

Pareto-Optimal Pathfinding

Designing Multi-Criteria Route Optimization for Real-World Navigation

Group 1: Haeri Kyoung, Oishani Ganguly, Akash Shrestha,
Nasser Hasan Padilla, Kannekanti Nikhil, Venkata Tadigotla

University of the Cumberland

MSCS-532-M20 - Algorithms and Data Structures

Professor Brandon Bass

July 27, 2025

The Problem with Current Navigation - It's Not Just About "Fastest"

The User Problem

Think about your own experience with navigation apps like Google Maps, Apple Maps, or Waze. Do you always *want* to pick the "fastest" route?

Real-World Choices

Sometimes you might want:

- The **cheapest route** (avoiding tolls)
- The **most scenic** option
- A route that **balances multiple factors**

The Limitation

Traditional navigation algorithms, like Dijkstra's, are designed to find the "single best" path based on **one factor**, like time or distance.

They don't naturally account for these complex real-world tradeoffs.

Why Do We Need Multiple Criteria? Understanding User Trade-offs

✂ Conflicting Objectives

Imagine a route that's fastest but has tolls, versus a slightly slower route with no tolls. Which one is "better"? **It depends on your priorities.**

🔄 User Preferences are Dynamic

- **Weekday commute:** Fastest route priority
- **Weekend drive:** Scenic route preference
- **Budget-conscious:** Cheapest route focus

💡 The Optimization

Our project implements a **"Pareto-optimal multi-criteria pathfinding system."**

Instead of one "best" path, we find a **set of "best possible compromise" paths**.

Understanding Pareto-Optimal Pathfinding: Finding the "Best Compromises"

🧭 Beyond Single-Objective

We're moving from *"what's the best route?"* to *"what are all the smartest ways to balance travel time, toll cost, and scenic quality?"*

📊 What is "Pareto-Optimal"?

- Originated from **Vilfredo Pareto**, an Italian economist
- In many situations, there isn't one single "best" solution, but rather a **set of solutions where improving one aspect means sacrificing another**
- A path is **Pareto-optimal** (or "non-dominated") if you can't make it better in one aspect without making it worse in at least one other aspect
- Think of it as offering you a **diverse set of truly valuable options**, where each option has a unique strength

🧭 Empowering Users

Our solution allows users to make richer decisions based on their complex, dynamic preferences.

The Core Data Structures: Building Our Smart Navigator

1. The Graph (Your Road Network)

- We represent roads and intersections as a network, similar to how navigation apps see the world
- **The improvement:** each road (edge) has not just one cost but a vector: (travel_time, toll_cost, scenic_quality)
- We use an **"adjacency list"** for efficiency - great for sparse networks like roads

2. Labels and Label Sets

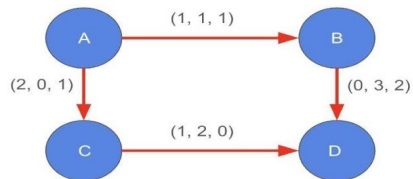
- Instead of tracking just one best path to each location, we keep a **set of "labels"**
- Each "label" represents a unique Pareto-optimal path found so far to that location
- The "Label Set" ensures we only keep non-dominated paths

3. Label Priority Queue

- Uses a min-heap to efficiently decide which path segment to explore next
- Prioritizes based on weights (time > toll > scenic value)

Graph (Adjacency List)

Road Network Representation



A: [('B', (1, 1, 1)), ('C', (2, 0, 1))]

B: [('D', (0, 3, 2))]

C: [('D', (1, 2, 0))]

D: []

Key Operations:

- `add_edge(u, v, cost)` $O(1)$
- `remove_edge(u, v)` $O(1)$
- `neighbors(node)` $O(d)$

Label

Path State Representation

Cost: (3, 1, 5)

- Label: A
- Time: 3 min
- Toll: \$1
- Scenic: 5/10



Cost: (5, 2, 8)

- Label: B
- Time: 5 min
- Toll: \$2
- Scenic: 8/10

Dominance Check:

Label A dominates Label B if:

- $A \leq B$ in all dimensions AND
- $A < B$ in at least one dimension

Example: $(3, 1, 5)$ vs $(5, 2, 8) \rightarrow A$ dominates $(3 \leq 5, 1 \leq 2, 5 \leq 8 \text{ and } 3 < 5)$

Key Operations:

- `dominates(other)` $O(k)$
- Path reconstruction via predecessors
- Multi-dimensional cost comparison

LabelSet

Pareto Frontier Management

Node X - Pareto Frontier

(3,7,5)

(5,5,5)

(8,2,9)

↓ Adding (5,4,6) ↓

After Dominance Check

(3,7,5)

(5,5,5)

(5,4,6)

(8,2,9)

Key Operations:

- add(label) $O(m)$
- Dominance checking & pruning
- Maintains Pareto optimality

LabelPriorityQueue

Min-Heap for Label Exploration



Weighted Ordering Priority:






