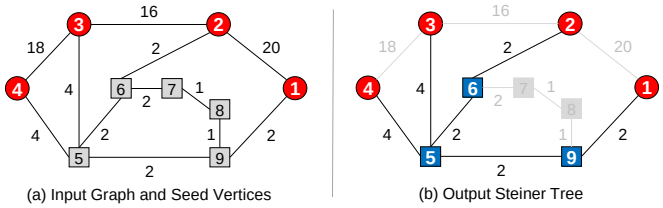
Graphical Abstract

Distributed Approximate Minimal Steiner Trees with Millions of Seed Vertices on Billion-Edge Graphs

Tahsin Reza, Trevor Steil, Geoffrey Sanders, Roger Pearce



Highlights

Distributed Approximate Minimal Steiner Trees with Millions of Seed Vertices on Billion-Edge Graphs

Tahsin Reza, Trevor Steil, Geoffrey Sanders, Roger Pearce

- Voronoi cell computation is an enabler for scaling to large graphs and seed sets.
- Message prioritization accelerates convergence of distance computation.
- Combining vertex- and edge- centric processing can bring forth superior performance.
- 2-approx. MST can preserve the theoretical bound of a 2-approx. Steiner tree algo.

Distributed Approximate Minimal Steiner Trees with Millions of Seed Vertices on Billion-Edge Graphs

Tahsin Reza*, Trevor Steil, Geoffrey Sanders, Roger Pearce

[a] *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, 94550, CA, USA*

Abstract

In this paper, we present a parallel 2-approximation Steiner minimal tree algorithm and its MPI-based distributed implementation. In place of expensive distance computations between all pairs of seed vertices, the solution we employ exploits a cheaper *Voronoi cell* computation. Our design leverages asynchronous processing and message prioritization to accelerate convergence of distance computations, and harnesses vertex and edge centric processing to offer fast time-to-solution. We demonstrate scalability and performance using real-world graphs with up to 128 billion edges and 512 compute nodes, and show the ability to find Steiner trees with up to one million seed vertices. Using 12 data instances, we present comparison with the state-of-the-art exact solver, SCIP-Jack, and two sequential 2-approximate algorithms. We empirically show that, on average, the total distance of the Steiner tree identified by our solution is 1.1290 times greater than the Steiner minimal tree – well within the theoretical approximation bound of 2.

Keywords: distributed computing, hpc, asynchronous processing,

*Corresponding author

Email addresses: reza2@llnl.gov (Tahsin Reza), steil1@llnl.gov (Trevor Steil), sanders29@llnl.gov (Geoffrey Sanders), rpearce@llnl.gov (Roger Pearce)

distributed graph processing, parallel graph algorithms, steiner tree
PACS: 0000, 1111
2000 MSC: 0000, 1111

---

1. Introduction

A network is often represented by a *distance-weighted* graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, d)$, with data entities represented by vertices $\mathcal{V}$, their relationships represented by edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, and distances $d : \mathcal{E} \to [1, \infty)$. In this work, smaller weights represent stronger relationships (or closer distances between the underlying data entities). For $\{u, v\} \in \mathcal{V}$, the *weight* or *distance* of $(u, v) \in \mathcal{E}$ is written $d(u, v)$, and for a knowledge network, is often a function of the metadata living on u, v, (u, v) and the relationship type of (u, v). The total weight or total distance of a set of edges associated with a subgraph $\mathcal{H}$ (e.g., a cluster, a tree or a path) is the sum of the distances $D(\mathcal{H}) = \sum_{(u,v)\in\mathcal{H}} d(u, v)$. For $\{u, v\} \in \mathcal{V}$ and $(u, v) \notin \mathcal{E}$, $d_1(u, v)$ is the shortest total distance of any path from u to v.

We call the user's entities of interest the *seed* vertices, $\mathcal{S} \subset \mathcal{V}$. Various applications need different magnitudes of seed set sizes. The goal is to compute a tree subgraph $\mathcal{T}$ that minimally connects all vertices in $\mathcal{S}$, preferring edges that have low distance over those that are larger – known in the literature as the *Steiner minimal tree* problem (1).

