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

Arindam Khanda
akkcm@mst.edu
Missouri University of Science and Technology
Rolla, MO, USA

S M Shovan
sskg8@mst.edu
Missouri University of Science and Technology
Rolla, MO, USA

Sajal K. Das
sdas@mst.edu
Missouri University of Science and Technology
Rolla, MO, USA

ABSTRACT

In dynamic networks, where continuous topological changes are prevalent, it becomes paramount to find and update different graph properties without the computational burden of recalculating from the ground up. However finding or updating a multi-objective shortest path (MOSP) in such a network is challenging, as it involves simultaneously optimizing multiple (conflicting) objectives. In light of this, our paper focuses on shortest path search and proposes parallel algorithms tailored specifically for large incremental graphs. We first present an efficient algorithm that updates the single-objective shortest path (SOSP) whenever a new set of edges are introduced. Leveraging this SOSP update algorithm, we also devise a novel heuristic approach to adaptively update a MOSP in large networks. Empirical evaluations on both real and synthetic incremental networks with shared memory implementations attest to the scalability and efficacy of the proposed algorithms.

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**.

KEYWORDS

dynamic graph, parallel algorithm, shortest path

ACM Reference Format:
Arindam Khanda, S M Shovan, and Sajal K. Das. 2023. A Parallel Algorithm for Updating a Multi-objective Shortest Path in Large Dynamic Networks. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3624062.3625134

1 INTRODUCTION

Shortest path computations have numerous real-world applications, such as route recommendations in road transportation networks [9], centrality computations in social network analysis [18], efficient communications in wireline, wireless, or sensor networks [34], drone-based goods delivery [15, 16], to name a few. However, shortest path computation is challenging for large dynamic networks, where the underlying graph topology changes with time. This complexity intensifies when the paths seek to optimize multiple objectives. For example, in *road transportation networks*, one may optimize different objectives such as distance, estimated travel time, traffic congestion, road conditions, fuel consumption, and road safety. On the other hand, in *wireless sensor networks* (WSNs), the data gathered by the sensor nodes are usually transferred to a data collection point (called the sink), and the data collection route typically follows a tree structure rooted at the sink. While a shortest-distance route from the sensor nodes to the sink decreases the data collection latency, it will deplete faster the energy (battery power) of the nodes closer to the sink, which in turn may reduce the overall lifetime of the network. Therefore, it is necessary to jointly optimize the latency and energy consumption along the data collection routes in WSNs [13].

In the above example scenarios, finding the most efficient routes is the same problem as finding the shortest paths that optimize various factors, known as the *multi-objective shortest path (MOSP)* search. *Pareto optimality* is a well-known method for finding reasonable solutions to multi-objective search problems, as it provides a condition where no preference criterion can be improved without harming at least one other preference criterion [4]. Therefore, many solution approaches seek Pareto optimal paths to achieve multi-objective optimization along the routes. Although there exists a few parallel algorithms to compute all Pareto optimal shortest paths [31], parallel algorithms for the MOSP problem in large dynamic networks are yet to be explored.

Since the networks representing many real-world complex systems are often dynamic, the MOSP computation can become stale due to topological changes over time. Recomputing the MOSP on periodic snapshots can be inefficient due to repetitive and redundant calculations. The temporal nature of dynamic networks poses additional challenges, as the uncertainty of changes creates irregular access patterns for graph algorithms. These motivate our work.

In this paper, we mainly focus on incremental networks, i.e., the networks that grow with time. We first develop a parallel algorithm (SOSP-update) to update a shortest path optimizing a single objective function. Then we exploit the SOSP-update algorithm to design an efficient heuristic for updating a MOSP in rapidly growing large networks. To the best of our knowledge, we propose the first parallel algorithms for updating MOSP in dynamic networks. Our novel contributions are summarized as follows:

- We first propose a parallel SOSP update algorithm that uses grouping techniques to reduce the total work while maintaining correctness.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3625134

- Then we use our proposed SOSP update algorithm to design a novel heuristic algorithm that quickly updates a single MOSP in a large network under time-varying dynamics.
- We present a shared-memory parallel implementation to compute SOSPs and MOSPs in large dynamic networks. Scalability tests on real and synthetic networks affirm our implementation's effectiveness.

The organization of the rest of this paper is as follows. Section 2 introduces preliminary concepts of SOSP, MOSP, Pareto optimality, and dynamic networks. Section 3.1 presents a parallel algorithm for updating SOSP; while Section 3.2 describes a parallel heuristic algorithm to find a single MOSP in dynamic networks. Section 4 describes a shared memory parallel implementation and experimental results. Section 5 reviews previous work related to SOSP and MOSP search, and relevant parallel approaches. The final section concludes the paper with future directions.

2 PRELIMINARIES

Let $G(V, E)$ be a directed graph, where V is the vertex set and E is the edge set. Each directed edge $e(u, v) \in E$ from vertex u to v has a non-negative weight $W(e)$. A directed path between two vertices is called a *shortest path* if the sum of the weights of edges between these two vertices is the smallest. The single source shortest path problem computes the shortest paths from a source vertex to all other vertices as destinations. For only one objective (i.e., $k = 1$), the problem is called *Single Objective Shortest Path* (SOSP) and the output is an *SOSP tree* (say, T) containing only those edges along shortest paths from the source to all other vertices.

2.1 Multi-objective Shortest Path (MOSP)

When a complex system with multiple objectives is modeled as a graph, the weight of an edge $e \in E$ becomes a vector, where i^{th} element of the vector describes the distance/similarity between two interacting entities considering only i^{th} objective. As a result, the shortest path problem optimizes multiple objective functions and becomes a MOSP search. Let the weight vector of edge e be $W(e) = (w_1, \ldots, w_k)$, where w_k is the weight related to k^{th} objective.