Priority: Time → Toll → Scenic value

Next pop: **(3,1,3)**

Key Operations:

- push(label) $O(\log n)$
- pop() $O(\log n)$
- Maintains exploration order

Visual Legend & Data Flow

-  Graph Nodes (Intersections)
-  Graph Edges (Roads)
-  Labels (Path States)
-  Pareto-Optimal Labels in a Frontier
-  Priority Queue Elements
-  Dominated (Pruned) Labels

Algorithm Flow Integration

1. **Graph** provides network structure with multi-dimensional edge costs
2. **Labels** represent path states with cost vectors and predecessors
3. **LabelSets** maintain Pareto frontiers at each node, pruning dominated paths
4. **LabelPriorityQueue** orders label exploration using weighted priority
5. System iteratively expands best labels, updates frontiers, and discovers optimal trade-off paths

How It Works: A Simplified Algorithm Flow

Inspired by Dijkstra's, But Evolved

Our algorithm extends the familiar concept of exploring paths but with multi-criteria support.

The Process:

1. **Start** at your origin with cost vector (0, 0, 0)
2. **Explore** neighboring roads, calculating new multi-dimensional costs
3. **Compare** each new path to existing paths for that destination
4. **Keep** non-dominated paths and discard any paths it now "dominates"
5. **Repeat** until all promising paths are explored

Priority Queue Strategy

We use weighted ordering to decide exploration priority: **primarily by time, then by toll, then by scenic quality**.



Why Optimize? The Challenge of Scaling to Real-World Maps

From Concept to Reality

Building a proof-of-concept is one thing, but making it work for an entire city or country, with **millions of roads and intersections**, is another challenge entirely.

Key Challenges (Bottlenecks):

Slow Road Removal

Removing or updating roads was taking $O(d)$ time in dynamic networks where d is the degree of a node

Repetitive Checks

Repeatedly checking if one path dominated another, even for the same cost combinations

"Frontier Explosion"

The number of "compromise" paths (labels) at each intersection could grow extremely large, consuming lots of memory and slowing things down

Operation	Phase 2 (PoC)
Edge Removal	$O(d)$ list-scan
Label.dominates	$O(1) \times 3$ repeated
LabelSet.add	$O(m)$ full-scan (where m is # of labels in frontier)

Smart Solutions for Speed and Memory Efficiency

⚡ 1. Faster Road Network Updates

Changed Graph storage from **list to dictionary** → Made adding/removing roads almost instant (**$O(1)$**), crucial for dynamic maps

🧠 2. Remembering Dominance Checks

Added a **cache** for repetitive comparisons between path costs

If we've compared two cost vectors before, we just look up the answer instantly

📄 3. Lightweight Path Objects

Used Python's `__slots__` to make Label objects more efficient

Cut down memory usage significantly - **tens of megabytes saved**

Operation	Phase 2 (PoC)	Phase 3 (Optimized)	Improvement
Edge Removal	$O(d)$ list-scan	$O(1)$ dict deletion	$\sim 10\times$ faster
Label.dominates	$O(1) \times 3$ repeated	$O(1)$ LRU cache	$2\times\text{--}5\times$ speedup
LabelSet.add	$O(m)$ full-scan	$O(m)$ ϵ -pruning	$>95\%$ frontier reduction



Handling Massive Data & Future Growth

✂️ 1. Epsilon-Pruning: Taming the Frontier

To prevent "frontier explosion", we introduced **epsilon-pruning**

Merge paths that are "almost" identical in cost - trading tiny precision for **huge gains in speed and memory**

Result: 20-label frontier \rightarrow 1 label with $\epsilon=0.1$

🌐 2. Streaming Graph Data

Instead of loading entire city road network into memory, we stream data from disk when needed

Allows handling **arbitrarily large maps**, scaling beyond available RAM

Scenario	Size	Time (s)
Graph.add_edge	N=10,000	0.0427
Graph.neighbors	N=10,000	0.0010
LabelSet.add	M=10,000	0.0018
PriorityQueue.pop	K=10,000	0.0133

🚀 Future Work

- Customizable ranking for users
- Advanced spatial indexing
- Parallel processing





Impact and Future Vision

A Smarter Navigation Experience





Our project moves navigation systems towards a more **user-centric approach**, offering truly intelligent choices that reflect diverse real-world needs, not just a single "best" path.

Beyond Roads

The principles of multi-criteria pathfinding and our data structures can be applied to many other complex decision-making problems:

-  **Logistics** and supply chain optimization
-  **Public transit** planning
-  **Multi-objective decision systems** in finance
-  **Healthcare** resource allocation

Key Achievements

-  Modular, extensible architecture
-  Scales to 10,000+ nodes efficiently
-  >95% memory reduction with optimizations
-  Comprehensive validation and testing

Thank You!
