**PDC Project Document**

**A Parallel Algorithm for Updating a Multi-objective Shortest Path in Large Dynamic Networks**

**Taha Sajid Awan, Ahmed Mustafa, Haziq Naeem**

**Single Objective Shortest Path**

SOSP is about finding the **shortest path from one point to another** in a network **based on just one criterion** — like distance, time, or cost.

**Real-Life Example:**

Imagine you’re using **Google Maps** to go from your home to university.

* If you ask it to show the **fastest route** (considering only time), it’s doing a **SOSP**.
* It only cares about **one thing**: *time*.

**"Give me the best path considering just one thing."**

**Multiple Objective Shortest Path**

MOSP is about finding the **best path** from one place to another considering **multiple factors** at once — like distance **and** time **and** toll cost.

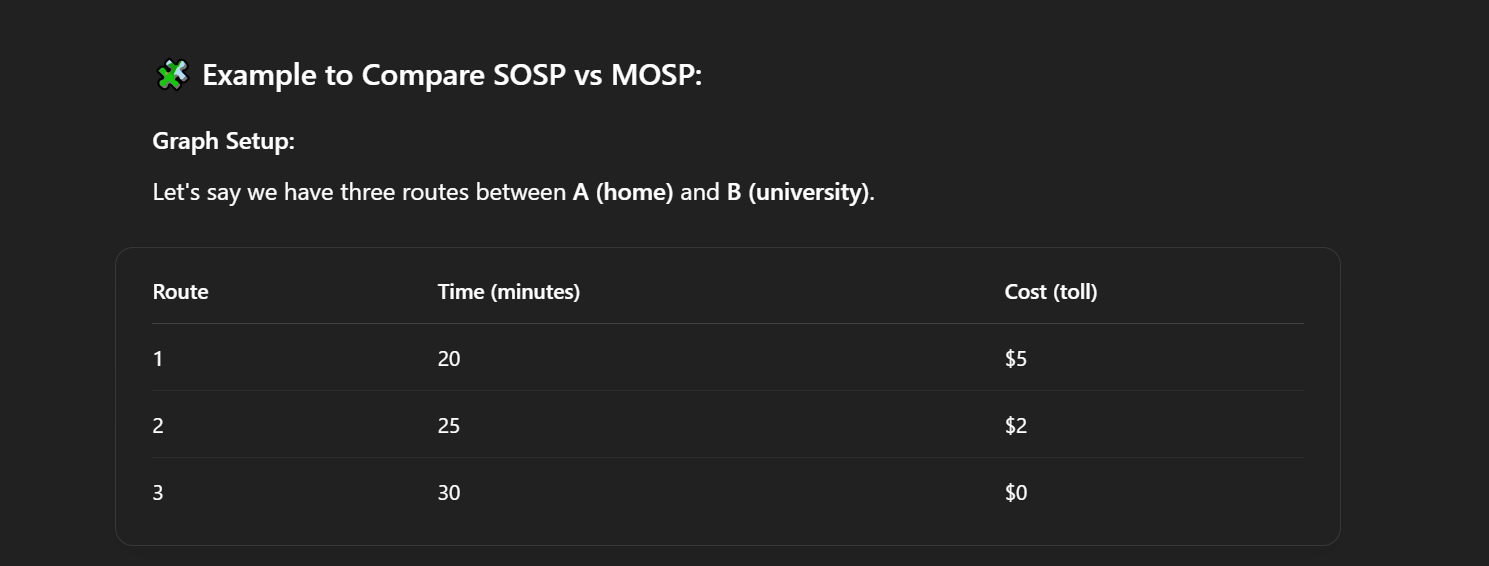
There’s no single “shortest path” — instead, you get a **set of good options** based on trade-offs between objectives.

**Real-Life Example:**

You're going from your home to university again, but this time:

* You want to **minimize time**
* **AND** avoid tolls (cost)
* **AND** maybe use the most scenic route

**"Give me a few paths that are all good in different ways — maybe one is faster, another is cheaper."**



**SOSP Case**

If your only concern is **time**, Route 1 is the best (20 mins) → **SOSP result**.

