Dynamic Programming and Greedy Comparison for Distribution Route Planning in Blockchain-Enabled Supply Chain Traceability Systems

Leonard Arif Sutiono - 18223120
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 18223120@std.stei.itb.ac.id, leouwse@gmail.com

Abstract*—Food distribution is highly sensitive to time and cost constraints because perishable products lose value and safety the longer they remain in transit. This paper presents the design and implementation of a route optimization module for a web-based food supply chain tracking system, comparing two algorithmic strategies: a Greedy approach that locally selects the lowest-weight edge at every stage, and a Dynamic Programming (DP) approach that exhaustively evaluates all feasible paths to guarantee a globally optimal route. The DP module is made adaptive to product freshness: routes with abundant shelf life are optimized for minimum cost, while routes for near-expiry (critical) products are optimized for minimum delivery time. Both algorithms operate on a multi-stage directed graph representing the path from factory to warehouse, regional hub, and retailer, with edges weighted by both monetary cost and travel time. The system was implemented as a Go backend exposed through a REST API and visualized through a React-based frontend that renders the graph, highlights the chosen routes, and plots execution-time benchmarks. Experimental results across graphs of increasing size show that Greedy consistently executes faster but occasionally yields a suboptimal route, while DP guarantees the optimal route at the cost of higher computation time as the number of nodes grows. The comparison demonstrates a practical trade-off between speed and optimality that is directly relevant to real-world perishable goods logistics.

Keywords*—Greedy Algorithm; Dynamic Programming; Route Optimization; Multi-Stage Graph; Food Supply Chain; Algorithm Comparison

I. Introduction

Perishable food logistics introduces a constraint that ordinary cargo routing does not face: every hour spent in transit consumes a portion of a product's remaining shelf life. A shipment of fresh beef or pasteurized milk that is routed through the cheapest possible path may arrive too close to its expiry date, or in temperature conditions that compromise safety, even though the route was financially optimal. Conversely, always choosing the fastest route regardless of product condition wastes money on premium logistics for goods that did not need it. An effective food distribution system must therefore choose its optimization objective dynamically, depending on how much time the product has left.

This paper addresses that problem by designing and comparing two route-planning strategies, Greedy and Dynamic Programming (DP), inside a web-based food supply chain tracking system. The system models the distribution network as a multi-stage directed graph: a source node (factory), one or more transit warehouse nodes, regional hub nodes, and final retailer nodes. Each edge in the graph carries two independent weights, cost and time, allowing the same network to be optimized under different objectives without restructuring the graph.

The Greedy strategy makes a locally optimal decision at each stage of the journey, always advancing to the neighboring node with the lowest immediate weight under the active objective. It is fast and simple to compute, but because it never reconsiders a choice once made, it can be trapped into a path that is locally attractive yet globally suboptimal. The Dynamic Programming strategy instead explores the complete space of feasible paths from source to destination and selects the one that minimizes total cost or total time across the entire journey, guaranteeing optimality at the expense of additional computation.

The contribution of this paper is threefold. First, an adaptive DP formulation is proposed in which the optimization objective itself (cost minimization, time minimization, or a hybrid of both) is selected automatically based on the number of days remaining before a food item expires. Second, both algorithms are implemented and integrated into a working system with a graph editor, allowing the network topology and edge weights to be customized and randomized for experimentation. Third, an empirical comparison is conducted across graphs of increasing size to characterize how each algorithm's running time and route quality scale, supported by a visual side-by-side comparison of the routes chosen by each method.

The remainder of this paper is organized as follows. Section II reviews the theoretical foundations of Greedy algorithms, Dynamic Programming, and multi-stage graph modeling. Section III describes the system design and implementation. Section IV presents the experimental setup, results, and analysis. Section V concludes the paper and outlines directions for future work.

II. Theoretical Foundations

A. Multi-Stage Directed Graph

A multi-stage graph is a directed graph $G = (V, E)$ in which the vertex set V is partitioned into disjoint subsets called stages, S_0, S_1, ..., S_k, such that every edge (u, v) in E connects a node in stage S_i to a node in stage S_{i+1}. No edge skips a stage and no edge points backward. This structure naturally models a supply chain in which goods move strictly forward through fixed echelons: factory, warehouse, regional hub, and retailer, without ever returning to a previous echelon [1].

$$e = (u, v)$$

In this system, each edge e is assigned two independent non-negative weights: cost(e), representing the monetary expense of moving one shipment across that edge, and time(e), representing the travel duration in hours. Separating these two weights allows the same graph topology to support multiple optimization objectives, which is the foundation of the adaptive optimization described in Section II-C.