Pareto optimization is a well-known technique [4] that provides solutions not worse than any other solution in a multi-objective optimization problem. In the context of shortest path search, the *Pareto optimal shortest paths* are those for which the multi-objective distance vectors are not dominated by any other path's distance vector. In this paper, we design efficient parallel algorithms to update Pareto optimal paths in large dynamic networks.

Let the shortest distance (Pareto optimal labels) of a vertex v from the source vertex be $(v, \overrightarrow{l}) = \{\{d_1^{p1}, \ldots, d_{|p1|}^{p1}\}, \ldots, \{d_1^{pz}, \ldots, d_{|pz|}^{pz}\}\}$, where d_j^{pi} denotes the j^{th} Pareto optimal distance measured from the source through parent vertex p_i. Each d_j^{pi} contains an individual distance component for each objective function, and hence $d = (\delta_1, \ldots, \delta_k)$, where δ_k is the distance component computed just for objective k along the Pareto optimal path.

Let Figure 1 be an illustration of a road network graph. Each edge has dual weights: travel time and fuel consumption between two points (nodes). Note that travel time and fuel consumptions are not linearly correlated due to road elevation and traffic. Here, the Pareto optimal label for vertex u_6 is $(u_6, \overrightarrow{l}) = \{u_4 : \{(6, 16), (12, 4)\}, u_5 : \{(17, 9), (23, 7), (14, 12)\}\}$. It indicates there are two and three shortest paths passing through vertex u_4 and u_5 respectively. Each tuple in the label has two distance components showing the required time and fuel consumption along a path.

Dominated distance: In the process of Pareto optimal shortest path computation, for a vertex v, a potential path distance $d_i = (\delta_1^i, \ldots, \delta_k^i)$ is called dominated iff there exists at least another distance $d_j = (\delta_1^j, \ldots, \delta_k^j)$ such that:

$$\delta_x^j < \delta_x^i, \text{ for a single value of } x, \text{ where } 1 \le x \le k \quad (1)$$

$$\delta_y^j \le \delta_y^i, \text{ for all } y \ne x \text{ and } 1 \le y \le k \quad (2)$$

If d_i is dominated by d_j, we will denote it as $d_j \prec d_i$. A dominated distance is eliminated from the Pareto optimal distance set.

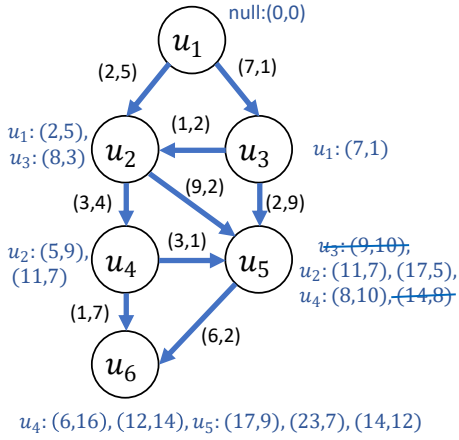


Figure 1: MOSP in an example graph

In Figure 1, $\{u_3 : (9, 10)\}$ is dominated by Pareto optimal label $\{u_4 : (8, 10)\}$ in the process of shortest distance computation of vertex e. Similarly, $\{u_4 : (14, 8)\}$ is dominated by $\{u_2 : (11, 7)\}$. Table 1 enumerates the list of symbols used in this paper.

Table 1: List of Symbols

Symbols	Meaning
$G(V, E)$	A directed weighted graph
$W(e)$	Weight vector (related to objectives) of edge e
k	Number of objectives
ΔE	Set of changed edges
Ins	Set of inserted edges in ΔE
T_i	SOSP tree related to objective i
L	A vector of Pareto optimal labels
$(u, \overrightarrow{l})$	Tuple containing vertex u and a vector of all Pareto optimal labels of u
d_j^u	j^{th} Pareto optimal distance through parent u
δ_i	Distance component related to objective i

2.2 Dynamic Networks

Time-varying dynamics in a graph include both vertex and edge-level changes. As vertex insertion (resp. deletion) can be converted to the addition (resp. deletion) of connecting edges, without loss of generality, we can assume all changes in the network are due to edge modifications. A dynamic graph algorithm accommodates

modifications in graph topology, including edge insertions, and deletions while determining specific graph attributes. Such algorithms can operate in real-time, updating after every alteration [30], or handle a batch of changes [14] in near real-time. Instead of recalculating from the beginning, a dynamic update algorithm tweaks the graph property in response to these changes requiring less computational resources.

SOSP is a classic shortest-path problem, for which numerous solutions have been proposed in the literature [1, 17, 33]. It has been observed that in a dynamic network, updating an SOSP requires less time than recomputing it from scratch when changes occur in the network topology [17]. Inspired by this finding we develop an enhanced algorithm for updating SOSP in large incremental networks and later design a MOSP search heuristic.

3 PROPOSED APPROACH

Here, we first develop an algorithm to update SOSP and use the technique to design a MOSP search algorithm in a dynamic incremental network. Let $G_t(V_t, E_t)$ be the directed graph at time step t and $(v, \overrightarrow{l})_t \in L_t$ be the Pareto optimal distance labels (shortest distances in case of SOSP) of a vertex v. Let from time step t to $t+1$, the set of inserted edges be $Ins_t = E_{t+1} - E_t$. Thus, the updated graph $G_{t+1}(V_{t+1}, E_{t+1})$ contains the edge set $E_{t+1} = (E_t \cup Ins_t)$. Our *aim* is to efficiently compute the shortest distance labels $(v, \overrightarrow{l})_{t+1} \in L_{t+1}$ for all $v \in V_{t+1}$, without recomputing from scratch. For generality, we omit the subscript t in our algorithm notation.