Given a set of seed vertices $\mathcal{S}$ (also called *terminal* vertices in the literature), a Steiner tree $\mathcal{T}(\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, d)$ of $\mathcal{G}(\mathcal{V}, \mathcal{E}, d)$ is an acyclic subgraph that connects all of $\mathcal{S}$. A Steiner tree $\mathcal{T}$ is called a Steiner minimal tree if its total distance $D(\mathcal{T}) = \sum_{e\in\mathcal{E}_\mathcal{T}} d(e)$ is minimal among all Steiner trees for

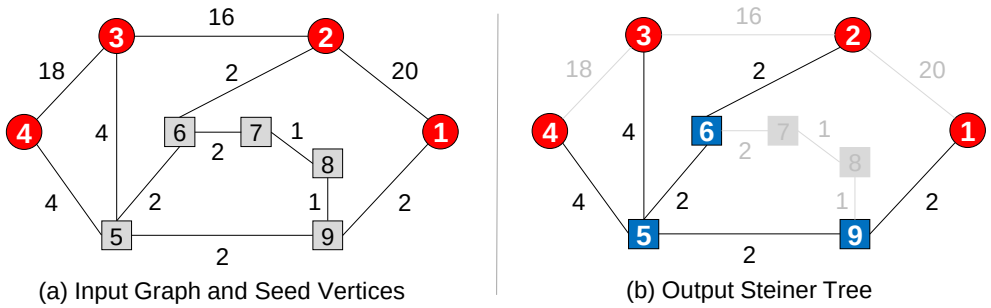


Figure 1: (a) A data graph $\mathcal{G}$, and given seed vertices $\mathcal{S}$, have red fill. (b) A Steiner tree $\mathcal{T}$ of $\mathcal{G}$. Steiner vertices $\mathcal{S}' \in \mathcal{V} \setminus \mathcal{S}$ have blue fill. Vertices and edges not in the Steiner tree $\mathcal{T}$ have light grey outline and fill.

$\mathcal{G}$ and $\mathcal{S} \subset \mathcal{V}$. In the literature, the vertex set $\mathcal{S}' = \mathcal{V}_{\mathcal{T}} \setminus \mathcal{S}$ is commonly refereed as *Steiner vertices*. A Steiner vertex may not be required to be in a Steiner tree but its inclusion may lower the total distance (2; 3; 1). Fig. 1 shows an example. Gilbert and Pollack (2) are often credited for formalising the Steiner minimal tree problem which has direct applications to VLSI design (4; 5), communication network optimization (6; 7), computational and systems biology (8; 9), and knowledge discovery and management (10; 11). The problem of finding a Steiner minimal tree of a graph and arbitrary seed vertices is known to be NP-hard (the decision variant is NP-complete) (1; 12). Therefore, polynomial time algorithms for finding a Steiner tree $\mathcal{T}$ with a total distance $D(\mathcal{T})$ close to the total distance of a Steiner minimal tree $D_{min}(\mathcal{G})$ are sought due to their practical relevance. There has been continuing interest in practical polynomial-time solutions with tight approximation bounds. Takahashi et al. (13) presents an algorithm with the approximation bound $D(\mathcal{T})/D_{min}(\mathcal{G}) \leq 2(1 - 1/|\mathcal{S}|)$. The algorithm by Kou et al. (14) (known as the KMB algorithm) improves the

bound to $D(\mathcal{T})/D_{min}(\mathcal{G}) \leq 2(1 - 1/l)$ where l is the minimum number of leaves in any Steiner minimal tree for $\mathcal{G}$ and $\mathcal{S}$. A corpus of algorithms capitalize over the KMB algorithm and offer improved sequential runtime complexity while preserving the 2-approximation bound: algorithms by Wu et al. (15) (known as the WWW algorithm), Widmayer (16) and Mehlhorn (17) are some of the most well known 2-approximation solutions.

Despite its rich theoretical literature, work related to scalable implementations of parallel Steiner minimal tree computations is rather scarce. In (18), we presented a parallel 2-approximation Steiner minimal tree algorithm and its MPI-based distributed implementation. The solution is based on computing *Voronoi cells* similar to the algorithm in (17): Mehlhorn replaces the most computationally expensive task in the KMB algorithm, i.e., computing all-pairs-shortest-paths (APSP) between the vertices in $\mathcal{S}$, with a Voronoi cell computation of each $s \in \mathcal{S}$. This solution approach stems from the observations that, in practice, the Voronoi cell approach is less expensive then APSP computation (see Table 1), and has higher parallel efficiency than minimum spanning tree (MST) computation on the entire graph (19) (that the WWW and Widmayer algorithms do). In (18), we demonstrated, scalability and performance of this distributed solution using eight real-world graphs with up to 128 billion edges and up to 512 compute nodes (8K processes), achieving up to 90% efficient strong scaling that, to the best of our knowledge, is the largest scale to date for the Steiner tree problem. We showed the ability to find Steiner trees with up to 10K seed vertices in under one minute. On average, the total distance $D(\mathcal{T})$ of the Steiner tree identified by our solution in (18) is 1.0527 times greater than the Steiner minimal tree –

Table 1: Runtime comparison of all-pairs-shortest-path (APSP) and Voronoi cell (VC) computation using two graphs and three seed sets. All experiments use a single thread. Dataset details are in §5, Table 4.