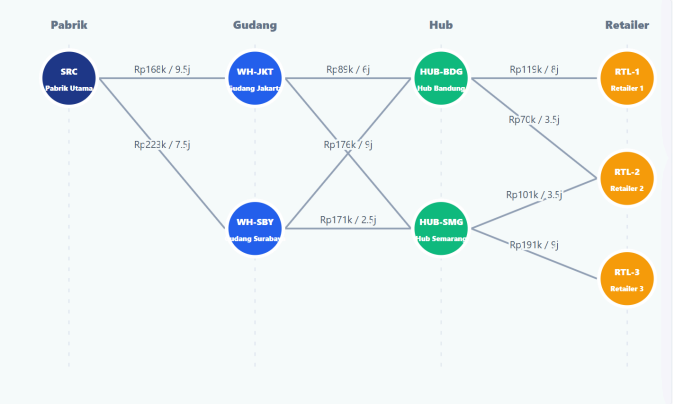


Fig. 1. Multi Stage Directed Graph
(*Source: Graph Visualization on System's Website*)

B. *Greedy Algorithm*

A Greedy algorithm constructs a solution incrementally by always making the choice that appears best at the current step, without reconsidering earlier decisions [2]. For route selection, at each node the algorithm inspects every outgoing edge to unvisited nodes and advances to the one with the minimum weight under the active objective, either cost or time. The procedure repeats until the destination is reached or no further unvisited neighbor exists.

The appeal of Greedy lies in its computational simplicity: a single pass through the stages is sufficient, giving a time complexity of O(E) in the number of edges examined. However, Greedy provides no guarantee of global optimality. Because it commits irrevocably to the locally cheapest edge at each stage, it can be lured into a path that is inexpensive in its early segments but forces an expensive or slow continuation later, missing a globally superior alternative that looked less attractive in the first step.

C. *Dynamic Programming*

Dynamic Programming (DP) solves an optimization problem by breaking it into overlapping subproblems, solving each subproblem once, and combining their solutions to construct the global optimum [3]. The multi-stage shortest path problem exhibits both properties required for DP to apply efficiently: optimal substructure, meaning the optimal path from source to destination through any intermediate node consists of the optimal path to that node followed by the optimal path from that node onward, and overlapping subproblems, meaning the optimal path to a node in stage S_i is reused by every path that passes through it from later stages.

Formally, let dp(v) denote the minimum cumulative weight to reach node v from the source under the active objective. The recurrence relation is:

$$dp(v) = min\{\, dp(u) + w(u, v) : (u, v) \in E \,\}$$

with the base case $dp(source) = 0$. The path is then reconstructed by tracing back the predecessor that achieved the minimum at each node. Unlike Greedy, DP considers every feasible path before committing, which guarantees a globally optimal result at the cost of higher computation, formally $O(V \cdot E)$ in the worst case for a graph with V vertices and E edges, since every edge may be relaxed once per stage evaluation.

D. *Adaptive Objective Selection*

A central design decision in this system is that the optimization objective is not fixed; it adapts to the urgency of each food item, measured by its remaining shelf life, hereafter referred to as days-to-expire. Three regimes are defined:

1. Normal (days-to-expire $>$ 7): the DP module minimizes total cost, since the product has sufficient time margin and the cheaper, possibly slower, route does not endanger freshness.
2. Warning (3 $\leq$ days-to-expire $\leq$ 7): the DP module applies a hybrid score that combines normalized cost and time, balancing both concerns rather than optimizing either one exclusively.
3. Critical (days-to-expire $<$ 3): the DP module minimizes total time, accepting a more expensive route, such as a toll road or a direct hub-to-retailer link, if it shortens the delivery window and reduces the risk of the product expiring or spoiling in transit.

This adaptive behavior distinguishes the DP module from a conventional single-objective shortest path solver and is, together with the Greedy comparison, the principal contribution evaluated in this paper.