3.1 Single-objective Shortest Path Update

The proposed SOSP update algorithm uses $G(V, E)$, the SOSP tree $T = \{\{(v, \delta) : v \in V\}, Parent\}$ of the last time instance, and the set of changed edges Ins as the inputs. (v, δ) and $Parent[v]$ store the distance and parent vertex of v in the SOSP tree respectively. A new set of directed edges $\{(u_1, v), \ldots, (u_x, v)\}$ can affect the distance of vertex v only. If they are processed using multiple asynchronous threads, each thread may update the distance differently and result in an incorrect shortest distance. Existing solutions [17] use multiple iterations to achieve correctness in such scenarios. Unlike this approach, we use a simple grouping technique to avoid multiple iterations. **Preprocessing (Step 0):** At preprocessing stage all the inserted directed edges (u, v) are grouped by the second endpoint v and stored in $\mathcal{I}[v]$. Here, the j^{th} element of $\mathcal{I}$ stores the set of changed edges having the possibility of affecting j^{th} vertex. The grouping simply performs set insert operations ($O(1)$ time on average), while reading the changed edges. **Process Changed Edges (Step 1):** In this step, each group of inserted edges is processed by a single thread. Therefore, the distance of a vertex v is updated only by a single thread and it removes the possibility of incorrect update due to race condition. If a newly added edge (u, v) decreases the distance of v, then the distance is updated as $(u, \delta) + W(u, v)$, and v is marked as affected (Algorithm 1 line 8 to 12). **Propagate the Update (Step 2):** An affected vertex may also affect the distance of its neighbors and updating the neighbors' distance is necessary to maintain the correctness. As multiple affected vertices can have common neighbors, processing the neighbors of each affected vertex in different threads may lead to race conditions. To avoid such a situation, step 2 first gathers all unique neighbors of all the affected vertices in a vector $\mathcal{N}$. Then the vertices $v \in \mathcal{N}$ are assigned to parallel threads where each thread checks for the predecessors which are already marked as affected. If found, the edge between the predecessor and v is relaxed to find if a path through the affected predecessor can decrease the distance of v. If v's distance decreases, it's marked as affected and its neighbors are processed in the next iteration. Therefore, Step 2 is an iterative process and it completes when no new affected vertices are identified (Algorithm 1 line 14). The SOSP update algorithm finds the updated shortest path for all the vertices from the source, i.e., the updated SOSP tree.

Figure 2 illustrates the proposed SOSP update algorithm. Let Figure 2a and 2b be an example network and its initial SOSP tree respectively. Let the changed edges be $Ins = \{(u_1, u_2, 7), (u_3, u_5, 1), (u_1, u_5, 4)\}$. Then after the preprocessing step, the group of changed edges becomes $\mathcal{I}[u_2] = \{(u_1, u_2, 7)\}$ and $\mathcal{I}[u_5] = \{(u_3, u_5, 1), (u_1, u_5, 4)\}$. After processing these two groups using two asynchronous threads in step 1, the distance of u_2 and u_5 are updated (see Figure 2c). At the first iteration of step 2, $\mathcal{N}$ contains $\{u_4, u_6\}$ and the distances of these two vertices are updated (see Figure 2d). In the next iteration, $\mathcal{N}$ contains only u_6 and its distance update is shown in Figure 2e. Finally, the iteration stops as no neighbor is left for update and the final updated SOSP tree is found (see Figure 2f).

3.2 Multi-objective Shortest Path Update

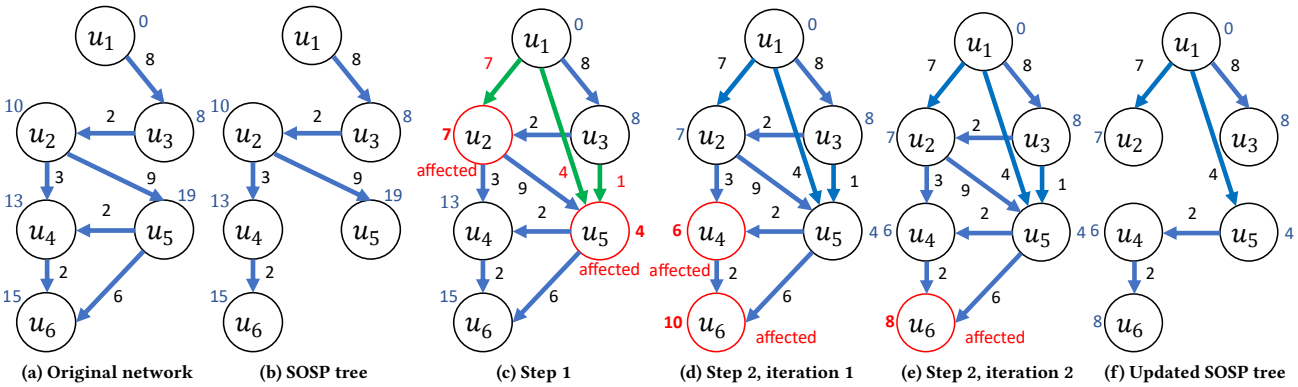
The Pareto optimal solution to find MOSP generally produces numerous shortest paths where none can be considered worse than the other. However, in most cases, the application requires only one MOSP. In a dynamic network, as the network topology changes fast, it is more important to find a single path quickly than to find all the paths. Searching for a single MOSP rather than finding all MOSPs can improve execution time and decrease resource requirements. To achieve this goal, we propose a novel time-efficient heuristic algorithm that finds a single MOSP in a large dynamic graph.

Step 1: Update SOSP trees In the presence of newly inserted edges, the algorithm (Algorithm 2) first updates each SOSP tree T_i related to i^{th} objective using Algorithm 1.

Step 2: Create a combined graph Each updated SOSP tree T_i provides an updated path for which the distance component δ_i is the minimum for all the vertices. However, the path T_i considers a single objective only. Real applications often require MOSP balancing all the objectives or combining different priorities and constraints. Therefore, in this step, the algorithm balances the objectives or prioritize i^{th} objective function by increasing the probability of selecting an edge (or sub-path) from T_i. The algorithm first creates an ensemble graph $\mathbb{E}$ by considering all the edges from the SOSP trees $T_i \forall i = 1, \ldots k$. If an edge $e \in \mathbb{E}$ appears in x number of SOSP trees, then the balanced approach assigns edge weight $(k - x + 1)$ to that edge. This approach assigns less weight to edges that appear in more SOSP trees while assigning more weight to uncommon edges.