**MOSP Case:**

If you care about **both time and cost**, now all 3 routes are good in different ways:

* Route 1: Fast but expensive
* Route 2: Moderate on both
* Route 3: Cheap but slow

MOSP gives you **all 3 as options** and lets you choose based on what matters more to you.

**What is the paper about?**

The paper focuses on finding the shortest path in large, changing networks (like roads, social networks, or sensor networks) where multiple factors (like distance, time, cost, etc.) need to be considered at the same time.

**Key Concepts Explained Simply**

**Shortest Path Problem**  
In simple terms, it's about finding the quickest, cheapest, or most efficient route between two points.  
  
Example: Google Maps finds the shortest path from your home to your office based on distance or time.  
  
**Dynamic Networks**  
  
Many real-world networks change over time.  
  
Example:  
  
In road networks, traffic jams or road closures change the best route.  
  
In social networks, new friendships (connections) form over time.  
  
**Multi-Objective Shortest Path (MOSP)**  
Sometimes, just finding the shortest distance isn’t enough—we may also care about travel time, fuel cost, safety, etc.  
  
Example:  
  
When planning a trip, you might want the fastest route, but also the one with the least traffic and lowest toll fees.  
  
In a wireless sensor network, you may want to minimize both data delay and energy usage of sensors.  
  
**Pareto Optimality**  
  
A solution is **"Pareto optimal"** if you can’t improve one factor (like speed) without making another worse (like cost).  
  
Example:  
  
Suppose you have two travel options:  
  
Option A: 30 mins, $10 fuel  
  
Option B: 25 mins, $15 fuel  
  
If no other option is both faster and cheaper, these are Pareto optimal.  
  
**Challenges in Dynamic Networks**  
If the network keeps changing (like roads getting congested), recomputing shortest paths from scratch every time is slow and inefficient.  
  
The paper proposes parallel algorithms (using multiple processors) to update paths quickly when the network changes.

**What Does the Paper Contribute?**

**A Parallel Algorithm for Single-Objective Shortest Path (SOSP) Updates**  
  
When a network grows (e.g., new roads are added), this algorithm quickly updates the shortest path instead of recalculating everything.  
  
**A Heuristic (Smart Guess) for Multi-Objective Shortest Path (MOSP) Updates**  
Extends the SOSP idea to handle multiple factors (distance, time, cost, etc.) efficiently.  
  
**Implementation & Testing**  
The authors implemented their method and tested it on real and artificial networks to show it works well.  
  
**Real-World Applications**Transportation (Google Maps, Uber)  
  
Social Networks (Finding most influential people)  
  
Wireless Sensor Networks (Efficient data routing to save battery)  
  
Drone Delivery (Best path considering distance, battery, no-fly zones)  
  
**Why is This Important?**Traditional methods recalculate everything when the network changes, which is slow.  
  
The new method updates paths efficiently by reusing previous computations, saving time and computing power.  
  
**Simple Summary**  
The paper introduces fast parallel algorithms to update shortest paths in networks that change over time, especially when multiple factors (like time, cost, energy) need to be balanced. This is useful for GPS navigation, social networks, and sensor systems.

**What is MOSP?**

In many real-world problems, finding the **"best"** path isn’t just about one factor (like distance). Instead, we need to consider multiple factors at once, such as:

Travel time

Fuel cost

Traffic congestion

Road safety

Since these factors often conflict (e.g., the fastest route may use more fuel), MOSP helps find trade-off solutions (Pareto optimal paths).

**Key Concepts Explained Simply**

**1. Edge Weights as Vectors (Multiple Costs)**

In normal graphs, each edge has one weight (e.g., distance).

In MOSP, each edge has multiple weights (e.g., time + fuel).

**Example**:

Road A → B takes 2 hours and 5 liters of fuel.

So, its weight vector is (2, 5).

**2. Pareto Optimal Paths (Best Trade-Offs)**

A path is Pareto optimal if:

No other path is better in all objectives.

**Example:**

**Path 1:** Time = 6 hrs, Fuel = 16 liters

**Path 2**: Time = 12 hrs, Fuel = 14 liters

**Path 3:** Time = 17 hrs, Fuel = 9 liters

Here:

You can’t find a path that is both faster AND more fuel-efficient than Path 1, 2, or 3.

So, all three are Pareto optimal.