$\|S\|$	10		100		1000	
	APSP	VC	APSP	VC	APSP	VC
LVJ	49.7s	30.0s	539.2s	35.1s	5,813.3s	104.5s
PTN	26.7s	12.9s	270.3s	26.6s	2,767.4s	85.5s

well within the theoretical bound.

In this paper, we report recent advancements in our distributed 2-approximate Steiner minimal tree solution and their experimental demonstration. To distinguish the two solution approaches, in the remainder of the paper, we refer to the work presented in (18) as D-1, and call the new solution D-2. The extended work D-2 addresses key limitations of our earlier effort in D-1: in (18) we mentioned that, on our testbed (§5), going from 1K seed to 10K seeds, we had to double the number of compute nodes to meet increased memory demands of D-1. The high memory usage is due to large message buffers required to perform MPI collective operations on the distance graph (§2). The issue is compounded by the design choice to compute the exact MST of the distance graph using a serial algorithm (locally on a compute node). Since the size of the distance graph grows proportionally with the number of seeds, D-1's design offers limited scalability in that regard. D-2 addresses each of these limitations of D-1 and embraces a fully scale out design that imposes no limitations on the size of the graph dataset and number of seed vertices it can support. Below we summarize the new contributions in this paper.

(i) We present a memory efficient scale out solution for 2- and 2β-approximate Steiner minimal tree computation. Similar to D-1 (18), the extended solution D-2 is also based on Voronoi cell identification for distance computation. However, D-2 significantly differs from D-1 with respect to distance graph construction and operations on the distance graph towards identifying the Steiner tree (§3 and §4).

(ii) D-2 replaces the serial routine in D-1 for computing exact MST of the distance graph with a distributed approximate spanning tree algorithm (A-MST) (20). This enables D-2 to accommodate larger seed sets more efficiently. We provide theoretical guarantees that the distributed A-MST approach is guaranteed to produce Steiner trees whose total distance is within a factor 2β of an actual minimal Steiner tree, where β is a configurable approximation bound on the approximate MST algorithm (§4.3 and § Appendix A).

(iii) D-1 partially relies on MPI collective operations for distance graph construction and distributing the MST of the distance graph to data partitions. With an increasing number of seeds, growing message buffers associated with MPI collective operations become the primary memory bottleneck. D-2 introduces new asynchronous distributed routines that eliminate the need for MPI collective operations, hence improving resource efficiency (§4).

(iv) We demonstrate scalability and performance using eight real-world graphs with up to 128 billion edges and up to 512 compute nodes (8K processes) (§5.4) and show the ability to find Steiner trees with up to 1M seed vertices

in under two minutes (§5.1). Using the same number of compute nodes, D-2 can identify Steiner trees for 10 seed vertices, all the way to 1M seeds – a significant improvement in memory efficiency over D-1 (§5.1 and §5.2). Using 12 data instances, we compare our solution with the state-of-the-art exact Steiner tree solver, SCIP-Jack (21), and two 2-approximation sequential algorithms, WWW (15) and Mehlhorn (17). Our distributed solution comfortably outperforms these related works on graphs with tens of million edges. On average, the total distance $D(\mathcal{T})$ of the Steiner tree identified by D-2 is 1.1290 times greater than the Steiner minimal tree $D_{min}(\mathcal{G})$, which is well within the theoretical bound of $\leq 2(1 - 1/l)$. Additionally, on average, total distance of the Steiner tree identified by D-2 is only ∼6.8% greater than the total distance of a Steiner tree identified by D-1. (§5.3).

