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

Athaar Naqvi 22i-1166
Ambreen Arshad 22i-1308
Fiza Wajid 22i-0809

1.Introduction

- **Overview & Motivation:**
  Computing shortest paths is essential in domains like transportation, social networks, and drone delivery. However, it becomes significantly more complex in large-scale, dynamic networks with multiple optimization objectives such as time, distance, and energy.
- **Key Challenges:**
  Dynamic networks—like wireless sensor networks (WSNs)—undergo frequent topology changes, making traditional recomputation approaches inefficient. Real-time, adaptive algorithms are necessary to cope with such changes.
- **Need for Multi-Objective Solutions:**
  In scenarios like WSNs, trade-offs between competing metrics (e.g., latency vs. energy) are common. This calls for Pareto-optimal solutions, where improving one objective doesn't degrade another.
- **Research Gap:**
  Current parallel multi-objective shortest path (MOSP) algorithms are ill-suited for dynamic environments. They often struggle with real-time adaptability, suffer from irregular memory access patterns, and face challenges with load balancing.
- **Contributions:**

1. A parallel single-objective shortest path (SOSP) update algorithm leveraging grouping techniques to reduce redundancy.
2. A heuristic-based MOSP update strategy designed for growing dynamic networks.
3. A scalable shared-memory implementation tested on both real-world and synthetic datasets

2. PRELIMINARIES

Basic Definitions:

- Graph G(V,E)G(V, E)G(V,E), with non-negative edge weights.
- **SOSP**: Single-objective shortest path from source to all destinations.
- **SOSP Tree**: Contains only edges in shortest paths.

2.1. Multi-objective Shortest Path (MOSP):

- Edge weights become vectors for kkk objectives: W(e)=(w1,...,wk)W(e) = (w_1, ..., w_k)W(e)=(w1,...,wk).

- **Pareto optimal path**: Not dominated by any other path.
  Pareto Optimality:

1. Each path label is a set of distance vectors representing multiple Pareto paths.
2. Example: A vertex can have several Pareto optimal paths with different trade-offs (e.g., time vs. fuel).

   Dominated Distance:

- A path is **dominated** if another path is strictly better in at least one objective and no worse in others.
- Dominated paths are excluded from the optimal set.

2.2. Dynamic Networks:

- Graphs change over time via edge insertions/deletions.
- Dynamic graph algorithms avoid full recomputation by **updating affected parts**.
- Prior studies show **SOSP update** is more efficient than recomputation.
- This paper builds upon that to support **MOSP updates**.

3.Proposed Approach

3.1 SOSP Update (Single-Objective Shortest Path)

This method efficiently updates shortest path labels in a directed graph when new edges are inserted, avoiding full recomputation.

Let $G^t(V^t, E^t)$ be the graph at time t and L^t the distance labels. After inserting new edges $Ins^t = E^{t+1} - E^t$, the graph becomes $G^{t+1}(V^{t+1}, E^{t+1})$.

Algorithm Overview

Inputs:

- Graph G(V, E)
- SOSP tree T = {(v, δ), Parent[v]}
- Inserted edges Ins

Step 0: Preprocessing
Group inserted edges (u, v) by destination v for efficient parallel processing.

Step 1: Process Changed Edges
Each group is handled by a thread. If a new edge gives a shorter path to v, update δ[v] and mark v as affected.

Step 2: Propagate Updates
Neighbors of affected vertices are checked. If their distances improve, they're marked and the process continues iteratively until no more updates.

Key Features:

- Thread-safe via vertex grouping
- Minimizes recomputation
- Efficient parallel updates
- Ensures correctness through iterative propagation

3.2 MOSP Update (Multi-Objective Shortest Path)

In dynamic networks (e.g., drone systems), finding all optimal paths for multiple objectives is slow. This approach finds one balanced path quickly.

Step 1: Update SOSP Trees

Each objective (e.g., time, energy) has its own SOSP tree. All are updated using the SOSP update method.