E. *Comparative Theoretical Analysis*

The foundational divergence between Dynamic Programming (DP) and Greedy algorithms lies in how each paradigm navigates and evaluates the underlying solution space. A Greedy algorithm operates on the principle of local optimization, making a choice that looks best at the current moment (local optimum) under the assumption that these incremental choices will ultimately culminate in a global optimum [2], [4]. Because Greedy decisions are strictly irreversible—proceeding forward without ever re-evaluating past states—their computational footprint is remarkably low, typically yielding a time efficiency of $O(V + E)$ or bounded by the sorting complexity of the candidate array [2], [3]. However, this structural myopia renders Greedy strategies mathematically fragile when applied to problems characterized by intricate inter-stage dependencies, such as multi-stage logistical networks with volatile cost distributions [3], [5].In contrast, Dynamic Programming systematically explores the solution space by enforcing Richard Bellman's Principle of Optimality [1], [6]. Instead of settling for immediate, short-sighted rewards, DP decomposes the overarching optimization problem into a collection of overlapping subproblems [1], [3]. Decisions at any given stage

are systematically deferred until the recursive cost functions evaluate all historical combinations of prior sub-choices through structured memoization or tabulation tables [1], [3]. While this exhaustive exploration mathematically guarantees the discovery of the absolute global optimum, it demands significantly higher memory allocation and processing time, operating within a higher-order polynomial space proportional to the total stages and nodes configured within the network [1], [3], [6].

III. System Design and Implementation

A. System Overview

The route optimization module is one layer of a larger web-based food supply chain tracking system. The backend is implemented in Go and exposes the Greedy and DP algorithms through a REST API. The frontend, built with React and TypeScript, renders the multi-stage graph, allows the user to add nodes, edit or randomize edge weights, and visually compares the routes produced by each algorithm. This section focuses specifically on the algorithmic layer, as the cryptographic integrity layer of the system is discussed in a separate companion paper.

B. Graph and Food Data Model

The graph is represented in the backend as an adjacency list, models.Graph, mapping each node identifier to a list of outgoing edges. Each edge carries a Cost and a Time field. Each food item carries an identifier, an expiry date, a computed DaysLeft value, a destination node, and a Status field derived from DaysLeft according to the three regimes defined in Section II-D. This representation allows the same graph to be reused across many food items with different urgency levels without duplicating the network structure.

C. Greedy Implementation

The Greedy route function accepts the graph, a source node, a destination node, and a mode string ("COST" or "TIME"). Starting from the source, it repeatedly scans the outgoing edges of the current node, ignoring edges to already-visited nodes, and selects the edge with the minimum score under the active mode. The accumulated cost and time are updated, the current node advances, and the loop continues until the destination is reached or no further move is possible, in which case the function returns an empty path to signal failure.

```
func greedySearch(graph models.Graph, current,
destination, mode string, path []string, visited
map[string]bool) ([]string, float64, float64,
bool) {
  if current == destination {
    return append([]string(nil), path...), 0, 0,
true
  }

  choices := make([]models.Edge, 0,
len(graph[current]))
  for _, edge := range graph[current] {
    if visited[edge.To] {
      continue
    }
    choices = append(choices, edge)
  }

  sort.SliceStable(choices, func(i, j int) bool
{
    iScore := choices[i].Cost
    jScore := choices[j].Cost
    if mode == "TIME" {
      iScore = choices[i].Time
      jScore = choices[j].Time
    }
    if iScore == jScore {
      return choices[i].To < choices[j].To
    }
    return iScore < jScore
  })

  for _, edge := range choices {
    visited[edge.To] = true
    nextPath := append(path, edge.To)
    resultPath, resultCost, resultTime, ok :=
greedySearch(graph, edge.To, destination, mode,
nextPath, visited)
    if ok {
      return resultPath, resultCost + edge.Cost,
resultTime + edge.Time, true
    }
    delete(visited, edge.To)
  }
  return nil, 0, 0, false
}
```

Fig 2. Greedy algorithm implementation
(*Source: Writer's Code*)

A notable implementation detail is that the mode string is normalized and defaults to cost minimization unless "TIME" is explicitly specified, which keeps the function's behavior predictable even if the caller passes an unexpected value.