Objectives with different priorities also can be dealt with using a similar method where the weight of an edge in T_i is assigned a positive value inversely proportional to the priority of objective i.

Application Scenario: In a drone-based delivery system, let there be two efficient delivery routes T^f, and T^e depending on the shortest flying time and the lowest energy consumption, respectively. Let the energy budget be B, and energy consumption to deliver an item by following T^f(resp. T^e) be c_f(resp. c_e). If $c_f > B > c_e$, the system prioritizes energy cost over delivery



(a) Original network (b) SOSP tree (c) Step 1 (d) Step 2, iteration 1 (e) Step 2, iteration 2 (f) Updated SOSP tree

Figure 2: SOSP update.

Algorithm 1: SOSP_Update($G(V, E), T, Ins$)

```
/* Step 0: Preprocessing */
Initialize an array I of size |V| where each element is an empty list.
for each directed edge e(u, v) ∈ Ins do
    Add (u, v) to I[v]
/* Step 1: Process Changed Edges */
Initialize an empty vector Aff
Initialize an array marked containing zeroes of size |V|
for each vertex v ∈ V in parallel do
    for each edge (u, v) ∈ I[v] do
        if (v, δ) > (u, δ) + W(u, v) then
            Add v in Aff
            Change the distance of v to (u, δ) + W(u, v)
            Parent[v] ← u
            marked[v] ← 1
/* Step 2: Propagate the update */
while Aff in not empty do
    Initialize empty vectors N and Aff'
    for each v ∈ Aff in parallel do
        Add the neighbors of v in N
    for each v ∈ N in parallel do
        for each predecessor neighbor u of v do
            if marked[u] ≠ 1 then
                continue
            if (v, δ) > (u, δ) + W(u, v) then
                Add v in Aff'
                Change the distance of v to (u, δ) + W(u, v)
                Parent[v] ← u
                marked[v] ← 1
    Aff ← Aff'
    Aff' is reset to empty vector
```

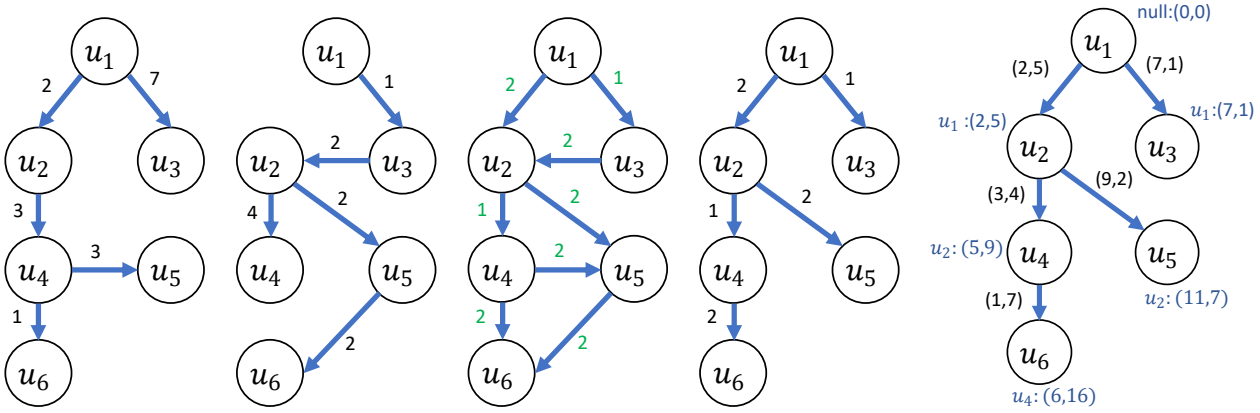
Algorithm 2: MOSP_Update($G(V, E), \{T_1, \ldots, T_k\}, Ins$)

```
/* Step 1: Find updated SOSP tree T_i */
for i = 1 to k do
    SOSP_Update(G(V, E), T_i, Ins)
/* Step 2: Create a combined graph */
E = ∪_{i=1}^{k} (e ∈ T_i)
for each e ∈ E in parallel do
    if e appears in x number of SSSP trees then
        Assign edge weight (k − x + 1) for e ∈ E
/* Step 3: Find SOSP in combined graph */
Find SOSP in E
Assign actual edge weights from updated G on the output SOSP tree to find the MOSP
```

time to ensure the drones can return to their charging point. However, if $B > c_f \geq c_e$, the system may choose to follow T^f to deliver the items faster. In addition, the system may need to adjust its delivery objectives under varying wind conditions. In a dynamic scenario, it may be beneficial to reserve some energy budget for emergencies and follow a MOSP approach to balance both time and energy objectives while delivering goods.

Step 3: Find SOSP in the combined graph The combined graph $\mathbb{E}$ contains the edges appearing in one SOSP tree at least, and the edge weights are assigned depending on the priority of the objectives. Next, the algorithm finds an SOSP in the combined graph using any parallel single source shortest path algorithm. If the edge weights from the actual graph G are reassigned to the edges of SOSP computed on the combined graph, it provides an optimal or sub-optimal solution for MOSP that satisfies the targeted combination of objectives.

Let Figure 1 be an updated graph G_{t+1} after a set of edge insertions; Figure 3a and 3b be the updated SOSP trees (after Algorithm 2 step 1) related to objectives 1 and 2, respectively. Following Step 2, a combined graph is created, as shown in Figure 3c. When SOSP is computed on this combined graph and the original edge weights from G_{t+1} are reassigned, the resulting path provides one of the MOSPs presented in Figure 1, as illustrated in Figure 3e.