Step 2: Combine Trees

Build a new graph by merging updated trees. Edges are weighted based on how often they appear:

- More trees → lower weight (preferred)
- Fewer trees → higher weight

This allows balancing or prioritizing objectives.

Step 3: Find Single Path

Run a shortest path algorithm on the combined graph. Re-evaluate the path using original weights to ensure it's a good multi-objective solution.

Real-World Example

A drone has two path options:

- T_f: fastest
- T_e: most energy-efficient

Depending on the energy budget or external factors, the system picks or combines paths for the best result.

Summary

- Efficiently updates paths after edge changes

- Avoids full recomputation
- Balances multiple objectives
- Ideal for real-time dynamic systems

4. Performance Evaluation

The proposed algorithms were implemented in C++ with OpenMP for shared-memory parallel computing. The performance evaluation primarily focuses on how the Single Objective Shortest Path (SOSP) update algorithm and the Multi-Objective Shortest Path (MOSP) update heuristic scale and perform on large, dynamic networks.

Key Implementation Aspects:

- **Data Structures:** Graphs are stored using adjacency lists; updated edges include metadata like weights and update status.
- **Parallelism:**
    - Edge-centric and vertex-centric approaches were used.
    - Threads are assigned per vertex or group of edges to minimize race conditions.
- **Heuristics for MOSP:** Convert multi-objective updates into a sequence of SOSP updates for efficiency.

Experimental Setup:

- Platform: Dual 32-core AMD EPYC Rome CPUs with 64 GB RAM.
- Networks: Tested on four large datasets: road-usa, rgg-n-2-20-s0, roadNet-CA, roadNet-PA.
- Synthetic changes introduced to simulate dynamic network updates.

4.1 Scalability Analysis

- Scalability was tested by varying thread count (1 to 64).
- Larger graphs (e.g., road-usa) showed better scalability and higher speedup (up to 15x).
- Sparse graphs performed better due to lower chance of thread contention in neighbor updates.
- Dense graphs and smaller graphs with many updates showed diminished scalability due to increased race condition risks.

4.2 Algorithm Step Breakdown

- Updating SOSP trees (especially multiple objectives) consumed ~90% of total execution time.
- Steps involving graph combination and the final MOSP computation took minimal time.
- The Parallel Bellman-Ford algorithm was used in the final stage, dealing with the merged graph for a quick SOSP to approximate the MOSP.

5. Related Works

5.1 MOSP Search:

Early MOSP algorithms focused on Pareto-optimal path discovery, label-setting, and label-correcting techniques. Heuristic methods and lexicographic ordering are commonly used.

5.2 Parallel SOSP:

Various GPU-accelerated and shared-memory parallel implementations exist, including bucket-based Dijkstra and dynamic Bellman-Ford.

5.3 MOSP in Dynamic Networks:

Prior work mostly tackled the dynamic SOSP problem or static MOSP. Dynamic MOSP, especially with efficient parallel implementations, remained underexplored before this paper.

5.4 Parallel MOSP:

Previous efforts handled bi-objective cases using B-trees or pruning techniques. However, no known work previously explored parallel MOSP update in dynamic networks at the scale tackled here.

6. Conclusion

This research introduces:

- A parallel SOSP update algorithm using a grouping technique to avoid race conditions.
- A novel heuristic that transforms the multi-objective path update problem into manageable single-objective updates.
- A shared-memory parallel implementation that achieves significant speedups and scalability across large networks.

Key Takeaways:

- The approach balances efficiency and accuracy by focusing on one representative MOSP instead of exhaustively computing all Pareto-optimal paths.
- The method is scalable, with real-world and synthetic experiments validating its performance.
- The algorithm is currently designed for edge insertions but can potentially be extended to handle deletions.
- Future directions include exploring hybrid parallelism (combining distributed and shared memory) to further reduce execution time for high-objective scenarios.