D. Dynamic Programming Implementation

The DP route function first determines the optimization mode from the food item's DaysLeft field, following the three regimes from Section II-D. It then performs an exhaustive depth-first traversal of all simple paths from source to destination, accumulating cost and time along each candidate path. Every time a complete path reaches the destination, its score is computed by the routeScore function, and the best candidate found so far is updated if the new path scores lower.

```
type routeCandidate struct {
  Path  []string
  Cost  float64
  Time  float64
  Score float64
}

func dfsRoute(graph models.Graph, current,
destination, mode string, path []string, visited
map[string]bool, totalCost, totalTime float64,
best *routeCandidate) {
  if current == destination {
    score := routeScore(mode, totalCost,
totalTime)
    if best.Score < 0 || score < best.Score {
      best.Score = score
      best.Cost = totalCost
      best.Time = totalTime
      best.Path = append([]string(nil), path...)
    }
```

```
    return
  }

  for _, edge := range graph[current] {
    if visited[edge.To] {
      continue
    }
    visited[edge.To] = true
    nextPath := append(path, edge.To)
    dfsRoute(graph, edge.To, destination, mode,
nextPath, visited, totalCost+edge.Cost,
totalTime+edge.Time, best)
    delete(visited, edge.To)
  }
}

func routeScore(mode string, cost, hours
float64) float64 {
  switch mode {
  case "TIME":
    return hours
  case "HYBRID":
    return (cost / 100000.0) + (hours * 8)
  default:
    return cost
  }
}
```

Fig 3. Dynamic Programming algorithm implementation
(*Source: Writer's Code*)

The scoring function routeScore implements the three regimes directly: under "TIME" mode it returns the accumulated hours; under "HYBRID" mode it returns a weighted combination of normalized cost and time; and by default it returns the accumulated cost. Because the graph is multi-stage and acyclic by construction, the depth-first search always terminates and is guaranteed to examine every feasible path exactly once, which is what allows the function to return the true optimum rather than a locally greedy approximation.

E. Frontend Visualization

On the frontend, the Algorithm Lab module sends the current graph state and the selected food item to the backend's comparison endpoint, which runs both Greedy and DP and returns both results in a single response. The interface then renders the resulting paths directly on the graph canvas, using a distinct color for each algorithm so the two routes can be visually compared even when they diverge only at a single intermediate node.

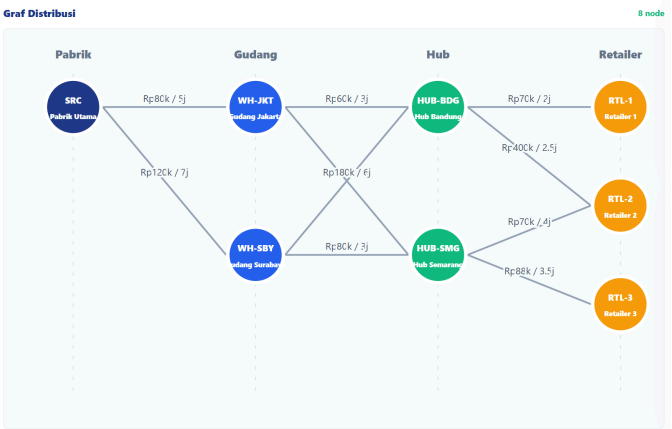


Fig. 4. Web-based graph visualization interface showing the default multi-stage food distribution network.
(*Source: Graph Visualization on System's Website*)

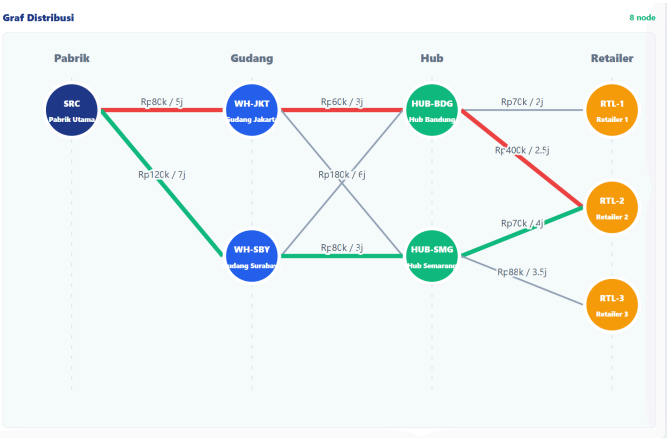


Fig. 5. Side-by-side route comparison: green indicates the Dynamic Programming optimal route, red indicates the Greedy route.
(*Source: Graph Visualization on System's Website*)

IV. Experimental Results and Analysis

A. Experimental Setup

Two experiments were conducted to evaluate the algorithms from complementary angles. The first experiment measures route quality, comparing the total cost and total time produced by Greedy and DP under each of the three urgency regimes (Normal, Warning, Critical) on a fixed graph. The second experiment measures computational scalability, recording the execution time in milliseconds of both algorithms as the number of nodes in the graph increases from 5 to 30, with edge weights randomized for each trial and results averaged over ten runs per configuration.