(a) SOSP: objective 1 (b) SOSP: objective 2 (c) Combined graph (d) SOSP in combined graph (e) Final MOSP

Figure 3: Finding a single MOSP.

Theorem 1. *Let T_i be the only SOSP tree related to objective i in graph G. Let an ensemble graph $\mathbb{E}$ be obtained by using all the edges from SOSP trees $T_i \forall i = 1, \ldots k$ and by assigning a single and equal edge weight for each of these edges. Let T^E be a possible SOSP in the ensemble graph. If real edge weights are reassigned for the edges of T^E, it will give a Pareto optimal shortest path solution for G.*

Proof. If there exists a single objective, trivially, the SOSP will be the Pareto optimal path. For a set of objectives, let us assume we found the Pareto optimal path from source to u that follows the edges from T^E. u itself can be the source vertex (base case). Let there be an edge $(u, v) \in T^E$. Now we need to prove that the path to v through u is Pareto optimal.

As we assume there exists only one SOSP tree for a single objective, there cannot exist other paths (not through u) to v that are better in terms of at least one objective function without being worse in the other objectives. Therefore, per the Pareto optimality's definition, the path to v through u is Pareto optimal. □

Lemma 2. *If T_i is the only SOSP tree related to objective i, any path along the tree will be a subpath of the Pareto optimal MOSP solution.*

Proof. Let the distance of vertex v be $d(v) = (\delta_1, \ldots, \delta_i, \ldots, \delta_k)$ along T_i and $d_x(v) = (\delta_1^x, \ldots, \delta_i^x, \ldots, \delta_k^x)$ along an alternate path. Assume, $d(v)$ is not Pareto-optimal and $d_x(v) \prec d(v)$. Then $\delta_i^x \leq \delta_i$. However, it is impossible as δ_i is the shortest distance of v along the only SOSP related to objective i. So, $d(v)$ is the Pareto optimal distance. Thus, any path along T_i overlaps with the Pareto optimal MOSP solution. □

Theorem 3. *If at time $t+1$, the updated dynamic graph $G_{t+1} = (G_t \cup Ins)$ has only one SOSP tree T_i for each objective i, the path found in G_{t+1} by Algorithm 2 will be a Pareto optimal shortest path.*

Proof. According to the definition of Pareto optimality, a label is Pareto optimal if it has at least one objective component less or equal to all other Pareto optimal labels. Finding the SOSP tree for each objective contributes exactly one Pareto optimal candidate. In the combined graph, the proposed algorithm provides weights to the edges to give priority to selecting the optimal path suggested by most of the SOSP trees. As we are choosing the path of the graph having the highest occurrence path among the SOSP trees, the algorithm provides exactly one, not necessarily all possible, Pareto optimal path. □

Probable Optimization- Updating SOSP in Combined Graph: Initially the algorithm needs to compute the SOSP tree in the combined graph from scratch. Later the algorithm can use the SOSP tree computed in $\mathbb{E}_t$ (at time t) and the changed edges found in the new ensemble graph $\mathbb{E}_{t+1}$ to update the SOSP tree using a similar approach proposed in Algorithm 2 Step 1A and 1B.

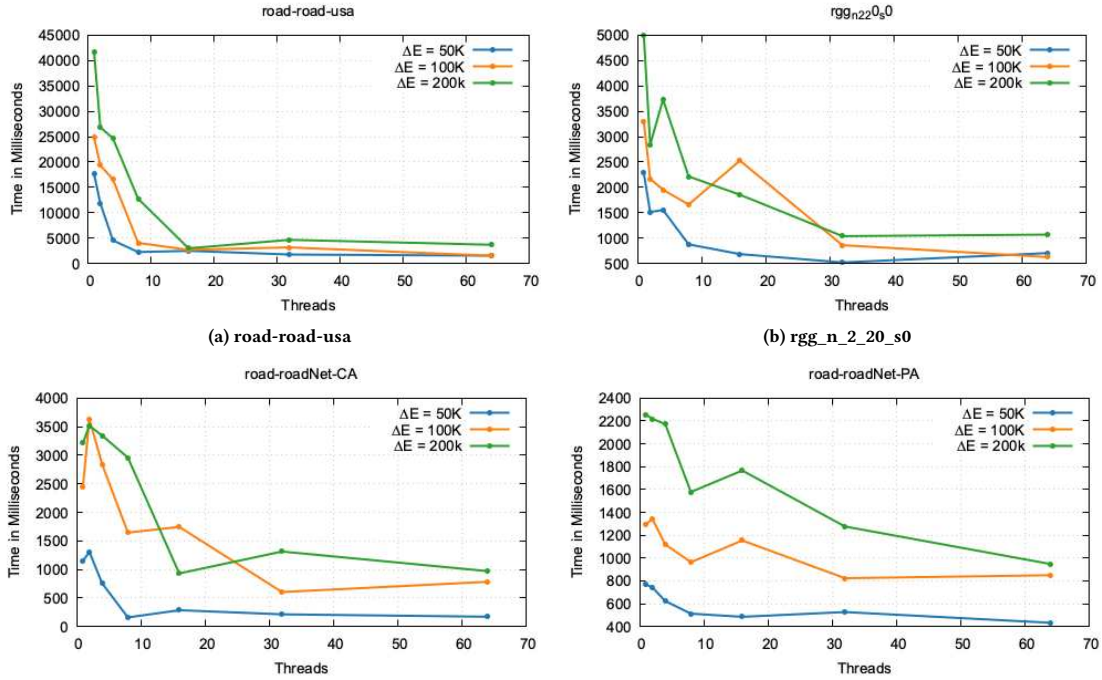
Discussion on Our Approach:

1. Finding a MOSP with two or more objectives is known to be an NP-hard problem. Our approach converts a MOSP problem into an SOSP problem, reducing total execution time.
2. Our algorithm helps to find an optimal or sub-optimal path where the multiple objectives are balanced. Reduced weight for the edges common in multiple SOSP trees increases the chance of those edges being selected in Step 3, which means the edges, optimizing multiple objectives together, get priority.
3. The weight assignment in the ensemble graph can be modified to implement objectives with different priorities. This allows the user to customize the optimization criteria according to their needs.
4. A single MOSP update algorithm in a dynamic network saves execution time and resources.

