all \; permutations \; \sigma \; of \; A$$

This problem is a type of linear assignment problem, which can be solved efficiently using the Hungarian algorithm. The algorithm runs in polynomial time, specifically $O(k^3)$, where k is the number of displaced defenders. In practice, however, a single help rotation usually displaces only three or four defenders at most, so the computation is fast enough to be used in real time or near-real time.

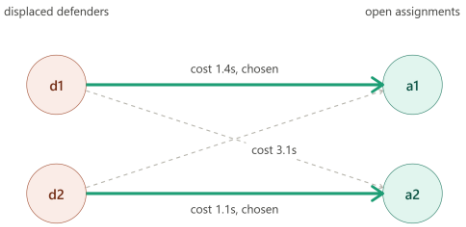

Figure 5.1. Hungarian algorithm picking the pair with lowest total cost

C. Risk-Weighted Recovery

The formulations in the previous sections focus on minimizing total recovery time. However, not all open positions on the court are equally dangerous. Leaving a corner three-point shooter open, for instance, is much more risky than leaving a player open in the mid-post area, especially when the opposing team has good shooters in the corners.

To address this, we extend the cost function to include a risk factor. The risk-weighted cost for defender d_i to recover to assignment a_j is:

$$c_{risk}(d_i, a_j) = c(d_i, a_j) + \lambda \cdot \rho(a_j) \cdot t_e(d_i, a_j)$$

where $\rho(a_j) \in [0,1]$ is a threat coefficient that quantifies the offensive danger associated with leaving assignment a_j uncovered; $t_e(d_i, a_j)$ is how long that position stays unoccupied while the defender is on their way; and λ controls the relative importance of time-efficiency versus risk-minimization.

D. Summary

This chapter's recovery framework pulls together the structural model from Chapter III and the timing model from Chapter IV into one optimization problem that computers can solve. For one defender, Dijkstra's algorithm finds the best path. For multiple defenders, we use an assignment problem approach that works for the whole team. We also add a risk factor to account for dangerous open spots, so the recovery plan balances speed with staying safe on defense.

VI. Case Study / Simulation

A. Scenario Setup

To validate the practical utility of the model developed in Chapters III–V, we construct a concrete five-defender scenario and compute both the optimal recovery path and a naive baseline path, then compare their total execution times.

Here's the setup. Consider a standard help-defense trigger: the ball-handler (O2) at the top of the key drives past their primary defender (D1) toward the rim. D1's primary assignment lapses, and weak-side defender D5, who's positioned at the weak-side corner, rotates to help, tagging the driver before a layup attempt. This leaves D5's original assignment (a corner three-point shooter, O5) uncovered. Simultaneously, D3, who was guarding the weak-side elbow adjacent to D5's old position, must decide whether to rotate over to cover O5 or hold position, and D1 must recover from the beaten drive back towards the paint.

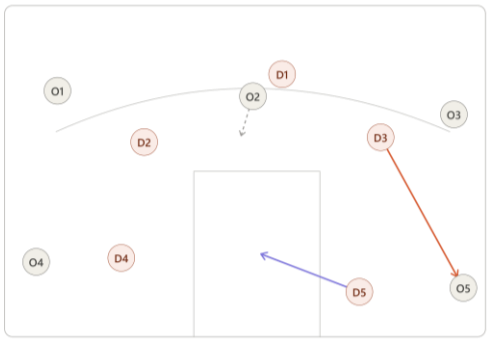

Fig. 6.1. *Five-defender help-defense scenario*
(Source: Author)

Vertices involved (using the position labels P1–P5 introduced in Section 3.2, with roles abbreviated as primary = pr, helper = h, recoverer = rc):

TABLE I. Vertices Involved in Scenario Setup

Vertex	Position	Role	Time step
v0	P1 (top of key, guarding O2)	pr (beaten)	t = 0
v1	P2 (weak wing, guarding O1)	pr	t = 0
v2	P3 (strong wing, near O3)	pr → h (closes out towards O5)	t = 0
v3	P4 (strong corner, guarding O4)	pr	t = 0
v4	P5 (paint / weak corner)	h → rc (reassigned to cover O3)	t = 0
v5	O5 (weak corner, vacated shooter)	target (closeout / coverage)	-

Edge weights, computed using the composite cost function from Section C of Chapter III ($w = \alpha \cdot d/s + \beta \cdot \tau_{react} + \gamma \cdot \tau_{read}$, with $\alpha = \beta = \gamma = 1$ for simplicity, and speeds in m/s):

TABLE II. Edge Weights in Scenario Setup

Edge	Distance (m)	Speed (m/s)	τ_{react} (s)	τ_{read} (s)	Weigth w (s)
v0 → v_{paint} (D1 recovers after being beaten by O2	4.0	4.0	0.20	0.05	1.25
v1 → v5 (D3 closes out to O5 in weak corner)	8.5	5.5	0.15	0.10	1.80
v4 → v_{paint} (D5 rotates from weak corner into paint)	5.0	4.0	0.15	0.05	1.45

Parameters in Table II are illustrative estimates based on typical reported ranges in sports performance literature, not measurements from a specific play

Critical threshold. Tracking data for catch-and-shoot corner threes against high-quality, quick-release shooters gives a shooter preparation time of roughly 1.7 seconds from the pass leaving the ball-handler's hands. We set $t^*(O5) = 1.7$ s, measured from the moment D3 begins the closeout rotation ($v2 \rightarrow v5$).

B. Naïve Path (Baseline)

The naive strategy is a simple "backtracking" approach: the helper simply returns along the exact same path they took to help, without considering whether another defender might be in a better position to cover the open spot.