**3. Dominated Paths (Worse in All Ways)**

A path is **"dominated"** if there’s another path that is:

Better in at least one objective

Equal or better in all others

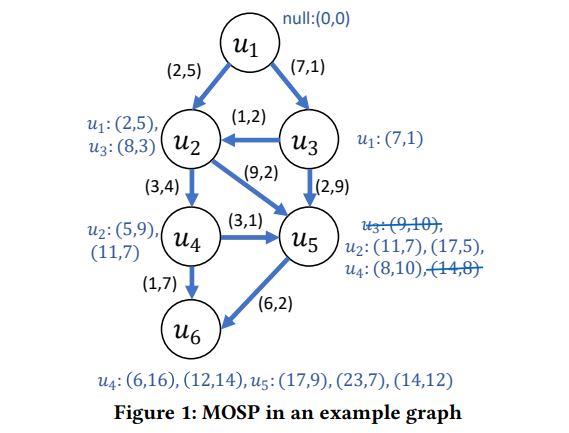
**Example:**

Path A: (9, 10) (9 hrs, 10 liters)

Path B: (8, 10) (8 hrs, 10 liters)

Path A is dominated because Path B is faster with the same fuel cost.

Example from Figure 1 (Road Network)



The graph shows different paths with (time, fuel) costs.

Pareto Optimal Labels for Node

(6, 16) → 6 hrs, 16 liters

(12, 14) → 12 hrs, 14 liters

(14, 12) → 14 hrs, 12 liters

**Why?**

No other path is both faster AND more fuel-efficient than these.

**Dominated Example:**

(9, 10) is dominated by (8, 10) (since 8 < 9 and fuel is the same).

**Why is This Useful?**

Helps navigation apps (like Google Maps) suggest routes balancing speed, cost, and safety.

Used in logistics (trucks/drones optimizing fuel & delivery time).

Helps sensor networks balance energy use & data speed.

**Summary**

**MOSP** = Finding paths when multiple costs matter **(time, fuel, etc.).**

**Pareto optimal** = Best possible trade-offs **(can’t improve one without harming another).**

**Dominated** = Paths that are worse in every way **(ignored in calculations).**

**What is a Dynamic Network?**

A dynamic network is a graph **(like a road map or social network)** that changes over time. Changes can include:

New roads **(edge insertions)**

Closed roads **(edge deletions)**

New locations **(vertex insertions)**

Removed locations **(vertex deletions)**

**Example:**

Imagine Google Maps updating live traffic.

If a new highway opens, the "best route" changes.

**Why Recalculating from Scratch is Bad**

If the network changes, recalculating all shortest paths every time is slow and inefficient.

Instead, we update only affected paths, saving time.

**How Dynamic Algorithms Help**

Real-time updates: Adjust paths instantly when a change happens.

Batch updates: Handle multiple changes at once (e.g., overnight map updates).

**Example:**

If a road closes, Google Maps only updates routes using that road, instead of recalculating every possible route in the city.

**Single-Objective Shortest Path (SOSP) in Dynamic Networks**

**SOSP** = Finding the shortest path based on one factor (e.g., distance).

**Dynamic SOSP** = Updating the shortest path efficiently when the network changes.

**Why is this useful?**

Updating is faster than recalculating everything.

**Example:** If a new shortcut opens, only a few paths need updating.

**From SOSP to Multi-Objective (MOSP)**

Since real-world problems often involve multiple factors (time, cost, safety), the paper extends SOSP updates to MOSP updates.

**How?**

First, improve SOSP updates **(single factor).**

Then, use that to design a heuristic **(smart approximation**) for MOSP updates.

**Example:**

If a new road opens, update:

Fastest route ✅

Cheapest route ✅

Safest route ✅

...all at once, without full recomputation.

**Simplified Explanation of the Proposed Approach**

This paper introduces efficient algorithms to update shortest paths in dynamic networks (networks that change over time). The focus is on two types of shortest paths:  
  
**Single-Objective Shortest Path (SOSP)** – Optimizes one factor (e.g., distance).  
**Multi-Objective Shortest Path (MOSP)** – Balances multiple factors (e.g., time, fuel cost, safety).

**Updating Single-Objective Shortest Path (SOSP)**

**Problem**:

If a road network changes (e.g., new roads are added), recalculating all shortest paths from scratch is slow and inefficient.