4 PERFORMANCE EVALUATION

We implement Algorithm 1 and use it to implement 2 for shared-memory computing architecture using C++ and OpenMP. The adjacency list and changed edges are stored using arrays of structures. Each element in the change edge structure stores the endpoints of an edge, edge weight, and a flag to indicate insertion/deletion status. As we modify the MOSP problem to a combination of multiple SOSP update problems, the SOSP tree is an important data structure in our implementation. We store the SOSP tree as a parent-child relationship among the vertices. Each element of the SOSP tree contains the *Parent* vertex, and *Distance* from the source. A vertex is marked as affected whenever the vertex's distance is changed in the process of MOSP update.

Updating an SOSP tree takes advantage of both edge-centric and vertex-centric parallel operations. Each group of changed edges is



(a) road-road-usa (b) rgg_n_2_20_s0

(c) road-roadNet-CA (d) road-roadNet-PA

Figure 4: Scalability Analysis.

processed by each shared-memory thread, which is scheduled dynamically. On the other hand, each neighbor of the affected vertices is assigned to a single thread for processing. In the current implementation, an SOSP tree T_i related to the i^{th} objective is updated only when T_{i-1} completes its update. However, in a distributed architecture or a scenario with a massive number of parallel threads, they can be updated independently.

In Step 2 of Algorithm 2, computing a set union and finding common edges from all T_i can be computationally expensive. To address this, we directly use the parent-child relationship in the tree structure to find the edges. We assign a single thread to each vertex to compare its parents among all the SOSP trees. Then, we use an *OpenMP custom reduction* to gather all the edges after assigning the weight $(k - x + 1)$ as described in the algorithm. For Step 3 of Algorithm 2, we use a parallel Bellman-Ford algorithm implementation to compute the SOSP on the combined graph.

Experimental Setup: All experiments are conducted on dual 32-core AMD EPYC Rome 7452 CPUs with 64 GB DDR4 RAM allocated. Four Large graphs from the network-repository collection [29] are chosen and a set of random edge weights are added depending on the number of objectives. Details of the networks are given in Table 2. We choose road networks and a random geometric graph, particularly considering the multi-objective application scenarios of road transportation and wireless sensor network respectively. To make our datasets dynamic in our experiment, we randomly generate batches of changed edges.

4.1 Scalability Analysis

For strong scaling analysis, we increase the number of *OpenMP* threads from 1 to 64 while keeping the batch size constant for each experiment. Figure 4 shows threads vs time when batch size is varied among three different sizes of ΔE 50K, 100K, and 200K. The general trend shows that the execution time decreases gradually when the number of threads increases.

Table 2: Networks in Our Test Suite [29]

Name	Num. of Vertices	Num. of Edges
road-usa	23M	28.9M
rgg-n-2-20-s0	1,048,576	6,891,620
roadNet-CA	1,971,281	5,533,214
roadNet-PA	1,090,920	3,083,796

Although the execution time depends on the location of the change in the graph, it is possible to explain the general nature of the plots for the different sizes of edge insertion.

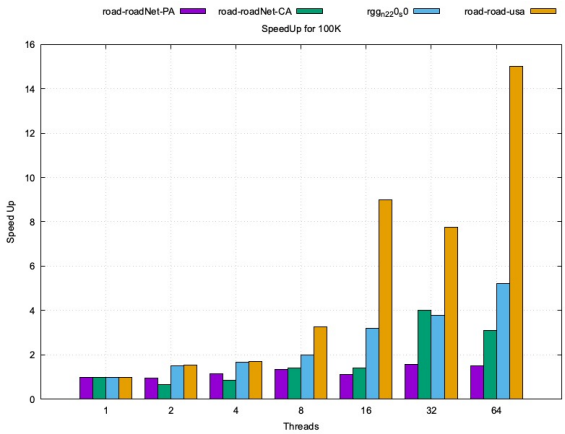


Figure 5: Execution time ratio (speedup) compared to single thread execution.

In our algorithm 3, lines 18 and 19 indicate that the exterior loop in line 18 is processed in parallel and the inner loop in line 19 is processed with a single thread due to nesting. If the graph is sparse, the sequential inner loop in line 19 will not significantly hinder the parallel performance compared to a dense graph. The scalability of sparse graphs will be greater than that of dense graphs. Similar experimental results are depicted in Figure 4. Despite the vast scale of the road-road-usa network, its scalability performance is superior to that of other plots. For obvious reasons, the scalability performance of smaller graphs with a large number of edge insertions is negatively impacted. In line 18 of Algorithm 3, the probability of having the same node affected by other nodes for multiple iterations increases if a large number of edge modifications are applied to smaller-scale graphs. For MOSP, the most dominant operation is the *SOSP_update* operation according to the Figure 6 covering around 90% of space regardless of the size and sparsity of the graph. Thus, the scalability performance of *SOSP_update* also implies the scalability performance of MOSP problem.

To the best of our knowledge, no parallel implementation is available to update a single MOSP in dynamic networks. Therefore, we consider a single thread execution, i.e., sequential approach as our baseline. Figure 5 shows the execution time ratio (speedup) of single and multi-thread executions when the datasets are varied. The largest network in our test suite, i.e., road-usa shows the maximum speedup (up to 15X).