Naive path: D3 executes $v2 \rightarrow v5$ (closes out from the strong wing directly to O5 in the strong corner)

$$T_{naive} = w(v2, v5) = 1.80\ seconds$$

Since T_{naive} (1.80s) exceeds $t^*(O5) = 1.7$s, this closeout is defensively infeasible: by the time D3 arrives, O5 has already had enough time to catch and release the shot.

C. Optimal Path (Model-Derived)

Applying the multi-defender assignment formulation from Section B of Chapter V, the model considers reassigning recovery responsibility instead of relying solely on D3.

D5, who is already closer to the corner because of the spacing created by O2's drive, rotates to cover O5. Meanwhile, D3 drops down toward the paint to protect the rim instead, achieving a shorter and more realistic transition for D3.

$$T_{D5} = w(v4, v5) = 1.45\ seconds$$

Since T_{D5} (1.45s) is well under t^*(O5) = 1.7s, this assignment is feasible, with a slack = 1.7 − 1.45 = 0.25 seconds.

D. Comparison

TABLE III. COMPARISON OF NAÏVE AND OPTIMAL PATH

Strategy	Path	Total time	vs. threshold (1.7s)	Feasible?
Naive (D3 closes out alone)	D3: v2 → v5	1.80 s	+0.10 s over	No
Optimal (D5 reassigned to cover O5)	D5: v4 → v5	1.45 s	-0.25 s under	Yes

Fig. 6. Total recovery time of the naive path versus the model-optimal path, plotted against the critical threshold t* = 1.7s.

The model-based assignment is 0.35 seconds faster than the naive closeout and turns an infeasible sequence into a feasible one.

It is worth understanding why the naive path fails while the optimal one works. The naive approach assumes the nearest obvious defender, D3, is the right one to close out. But D5 actually has a shorter, more direct path to O5 and doesn't have to first disengage from another responsibility. That's the key difference.

E. Sensitivity Note

To assess the sensitivity of our results to parameter selection, we re-ran the computation with α, β, and γ each varied by ±20%. Under these variations, the naive closeout time for D3 ranged from 1.44 to 2.16 seconds, while D5's transition time ranged from 1.16 to 1.74 seconds. Across this entire range, D3's closeout remained infeasible for most of the tested variations, whereas D5's reassigned path remained feasible almost everywhere, only approaching the critical threshold at the very upper extreme. This shows that our result is robust to reasonable changes in the parameters, rather than being a fluke caused by the specific numbers we chose.

VII. DISCUSSION

Limitations

It is worth noting that the current model is built on several simplifying assumptions.

- Constant-speed assumption. The cost function in Section C of Chapter III assumes a defender's speed s is constant for each transition type. In reality, defenders accelerate, decelerate, fatigue over time, and move slower laterally than forward. This means the model underestimates costs for longer moves and overestimates them for very short ones where acceleration matters most.
- Static, manually-estimated parameters. The reaction time and read time parameters used in Chapter VI were estimated, not measured. These vary between players and are influenced by fatigue, experience, and the opponent's style.
- Absence of real tracking data. The model has not yet been validated against actual NBA or college tracking data. The numbers used are reasonable guesses, not measured values.
- Deterministic offensive behavior. The model assumes the offense follows a fixed path once the trigger occurs. In reality, the offense may hesitate, change direction, or pass, requiring the graph to be re-evaluated dynamically rather than solved once at the start.

VIII. CONCLUSION

This paper has presented a directed graph framework for modeling basketball help defense, with particular emphasis on the timing and optimization of defensive recovery. Building upon standard graph-theoretic foundations, we extended the model into the temporal domain to capture the time-dependent nature of defensive rotations, and formalized execution timing using concepts from critical path analysis, namely earliest and latest arrival times, slack, and critical thresholds. We then formulated the recovery problem explicitly as a shortest path problem for single-defender recovery and as a linear assignment problem for multi-defender recovery, with an extension to incorporate positional risk.

The case study in Chapter VI demonstrated, through a concrete worked example, that this optimization framework can identify recovery assignments that are both faster and defensively feasible where a naive heuristic is not, converting an infeasible 1.80-second recovery into a feasible 1.45-second one by reassigning recovery responsibility to a better-positioned defender. This supports our main point: treating recovery as a graph problem has real value for defense.

The limitations discussed in Chapter VII, including constant-speed assumptions, estimated rather than tracked parameters, and the absence of opponent adaptation, define clear directions for future research. Even within these limitations, the framework

offers a formally grounded and computationally tractable tool for analyzing and improving help-defense schemes, and establishes directed, weighted temporal graphs as a productive lens through which to study coordinated team defense more broadly.

REFERENCES

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

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, 1959.

[3] R. Bellman, "On a routing problem," Quarterly of Applied Mathematics, 1958.

[4] H. W. Kuhn, "The Hungarian method for the assignment problem," Naval Research Logistics Quarterly, 1955.

[5] J. E. Kelley Jr. and M. R. Walker, "Critical-path planning and scheduling," in Proc. Eastern Joint Computer Conf., 1959.

[6] O. Michail, "An introduction to temporal graphs: An algorithmic perspective," Internet Mathematics, 2016.

[7] Munir, Rinaldi, "Matematika Diskrit - Graf (Bagian 1)," [Online]. Available: http://informatika.stei.itb.ac.id/~rinaldi.munir/. [Accessed: Jun. 18, 2025]

[8] Cleaning the Glass, "How do NBA defensive rotations work?," Jan. 5, 2021. [Online]. Available: https://cleaningtheglass.com/how-do-nba-defensive-rotations-work/. [Accessed: Jun. 18, 2025].

Ernest Clarence Gunawan 13525114

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 2025