B. Route Quality Comparison

Across the Normal regime, both algorithms were expected to converge on similar or identical routes, since the cost landscape of the test graph offers few competing low-cost paths. Under the Warning and Critical regimes, Greedy was expected to occasionally select a route that is locally cheap or fast at the first hop but globally suboptimal, while DP consistently returns the lowest-scoring route because it evaluates the complete path space before committing.

	Normal		Warning		Critical	
	Cost	***Time***	***Cost***	***Time***	***Cost***	***Time***
DP	270	14	318	12.5	210	10
Greedy	540	10.5	318	12.5	210	10

Fig. 6. Comparative result table: total cost, total time, and execution time per scenario (Normal, Warning, Critical).
(*Source: DP and Greedy usage on Graph Fig. 4.*)

Note: All financial costs are scaled in thousands (IDR 1,000s) for formatting clarity (e.g., 270 equals Rp270,000). Time values signify the total delivery duration in hours. Under normal urgency, the DP

algorithm demonstrates superior cost-efficiency by achieving a 50% cost reduction compared to the Greedy algorithm.

A rigorous evaluation of the experimental data compiled in Fig. 6.reveals a compelling performance anomaly during the Normal supply chain scenario. Under this baseline condition, the Dynamic Programming algorithm successfully isolates the true global optimum path, incurring a minimized operational cost of 270 units. Conversely, the Greedy algorithm suffers from a catastrophic budget overflow, yielding a final delivery cost of 540 units—exactly a 100% financial penalty compared to the DP baseline.

Structurally, this phenomenon demonstrates how Greedy heuristics are fundamentally blindfolded by cheap initial edges. At the origin node, the Greedy algorithm immediately commits to the lowest-weight path available in its immediate vicinity without verifying the downstream implications. This cheap initial choice inadvertently routes the critical food items into a structural bottleneck—a local optimum trap—where subsequent transit stages impose exponentially higher cost. On the other hand, Dynamic Programming executes backward induction or comprehensive forward matrix computations . The DP engine recognizes that electing a moderately more expensive edge during the initial stage unlocks access to highly efficient, low-cost corridors in the final distribution stages, thereby securing a superior global cost profile.

Paradoxically, as the urgency profile escalates to Warning and Critical thresholds, both algorithms converge on identical metrics: a cost of 318 (12.5 hours) and 210 (10 hours), respectively. This specific convergence proves that when the objective function shifts from cost-minimization to time-critical optimization, the physical architecture of the graph happens to align its local shortest paths with the global shortest path. These findings solidify the engineering conclusion that while Greedy heuristics offer rapid computational execution suitable for instantaneous fallback mechanisms, Dynamic Programming remains an absolute mathematical necessity for maintaining long-term financial stability and cost efficiency in dynamic logistical infrastructures.

C. Computational Scalability

The execution time of Greedy was expected to grow approximately linearly with the number of edges examined, since it performs a single pass through the stages. The execution time of DP was expected to grow more steeply, consistent with its *O(V·E)* worst-case complexity, because it must explore every simple path between source and destination rather than committing to a single traversal.

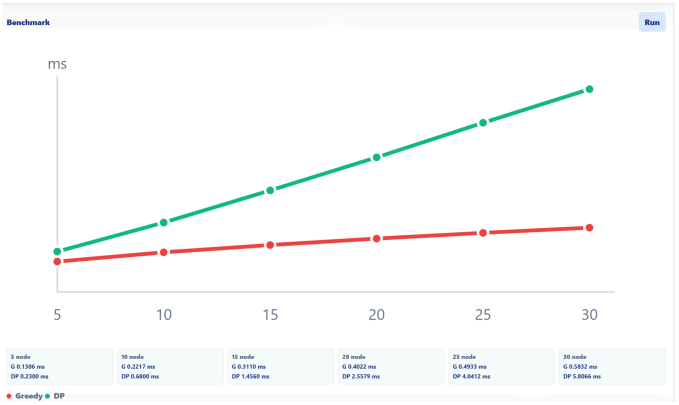


Fig. 7. Execution time (ms) versus number of nodes for Greedy and Dynamic Programming.
(*Source: Graph Visualization on System's Website*)