2. Preliminaries

Table 2 lists the symbolic notation used in this paper. A *path* is a sequence of connected edges in $\mathcal{V}$ that does not go through any vertex more than once. The total distance between two vertices $i, j \in \mathcal{V}$, $d_1(i, j)$, is equal to the distance of a shortest path from i to j in $\mathcal{G}$. For every seed vertex $s \in \mathcal{S}$, the *Voronoi cell* or *Voronoi region*, $\mathcal{R}_s$, is the set of vertices in $\mathcal{V}$ that are at a shorter distance (an edge or a path) to s than to any other vertex in $\mathcal{S}$. $\mathcal{R}_s \cap \mathcal{R}_t = \emptyset$ for any $s, t \in \mathcal{S}$ and $s \neq t$, with $v \in \mathcal{R}_s \Rightarrow d_1(v, s) \leq d_1(v, t)$. An edge $(u, v) \in \mathcal{E}$ is a *cross-cell edge* if $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t$, and $s, t \in \mathcal{S}$, $s \neq t$. A cross-cell edge bridges two Voronoi cells.

Kou et al. were the first to propose an algorithm (Alg. 1) for finding

Table 2: Symbolic notation used.

Object(s)	Notation
graph, vertices, edges, distance	$\mathcal{G}(\mathcal{V}, \mathcal{E}, d)$
set of vertices adjacent to i in $\mathcal{G}$	$adj(i)$
predecessor, successor of i	$pred(i)$, $scsr(i)$
distance function	$d(i,j) \in [1, \infty)$
tree subgraph, vertices, edges	$\mathcal{T}(\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, d)$
Steiner tree	$\mathcal{S} \subset \mathcal{V}_\mathcal{T}$ and $\mathcal{T}$ is connected
minimal Steiner tree	$\mathcal{T}^\star$
seed vertices, Steiner vertices	$\mathcal{S} \subset \mathcal{V}$, $\mathcal{S}' = \mathcal{V}_\mathcal{T} \setminus \mathcal{S}$
Voronoi cell of vertex $s \in \mathcal{S}$	$\mathcal{R}_s$
source vertex of i	$src(i) = s$ if $i \in \mathcal{R}_s$
total distance of $\mathcal{T}$	$D(\mathcal{T}) = \sum_{(i,j)\in\mathcal{E}_\mathcal{T}} d(i,j)$
distance of Steiner minimal tree	$D_{min}(\mathcal{G})$
approximation ratio	$D(\mathcal{T})/D_{min}(\mathcal{G})$
all-pairs closure of $\mathcal{S}$ w.r.t $\mathcal{G}$	$\mathcal{G}_\mathcal{S}(\mathcal{S}, \mathcal{E}_\mathcal{S}, d_\mathcal{S})$
sparse representation of $\mathcal{G}_\mathcal{S}$	$\hat{\mathcal{G}}_\mathcal{S}(\mathcal{S}, \hat{\mathcal{E}}_\mathcal{S}, \hat{d}_\mathcal{S})$

a Steiner tree $\mathcal{T}$ with $D(\mathcal{T})/D_{min}(\mathcal{G}) \leq 2(1-1/l)$, where l is the minimum number of leaves in any Steiner minimal tree for $\mathcal{G}$ and $\mathcal{S}$. The most expensive step in the KMB algorithm (Alg. 1) is the all-pairs shortest path computation among the seed vertices to form $\mathcal{G}_\mathcal{S}$, the dense graph of minimal distance between all seeds. Mehlhorn (17) proposes to replace Step 1. in Alg. 1 by a cheaper alternative which is based on a Voronoi cell computation using every $s \in \mathcal{S}$. Mehlhorn improves the sequential complexity of KMB from $O(|\mathcal{S}||\mathcal{V}|^2)$ to $O(|\mathcal{V}|log|\mathcal{V}| + |\mathcal{E}|)$ with an algorithm that is more amenable to parallel and distributed computing.

Mehlhorn observes that it is possible to construct a distance graph $\hat{\mathcal{G}}_\mathcal{S}$

which is a subgraph of $\mathcal{G}_{\mathcal{S}}$ in Alg. 1, where $\hat{\mathcal{G}}_{\mathcal{S}}(\mathcal{S}, \hat{\mathcal{E}}_{\mathcal{S}}, \hat{d}_{\mathcal{S}})$ is defined by:

$\hat{\mathcal{E}}_{\mathcal{S}} = \{(s,t); s,t \in \mathcal{S}$ and $\exists (u,v) \in \mathcal{E}$ with $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t\}$ and $\hat{d}_{\mathcal{S}}(s,t) = min\left\{d_1(s,u) + d(u,v) + d_1(v,t)\right\}$, for $(u,v) \in \mathcal{E}$, with $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t$.
Note: in general, $\hat{d}_{\mathcal{S}}$ is not merely the restriction of d_1 to the set $\hat{\mathcal{E}}_{\mathcal{S}}$. Two adjacent Voronoi cells can have multiple cross-cell paths (thus cross-cell edges) connecting the corresponding seed vertex pair, as shown in Fig. 2(a). The cross-cell path with the minimum weighted distance represents an edge in the distance graph between the seed vertex pair. $\hat{\mathcal{E}}_{\mathcal{S}}$ is the edge set of the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$. $\hat{d}_{\mathcal{S}}(s,t)$ is the weight of an edge (s,t) in $\hat{\mathcal{E}}_{\mathcal{S}}$. Also, the implementation must maintain the cross-cell edge (u,v) corresponding to each minimum distance cross-cell path to later retrieve the actual path from the MST of the distance graph.

Mehlhorn proves the existence of an MST, $\mathcal{T}^{\star}_{\mathcal{S}}$, of $\mathcal{G}_{\mathcal{S}}$ that is also a subgraph of $\hat{\mathcal{G}}_{\mathcal{S}}$, that the associated distance functions $d_{\mathcal{S}}$ and $\hat{d}_{\mathcal{S}}$ agree on edges of $\mathcal{T}^{\star}_{\mathcal{S}}$, and further, that every MST of $\hat{\mathcal{G}}_{\mathcal{S}}$ is also a MST of $\mathcal{G}_{\mathcal{S}}$. This fact allows the approximation bound of the KMB algorithm to apply to Mehlhorn's algorithm as well.

3. Parallel Steiner Tree Algorithm

In this section, we introduce our parallel Steiner tree algorithm, while in §4, we discuss the distributed implementation. The algorithm produces a Steiner tree $\mathcal{T}$ with the approximation ratio $D(\mathcal{T})/D_{min}(\mathcal{G}) \leq 2(1 - 1/l)$, the same as (14; 15; 16; 17). Our solution is based computing Voronoi cells similar to the algorithm in (17). The generalized minimum spanning tree computation approach (15; 16) may appear attractive because of its simplic-

ity and greater work efficiency, however, we observe that it is possible to develop an algorithm with higher parallel efficiency for Voronoi cell computation compared to that of the MST computation. Bader et al. (19) and authors of the Galois (22) project demonstrated MST computation suffers from rapid decrease in the available parallelism (23).

Voronoi cell computation naturally lends itself to vertex-centric parallel processing (24). Computation of a single cell closely resembles single source shortest path (SSSP) computation, and fast, Bellman-Ford-based, vertex parallel SSSP algorithms are well known (24; 25). It is worth noting that Ceccarello et al. (26) used the work-efficient Δ-Stepping algorithm in parallel shortest path computation from multiple sources for diameter approximation of weighted graphs (comparable to Voronoi cell computation). We, however, base our distributed implementation of Voronoi cell computation on Bellman-Ford's algorithm which can harness asynchronous distributed processing (i.e.,

Algorithm 1 KMB Algorithm (14)

Input: edge-weighted graph $\mathcal{G}$, seed vertices $\mathcal{S}$

Output: Steiner tree $\mathcal{T} = \mathcal{G}_5$.

1: Construct the distance graph $\mathcal{G}_{\mathcal{S}}(\mathcal{V}_{\mathcal{S}}, \mathcal{E}_{\mathcal{S}}, d_{\mathcal{S}})$ where $\mathcal{V}_{\mathcal{S}} = \mathcal{S}$ and, for every $(i, j) \in \mathcal{E}_{\mathcal{S}}$, $d_1(i, j)$ is equal to the distance of a shortest path from i to j in $\mathcal{G}$.

2: Find MST $\mathcal{T}^{\star}_{\mathcal{S}}$ of $\mathcal{G}_{\mathcal{S}}$.

3: Construct a subgraph $\mathcal{G}_3$ of $\mathcal{G}$ by replacing each edge in $\mathcal{G}_2$ by (one of) the corresponding shortest path(s) in $\mathcal{G}$.

4: Find MST $\mathcal{G}_4$ of $\mathcal{G}_3$.

5: Construct a Steiner tree $\mathcal{G}_5$ from $\mathcal{G}_4$ by deleting edges in $\mathcal{G}_4$, if necessary, so that no leaves in $\mathcal{G}_5$ are Steiner vertices.