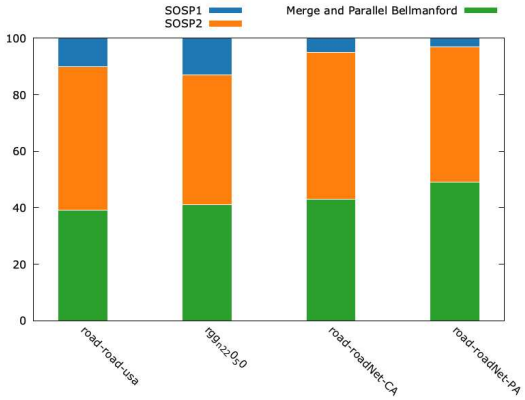


Figure 6: Execution time of different steps.

4.2 Experiment on Different Algorithmic Steps

Figure 6 shows the percentage of time taken for executing different steps of the algorithm when the number of threads is fixed to 4. Without loss of generality, we consider a bi-objective case here. In the figure, *SOSP1* (resp. *SOSP2*) indicates the time taken to update T_1 (resp. T_2) in presence of ΔE be 100K. Updating T_1 and T_2 takes the most time in the whole process, whereas creation of the combined tree (merge operation) takes barely any time. The Parallel Bellman-Ford algorithm finds an SOSP on a combined graph of $2 * (|V| - 1)$ or fewer edges and consumes a small fraction of the total time.

5 RELATED WORKS

This section reviews previous works related to MOSP search, parallel SOSP, MOSP in dynamic networks, and parallel MOSP search.

5.1 Multi-objective Shortest Path

An algorithm to compute bi-objective shortest path was first introduced in [8]. In [21], it was generalized into a multi-objective technique by including lexicographic ordering of labels. Later, the algorithm in [8] was expanded and the concept of Pareto optimality was introduced in [32]. The authors in [10] conducted a comparison study to determine if there was a relationship between the graph density and label setting and label correction algorithm performance. Using a bi-objective shortest path as the focus, a thorough analysis of label-setting algorithms was reported in [26]. This study demonstrated the potential of parallelism for a two-phase strategy, the first of which decomposes the actual problem. In [24] is investigated a collection of previously known labeling techniques. A method to support multi-objective $A*$ algorithms that can estimate the cost of achieving the target state for more than one objective was proposed in [20]. The research presented in [19] determined that the multi-objective $A*$ algorithm provides high-quality solutions and can significantly benefit from heuristic information.

5.2 Parallel Single-objective Shortest Path

A plethora of parallel SOSP algorithms have been proposed in the literature. In [19], Dijkstra's method was divided into independent phases. The authors in [22] adopted the bucket data structure to keep track of the approximate distances. A high-performance graph library, called Gunrock [33], provides a three-step architecture (advance, filter, and compute) based implementation to compute SOSP on Nvidia GPUs. Efficient dynamic parallelism-based implementation of the Bellman-Ford algorithm using two queues on GPU is proposed in [1]. An architecture-independent framework to update SOSP in fully dynamic networks is proposed in [17], which provides shared memory and GPU-based implementations.

5.3 MOSP in Dynamic Networks

The first attempt to compute MOSP in dynamic networks was made in [2], which modified Bellman's method and used a recursive formula to determine the shortest distance. The author in [6] extended it for dynamic instances based on first-in first-out (FIFO) property and non-overtaking property. By managing numerous modifications simultaneously, the dynamic shortest path problem was generalized in [27]. Dynamic all-pair shortest path methods were first used in practice in [5]. Two factors of dynamic nature , namely temporal variation and weight updates, were considered in [23]. Multiple objective optimization in transportation is dealt in [11], which demonstrated that fuel usage varies with truckload and is correlated with carbon emission levels. To adopt the SOSP algorithm for multiple objectives, the authors combined the three elements into a single polynomial. A dynamic programming-based shortest path algorithm is proposed in [28] for non-additive edge weights maintaining multi-objectives and multi-constraints.

5.4 Parallel MOSP

The authors in [31] were the first to tackle the parallel MOSP challenge. They developed a shared-memory parallel algorithm for bi-objective shortest path problem [7] and hypothesized a parallel multi-objective variant. To achieve further parallelism, they used the B-tree data structure for two objectives. However, The data structure fails in multi-objective instances. Strategies to lower the dimensionality and convert the multi-objective problem into a bi-criteria dilemma are proposed in [25]. A new pruning-based technique which was devised in [3] to check the dominance of

the labels concurrently. This algorithm performed up to 2–9 times better than the Martin's algorithm[12].

To the best of our knowledge, the parallel MOSP problem in large dynamic graphs is not explored before.

6 CONCLUSION

In this work, we first presented a parallel SOSP update algorithm that incorporates grouping techniques. This approach effectively reduces computational efforts by decreasing the total iteration count. Building on this foundation, we devised a heuristic algorithm tailored to promptly update a single MOSP in large networks, particularly under time-varying dynamics. Finally, we developed shared-memory parallel implementations optimized for efficient computation of both SOSPs and MOSPs in incremental networks. The effectiveness and practicality of our implementations are validated through scalability assessments conducted on both real-world and synthetic networks.

While our paper primarily focuses on incremental graphs, specifically edge insertions, the algorithm has the potential to be adapted for edge deletions. We plan to address this in upcoming work. Our current implementation updates the SOSP trees one after another leading to longer execution times with a higher number of objectives. A potential solution lies in adopting hybrid parallelism: distributing tasks associated with each SOSP tree across processors, and then utilizing shared-memory parallelism within each processor for the SOSP update. We foresee a reduction in execution time with this approach and aim to investigate this area in the future.

ACKNOWLEDGMENTS

This work was supported by the NSF grants "Sparsification-based Approach for Analyzing Network Dynamics" (SANDY) under award # OAC-1725755, and "Cyberinfrastructure for Accelerating Innovation in Network Dynamics" (CANDY) under award # OAC-2104078.

REFERENCES

[1] Federico Busato and Nicola Bombieri. 2015. An efficient implementation of the Bellman-Ford algorithm for Kepler GPU architectures. *IEEE Trans. Parallel and Distributed Systems* 27, 8 (2015), 2222–2233.