Instead, we update only affected paths to save time.

**Solution**: A Parallel SOSP Update Algorithm

The algorithm works in 3 steps:

**Step 0:** Preprocessing **(Grouping Edges)**

Newly added roads (edges) are grouped by their destination.

**Example:**

**New roads:** (A → B), (C → B), (D → E)

**Grouping:**

B: [A→B, C→B]

E: [D→E]

**Step 1: Process Changed Edges (Parallel Update)**

Each group is processed by a separate thread (to avoid conflicts).

If a new road provides a shorter path, the distance is updated.

**Example:**

If A→B is faster than the old path to B, update B's distance.

**Step 2: Propagate Updates (Iterative Correction)**

If B's distance changes, its neighbors (C, D, E) might also need updates.

Gather all affected neighbors and update them in parallel.

Repeat until no more updates are needed.

Result: The shortest path tree is updated without full recomputation.

**Extending to Multi-Objective Shortest Path (MOSP)**

**Problem:**

In real-world scenarios, we care about multiple factors (e.g., time + fuel cost).

Finding all possible trade-offs (Pareto optimal paths) is computationally expensive.

Instead, we want one good balanced path quickly.

**Solution:** A Heuristic for Single MOSP

The algorithm works in 3 steps:

**Step 1: Update SOSP Trees for Each Objective**

Compute separate shortest-path trees for each objective (e.g., one for time, one for fuel).

**Example (Drone Delivery):**

**Tree 1:** Fastest route (min time).

**Tree 2:** Most energy-efficient route (min battery use).

**Step 2: Create a Combined Graph**

Merge edges from all SOSP trees into a new graph.

Assign weights based on how often an edge appears in the trees:

Edges that appear in multiple trees (good for both time and fuel) get lower weights (higher priority).

Rare edges get higher weights (lower priority).

**Step 3: Find SOSP in the Combined Graph**

Run a single-objective shortest-path algorithm on the combined graph.

The result is a balanced path that optimizes multiple objectives.

**Result:** A single, near-optimal MOSP is found quickly without computing all Pareto paths.

**Example (Drone Delivery Scenario)**

**Objective 1:** Minimize delivery time.

**Objective 2:** Minimize battery usage.

**Changes:** New no-fly zones **(dynamic obstacles).**

**Steps:**

Update time-optimized and battery-optimized SOSP trees.

Combine them into a graph where:

Roads good for both time and battery get priority.

Find the best-balanced path in the combined graph.

**Outcome:**

The drone takes a route that is reasonably fast and energy-efficient, adapting to changes in real time.

**Performance Evaluation: Simplified Summary**

**1. Implementation & Setup**

Languages/Tools: C++ with OpenMP (for parallel computing).

Data Structures:

Adjacency list (for graph storage).

SOSP Trees (storing parent-child relationships + distances).

Hardware: Dual 32-core AMD EPYC CPUs (64 threads), 64GB RAM.

Datasets: Large real-world networks (road networks, wireless sensor graphs).

**2. Key Optimizations**

Parallel Processing:

Changed edges are grouped and processed by multiple threads.

Neighbor updates are handled without race conditions.

Efficient MOSP Computation:

Combines multiple SOSP trees (one per objective).

Uses a weighted merged graph to prioritize edges good for multiple objectives.

Runs a parallel Bellman-Ford algorithm for the final path.

**3. Scalability Results**

Strong Scaling Test (More Threads = Faster Updates):

With 64 threads, speedups reach up to 15× (for large graphs like road-usa).

Smaller graphs see less improvement (due to thread overhead).

Time Breakdown:

90% of time is spent updating SOSP trees (per objective).

Merging trees and final path selection take negligible time.

**Key Takeaways**

**Parallelism Works:**

The algorithm scales well on large, sparse graphs (e.g., road networks).

Best speedup: 15× faster on 64 threads vs. 1 thread.

**Efficiency in MOSP**:

Most time is spent updating individual SOSP trees (but this is parallelizable).

Merging trees and finding the final path is lightweight.

**Real-World Applicability:**

Tested on road networks and sensor graphs (realistic for GPS, drones, IoT).