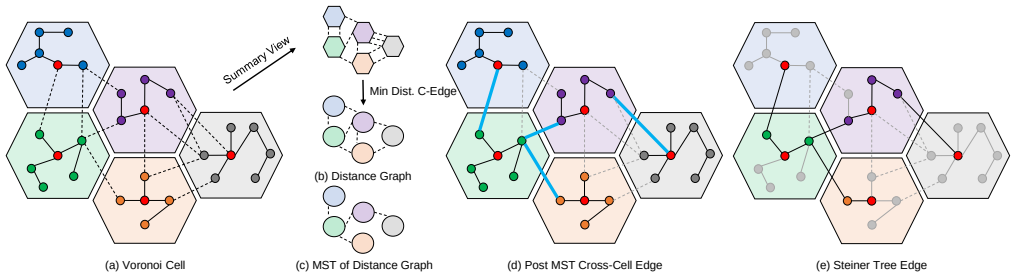


Figure 2: High-level illustration of output of key algorithm steps: (a) Each hexagon represents the Voronoi cell of a seed vertex (solid red fill); cross-cell edges are shown using broken lines. (b) The distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$. (c) MST $\hat{\mathcal{T}}_{\mathcal{S}}^{\star}$ of $\hat{\mathcal{G}}_{\mathcal{S}}$. (d) Post MST cross-cell edges: "deleted" cross-cell edges have light grey outline; "active" cross-cell edges (corresponding edges of MST $\hat{\mathcal{T}}_{\mathcal{S}}^{\star}$ in $\mathcal{G}$) are shown using blue outline. (e) The vertices and edges in the final Steiner tree.

overlapping latency prone distributed communication with computation) and accelerate the convergence rate (details in §4). Recently, Wang et al. (27) presented a shared memory, asynchronous SSSP implementation for a single Nvidia GPU; the solution is an adaptation of the iterative Δ-Stepping algorithm. The technique, however, does not naturally extend to distributed memory. Our parallel algorithm computes all Voronoi cells in parallel – comparable to running $|\mathcal{S}|$ parallel instances of a parallel SSSP algorithm on the same graph. Alg. 2 presents a high-level overview of the proposed parallel algorithm and Fig. 2 illustrates the key algorithmic steps. D-1 in (18) and D-2 in Alg. 3 are shared-nothing distributed implementations of the parallel algorithm in Alg. 2

Step 1. in Alg. 2 computes Voronoi cells in parallel based on Bellman-Ford's SSSP algorithm. For given $\mathcal{G}$ and $\mathcal{S}$, Fig. 2(a) shows the computed Voronoi cells. Cross-cell edges are shown using broken lines. **Step 2.** con-

structs the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$. Two cells can have multiple cross-cell edges, as shown in Fig. 2(b). A parallel routine identifies unique minimum distance edges, each connecting a pair of cells (a tie-breaking scheme is used to ensure uniqueness). The distance information of these cross-cell edges determines the edge weights of $\hat{\mathcal{G}}_{\mathcal{S}}$. **Step 3.** generates $\hat{\mathcal{T}}^{\star}_{\mathcal{S}}$, an MST of $\hat{\mathcal{G}}_{\mathcal{S}}$ (Fig. 2(c)). Often $\hat{\mathcal{G}}_{\mathcal{S}}$ is small, having at most $\binom{|\mathcal{S}|}{2}$ edges. Even for $|\mathcal{S}| = 1$K, there are less than 500K edges, orders of magnitude smaller than the billion-edge data graphs we used for evaluation in §5. When the distance graph is small, a serial routine is sufficient for computing the MST $\hat{\mathcal{T}}^{\star}_{\mathcal{S}}$. Given the problem size

Algorithm 2 A Parallel Steiner Tree Algorithm based on Mehlhorn (17)

Input: edge-weighted graph $\mathcal{G}$, seed vertices $\mathcal{S}$

Output: Steiner tree $\mathcal{T}$, total distance $D(\mathcal{T})$

1: In parallel, for every $s \in \mathcal{S}$, compute Voronoi cell $\mathcal{R}_s$. Every $u \in \mathcal{V}$, $u \in \mathcal{R}_s$, maintains states $src(u)$, $pred(u)$, $d_1(s,u)$.

2: Construct the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}(\mathcal{S}, \hat{\mathcal{E}}_{\mathcal{S}}, \hat{d}_{\mathcal{S}})$: In parallel, for every $(s,t) \in \hat{\mathcal{E}}_{\mathcal{S}}$, compute $\hat{d}_{\mathcal{S}}(s,t) = min(d_1(s,u) + d(u,v) + d_1(v,t))$, where $s,t \in \mathcal{S}$, $s \neq t$ and $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t$.