[2] Kenneth L Cooke and Eric Halsey. 1966. The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications* 14, 3 (1966), 493–498.

[3] Pedro Maristany de las Casas, Antonio Sedeno-Noda, and Ralf Borndörfer. 2021. An improved multiobjective shortest path algorithm. *Computers & Operations Research* 135 (2021), 105424.

[4] Kalyanmoy Deb and Himanshu Gupta. 2005. Searching for robust Pareto-optimal solutions in multi-objective optimization. In *Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005. Proceedings 3*. Springer, 150–164.

[5] Camil Demetrescu and Giuseppe F Italiano. 2006. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Transactions on Algorithms (TALG)* 2, 4 (2006), 578–601.

[6] Stuart E Dreyfus. 1969. An appraisal of some shortest-path algorithms. *Operations research* 17, 3 (1969), 395–412.

[7] Stephan Erb, Moritz Kobitzsch, and Peter Sanders. 2014. Parallel bi-objective shortest paths using weight-balanced b-trees with bulk updates. In *International Symposium on Experimental Algorithms*. Springer, 111–122.

[8] Günter Fandel and Tomas Gal. 2012. *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*. Vol. 177. Springer Science & Business Media.

[9] Honghao Gao, Wanqiu Huang, and Xiaoxian Yang. 2019. Applying Probabilistic Model Checking to Path Planning in an Intelligent Transportation System Using Mobility Trajectories and Their Statistical Data. *Intelligent Automation & Soft Computing* 25, 3 (2019).

[10] F Guerriero and R Musmanno. 2001. Label correcting methods to solve multicriteria shortest path problems. *Journal of optimization theory and applications* 111, 3 (2001), 589–613.

[11] Yi-Nan Guo, Jian Cheng, Sha Luo, Dunwei Gong, and Yu Xue. 2017. Robust dynamic multi-objective vehicle routing optimization method. *IEEE/ACM transactions on computational biology and bioinformatics* 15, 6 (2017), 1891–1903.

[12] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.".

[13] Sk Kajal Arefin Imon, Adnan Khan, Mario Di Francesco, and Sajal K. Das. 2013. RaSMaLai: A Randomized Switching algorithm for Maximizing Lifetime in tree-based wireless sensor networks. In *2013 Proceedings IEEE INFOCOM*. 2913–2921. https://doi.org/10.1109/INFCOM.2013.6567102

[14] Arindam Khanda, Sanjukta Bhowmick, Xin Liang, and Sajal K Das. 2022. Parallel Vertex Color Update on Large Dynamic Networks. In *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 115–124.

[15] Arindam Khanda, Federico Corò, and Sajal K Das. 2022. Drone-Truck Cooperated Delivery Under Time Varying Dynamics. In *Proceedings of the 2022 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems* (Salerno, Italy) *(ApPLIED '22)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/3524053.3542743

[16] Arindam Khanda, Federico Corò, Francesco Betti Sorbelli, Cristina M. Pinotti, and Sajal K. Das. 2021. Efficient Route Selection for Drone-based Delivery Under Time-varying Dynamics. In *IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. 437–445. https://doi.org/10.1109/MASS52906.2021.00061

[17] Arindam Khanda, Sriram Srinivasan, Sanjukta Bhowmick, Boyana Norris, and Sajal K. Das. 2022. A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 929–940. https://doi.org/10.1109/TPDS.2021.3084096

[18] Jianxin Li, Taotao Cai, Ke Deng, Xinjue Wang, Timos Sellis, and Feng Xia. 2020. Community-diversified influence maximization in social networks. *Information Systems* 92 (2020), 101522.

[19] Enrique Machuca, Lawrence Mandow, JL Pérez De La Cruz, and Amparo Ruiz-Sepúlveda. 2012. A comparison of heuristic best-first algorithms for bicriterion shortest path problems. *European Journal of Operational Research* 217, 1 (2012), 44–53.

[20] Lawrence Mandow and José Luis Pérez De La Cruz. 2008. Multiobjective A* search with consistent heuristics. *Journal of the ACM (JACM)* 57, 5 (2008), 1–25.

[21] Ernesto Queiros Vieira Martins. 1984. On a multicriteria shortest path problem. *European Journal of Operational Research* 16, 2 (1984), 236–245.

[22] Ulrich Meyer and Peter Sanders. 2003. Δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms* 49, 1 (2003), 114–152.

[23] Giacomo Nannicini and Leo Liberti. 2008. Shortest paths on dynamic graphs. *International Transactions in Operational Research* 15, 5 (2008), 551–563.

[24] José Manuel Paixão and José Luis Santos. 2013. Labeling methods for the general case of the multi-objective shortest path problem–a computational study. In *Computational Intelligence and Decision Making*. Springer, 489–502.

[25] Francisco-Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la Cruz. 2015. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research* 64 (2015), 60–70.

[26] Andrea Raith and Matthias Ehrgott. 2009. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research* 36, 4 (2009), 1299–1331.

[27] Ganesan Ramalingam and Thomas Reps. 1996. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms* 21, 2 (1996), 267–305.

[28] Line Blander Reinhardt and David Pisinger. 2011. Multi-objective and multi-constrained non-additive shortest path problems. *Computers & Operations Research* 38, 3 (2011), 605–616.

[29] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.

[30] Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. 2016. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 347–358.

[31] Peter Sanders and Lawrence Mandow. 2013. Parallel label-setting multi-objective shortest path search. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 215–224.

[32] Chi Tung Tung and Kim Lin Chew. 1992. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research* 62, 2 (1992), 203–209.

[33] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D Owens. 2016. Gunrock: A high-performance graph processing library on the GPU. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1–12.

[34] Xiaobing Wu, Guihai Chen, and Sajal K Das. 2008. Avoiding energy holes in wireless sensor networks with nonuniform node distribution. *IEEE Transactions on parallel and distributed systems* 19, 5 (2008), 710–720.