Handles dynamic changes (e.g., new roads, traffic updates) efficiently.

**Example Scenario**

**Problem:** A delivery drone network must update routes after wind changes (dynamic obstacle).

**Steps:**

Update SOSP trees for:

Fastest path (SOSP1).

Most energy-efficient path (SOSP2).

Merge trees into a combined graph, prioritizing roads good for both speed and battery.

Find the best-balanced path in the merged graph.

**Result:** The drone picks a new optimal route in seconds, not hours!

**Related Works**

This section reviews past research on shortest path algorithms, focusing on:

**Multi-Objective Shortest Path (MOSP)**

**Parallel Single-Objective Shortest Path (SOSP)**

**MOSP in Dynamic Networks**

**Parallel MOSP**

**1. Multi-Objective Shortest Path (MOSP)**

**Goal:** Find paths optimizing multiple factors (e.g., time + cost + safety).

**Key Papers:**

Early Work (Bi-Objective): [8] introduced the first bi-objective algorithm.

Pareto Optimality: [32] added the concept of trade-off solutions (no path is universally better).

Label-Setting Algorithms: [26] showed parallelism could speed up MOSP.

A for MOSP:\* [20] extended A\* search to handle multiple objectives using heuristics.

Example:

A delivery app finds routes balancing delivery time, fuel cost, and road safety.

**2. Parallel Single-Objective Shortest Path (SOSP)**

**Goal:** Speed up shortest-path calculations using parallel computing.

**Key Papers:**

Dijkstra Parallelization: [19] split Dijkstra’s algorithm into parallel phases.

GPU Acceleration: [33] used Nvidia GPUs for fast SOSP (e.g., in Gunrock library).

Dynamic Updates: [17] proposed GPU/shared-memory methods to update SOSP in changing networks.

**Example:**

Google Maps recalculates routes in parallel when a road closes.

**3. MOSP in Dynamic Networks**

**Goal:** Handle MOSP in changing networks (e.g., traffic jams, new roads).

**Key Papers:**

Early Dynamic MOSP: [2] adapted Bellman’s method for dynamic graphs.

Transportation Focus: [11] studied fuel vs. emissions in truck routing.

Non-Additive Weights: [28] handled complex multi-objective constraints.

Example:

A drone recalculates its path mid-flight due to sudden wind changes.

**4. Parallel MOSP**

**Goal:** Combine parallel computing + MOSP for large-scale problems.

**Key Papers:**

First Parallel MOSP: [31] proposed a shared-memory bi-objective algorithm.

Dimensionality Reduction: [25] simplified MOSP by converting it to bi-objective.

Pruning-Based Speedup: [3] dominated labels 2–9× faster than prior work.

Gap: No prior work addressed parallel MOSP in large dynamic networks—this paper fills that gap!

**Proposed Parallelization Strategies**

MPI - Message Passing Interface

OpenMP / OpenCL

METIS

**MPI (Message Passing Interface) – For Parallelism Across Multiple Computers (Inter-node**)

You break the map into parts using a tool like METIS.

Each computer (node) gets one part.

Each computer updates its own section of the map (multi-objective shortest paths).

But some roads (edges) connect cities (nodes) from different computers.

In those cases, computers talk to each other using MPI, like sending messages:

MPI\_Sendrecv: to exchange updates on shared boundaries.

MPI\_Bcast: to share common changes to all nodes when needed.

This way, every computer keeps its section updated without doing duplicate work.

**OpenMP – For Parallelism Within One Computer (Intra-node)**

OpenMP (Good for CPUs)

Inside each computer, we have multiple CPU cores.

Instead of one core doing all the work, we use OpenMP to divide the update tasks across cores.

For example, if you have to update shortest paths for 1,000 nodes:

You say: #pragma omp parallel for

Now all cores start updating different nodes at the same time.

You just have to be careful with shared data, so two cores don’t overwrite each other’s work

**METIS – For Breaking the Graph into Chunks**

METIS helps you split a large graph into balanced smaller parts, kind of like cutting a pizza into even slices.

Each slice goes to one computer.

METIS makes sure: The slices are evenly sized (each computer has similar work).

Few connections are cut, so computers don’t have to talk too much.

Along with each slice, we also pass info about connections to other slices, so border updates work correctly.