3: Find MST $\hat{\mathcal{T}}^{\star}_{\mathcal{S}}$ of $\hat{\mathcal{G}}_{\mathcal{S}}$ (using a serial or parallel MST algorithm).

4: In parallel, for every "active" cross-cell edge $(u,v) \in \mathcal{E}$ (i.e., the edge is present in the MST $\hat{\mathcal{T}}^{\star}_{\mathcal{S}}$), where $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t$ and $s \neq t$, identify all edges in the shortest path from u to s, and v to t, by following respective predecessor vertices stored in $pred$.

5: $\mathcal{T}(\mathcal{V}_{\mathcal{T}}, \mathcal{E}_{\mathcal{T}}, d)$ is the final Steiner tree where $\mathcal{E}_{\mathcal{T}}$ only includes "active" cross-cell edges in $\mathcal{G}$, and edges identified in Step 5.

6: Compute $D(\mathcal{T}) = \sum_{e \in \mathcal{E}_{\mathcal{T}}} d(e)$.

is often small, attempting to parallelize MST computation is likely to yield marginal or no gain at all ((19)). However, a parallel or distributed solution would be more appropriate for accommodating large seed sets. **Step 4.** in parallel, starting from each vertex of every "active" cross-cell edge (present in the MST $\hat{\mathcal{T}}_{\mathcal{S}}^{\star}$, shown using blue lines in Fig. 2(d)), identifies all the edges in the shortest path from $u \in \mathcal{R}_s$ to $s \in \mathcal{S}$ by following predecessor vertices identified during Voronoi cell computation. The remaining cross-cell edges after edge pruning in Step 4. and the edges identified in Step 5. form a valid Steiner tree, as shown in Fig. 2(e).

The 2-approximation bound $D(\mathcal{T})/D_{min}(\mathcal{G}) \leq 2(1-1/l)$ of our algorithm is guaranteed by Mehlhorn's proof (17) that every MST of the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$ in Alg. 2 is also a MST of the distance graph $\mathcal{G}_{\mathcal{S}}$ in Alg. 1, which allows the 2-approximation bound of the KMB algorithm to apply to the algorithm in Alg. 2.

4. Distributed Solution Design and Implementation

We present a distributed design of the parallel Steiner tree algorithm in §3, Alg. 2 and its MPI-based distributed implementation. To accommodate large-scale graphs with tens to hundreds of billion edges, we embrace a scale out design – partition the data graph for processing on shared-nothing distributed platforms. Each partition is assigned to an MPI process which enables processing the partitions in parallel. The solution employs a combination of vertex and edge centric processing, asynchronous processing, and a message prioritization scheme to achieve fast time-to-solution and resource efficiency.

4.1. Motivation behind Design Choices

The Voronoi cell computation is the most expensive step in our algorithm (asymptotic time complexity is presented in §4.4). Therefore, we seek a solution approach that accelerates the throughput of this step. Previous studies (28; 25) showed that asynchronous processing offers notable advantages over bulk synchronous processing (BSP) for distributed shortest path computation, the former enabling faster convergence. To this end, we begin with HavoqGT (29) as the foundation and implement other features required by the distributed solution presented in Alg. 3. Our choice for HavoqGT is motivated by a number of considerations: HavoqGT supports asynchronous processing, where latency prone communication can be overlapped with computation. In HavoqGT, graph partitions have approximately equal numbers of vertices; and offers further load balancing for scale-free graphs through vertex-cut partitioning by distributing edges of high-degree vertices across multiple partitions – crucial for scaling to large graphs with skewed degree distributions. In HavoqGT, algorithms are implemented as vertex-callbacks: the user-defined $visit()$ callback accesses and updates a vertex's state(s). HavoqGT generates events, called a VISITOR, that invoke $visit()$ callbacks – either on all graph vertices using the $do_traversal(arguments, traversal_type \leftarrow init_all)$ method, or for a neighboring vertex using the $push$(VISITOR) method. This enables asynchronous data exchange between graph vertices. The graph computation completes when all visitors in the message queue have been processed (29).

4.2. Design Improvements over the Prior Distributed Solution

To address resources inefficiencies (discussed in §1) in our earlier solution D-1 (18), the new solution D-2 introduces two key design changes: *(i)* D-2 substitutes all MPI collective operations with asynchronous distributed routines for constructing the distance graph from Voronoi cells. With an increasing number of seeds, growing message buffers associated with MPI collective operations become the primary memory bottleneck in D-1. *(ii)* D-2 replaces the serial MST routine in D-1 with a distributed approximate spanning tree (A-MST) solution. This enables D-2 to accommodate larger seed sets more efficientsly on shared-nothing platforms.