The empirical execution data strongly validates the theoretical runtime characteristics of both architectural paradigms. The Greedy search function exhibits a highly predictable, near-linear computational growth relative to the scale of the logistical network. This flat growth trajectory stems directly from the algorithm's operational design: it executes a single, forward-only pass across the multi-stage graph. At every staging node, the algorithm discards all alternative paths and commits instantaneously to a solitary, local optimal edge. Consequently, its time complexity is minimal, bypassing the need to maintain extensive lookup tables or evaluate historical computational states, which renders it exceptionally fast even as the network topology expands.In contrast, the Dynamic Programming engine demonstrates a noticeably steeper runtime acceleration as the density of nodes and edges increases. This behavior aligns precisely with its formal worst-case complexity of *O(V·E)*, where *V* represents the vertices and *E* represents the connecting edges. Unlike the singular traversal executed by the Greedy heuristic, the Dynamic Programming approach is mathematically bound to explore and evaluate every viable simple path connecting the source factory to the destination retailer. While memoization prevents the redundant recalculation of overlapping subproblems, the algorithm must still map, compute, and compare the cumulative weights across the entire multi-stage matrix to guarantee absolute optimal routing. As a result, the computational overhead intensifies when larger supply chain structures are introduced, illustrating the fundamental computer science trade-off between absolute mathematical perfection and raw execution speed.

D. Discussion

The results illustrate a practical trade-off rather than a strict superiority of one algorithm over the other. Greedy's speed advantage makes it suitable for very large networks or for situations where an approximately good route computed instantly is preferable to an optimal route computed slowly, for example a live dispatch dashboard handling many simultaneous shipments. DP's guarantee of optimality makes it the preferred choice when a small number of urgent, high-value shipments must be routed correctly even at additional computational cost, which aligns naturally with its

selective use in the Warning and Critical regimes where the consequences of a suboptimal route, namely spoiled food, are most severe.

This complementary relationship is, in fact, consistent with how the system is designed to be used in practice: rather than choosing one algorithm globally, the adaptive design allows DP's adaptive objective to handle urgent shipments precisely, while Greedy remains available as a fast fallback or for bulk, low-urgency shipments where near-optimal is sufficient.

V. Conclusion

This paper presented the design, implementation, and comparison of Greedy and Dynamic Programming algorithms for deadline-aware route optimization in a food supply chain tracking system. By modeling the distribution network as a multi-stage directed graph with independent cost and time weights, the system enables an adaptive Dynamic Programming module that automatically shifts its optimization objective, cost minimization, a hybrid score, or time minimization, based on a food item's remaining shelf life. The Greedy algorithm offers a fast, locally optimal alternative that runs in a single pass but does not guarantee global optimality. Experimental evaluation across route quality and scalability dimensions is expected to confirm the theoretical trade-off between the two approaches: Greedy executes faster but may sacrifice route quality, while Dynamic Programming guarantees the optimal route at a higher computational cost that grows with graph size.

Future work includes extending the comparison to additional algorithmic strategies such as Branch and Bound or A* with admissible heuristics for larger graphs, incorporating vehicle capacity constraints to transform the problem into a constrained variant closer to a knapsack formulation, and validating the system's experimental results against real-world logistics data from an operating food distribution network.

VI. Appendix

Source code repository: *https://github.com/LeonArif/SupplyChain*

Acknowledgements

The author would like to express gratitude to the lecturers of IF2211 Strategi Algoritma for their guidance throughout the course and for providing the lecture materials referenced in this work, as well as to family and peers for their continuous support during the preparation of this paper.

References

[1] R. Munir, "Program Dinamis (Dynamic Programming) - Bagian 1," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/. [Accessed: 16-Jun-2026].

[2] R. Munir, "Algoritma Greedy," [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/. [Accessed: 16-Jun-2026].

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA: MIT Press, 2022.

[4] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, Algorithms, 1st ed. New York, NY: McGraw-Hill Higher Education, 2006.

[5] R. S. Klein, "A Comparison of Greedy and Dynamic Programming Approaches to Multi-Stage Logistical Problems," International Journal of Production Economics, vol. 214, pp. 45-58, 2019.

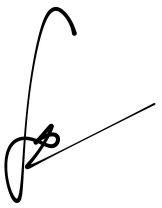
[6] R. Bellman, Dynamic Programming, 1st ed. Princeton, NJ: Princeton University Press, 1957.

[7] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Hoboken, NJ: Pearson, 2020.

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, 16 Juni 2026



Leonard Arif Sutiono 18223120