4.3. Design of the Improved Distributed Solution

In a distributed setting, each process runs an instance of Alg. 3 which iterates over a number of procedures to produce the final 2-approximation Steiner minimal tree. The INITIALIZATION procedure allocates per-partition memory for distributed vertex states and initializes them (lines 6–10). The distributed map $\mathcal{E}_N$ stores the edges of the graph that are associated with the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$. The STEINER_TREE procedure invokes the key Steiner tree computation kernels (related to the steps in Fig. 2) and output results. The following subsections describe the distributed design of these key steps.

VORONOI_CELL_ASYNC. Voronoi Cells are computed asynchronously, initiated by invoking $do_traversal()$. Alg. 4 presents the corresponding $visit()$ procedure in HavoqGT's vertex-centric abstraction; lines 15–18 list the VISITOR states. Initially, only a vertex $s \in \mathcal{S}$ visits its neighbors (line 5). In general, when a vertex v_j with $src(v_j) = s$ is visited by another vertex v_p with

$src(v_p) = t$ such that $s \neq t$, if $d_1(s, v_j) > d(v_j, v_p) + d_1(t, v_p)$ then v_j, becomes a member of the Voronoi cell $\mathcal{R}_t$, adopts v_p as its predecessor, updates its distance accordingly (line 8), and notifies its neighbors about the state update (line 11).

Algorithm 3 Distributed Steiner Tree Algorithm (D-2)

Input: edge-weighted graph $\mathcal{G}$, seed vertices $\mathcal{S}$

Output: Steiner tree $\mathcal{T}$, total distance $D(\mathcal{T})$

1: **procedure** INITIALIZATION
2: each $s \in \mathcal{S}$ is assigned a unique ID s' in the range $[0, |\mathcal{S}|)$, which is its vertex identifier in the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$
3: $\mathcal{T}(\mathcal{V}_{\mathcal{T}} = \emptyset, \mathcal{E}_{\mathcal{T}} = \emptyset, d_{\mathcal{T}})$
4: $\mathcal{E}_N \leftarrow \emptyset$ ▷ distributed map of min. distance cross-cell edges, represents $\hat{\mathcal{G}}_{\mathcal{S}}$
5: $\mathcal{E}'_N \leftarrow \emptyset$ ▷ partition local map of min. distance cross-cell edges
6: **for all** $v \in \mathcal{V}$ **do**
7: **if** $v \in \mathcal{S}$ **then**
8: $src(v) \leftarrow v$; $pred(v) \leftarrow v$; $d_1(src(v), v) \leftarrow 0$
9: **else**
10: $src(v) \leftarrow \infty$; $pred(v) \leftarrow \infty$; $d_1(src(v), v) \leftarrow \infty$
11: **procedure** STEINER_TREE
12: VORONOI_CELL_ASYNC($\mathcal{G}, \mathcal{S}$); **barrier**
13: LOCAL_MIN_DIST_C-EDGE_ASYNC($\mathcal{G}, \mathcal{S}, \mathcal{E}'_N$); **barrier**
14: GLOBAL_MIN_DIST_C-EDGE_ASYNC($\mathcal{E}'_N, \mathcal{E}_N$); **barrier**
15: $\hat{\mathcal{T}}_{\mathcal{S}} \leftarrow$ DISTRIBUTED_A-MST($\mathcal{E}_N$); **barrier**
16: TREE_EDGE_ASYNC($\mathcal{G}, \hat{\mathcal{T}}_{\mathcal{S}}, \mathcal{T}$); **barrier**
17: **return** $\mathcal{T}$, $D(\mathcal{T}) = \sum_{e \in \mathcal{E}_{\mathcal{T}}} d_{\mathcal{T}}(e)$

To accelerate convergence of shortest path computations in Voronoi cells, we employ a message prioritization technique in the message queue of each graph partition (vq in line 3) which gives precedence to a message from a vertex at a lower distance (i.e., the value of the VISITOR state r). This can produce a similar effect of the min-priority queue in Dijkstra's serial algorithm, enabling the Bellman-Ford-based distributed shortest path ker-

Algorithm 4 Voronoi Cell

1: **procedure** VORONOI_CELL_ASYNC $(\mathcal{G}, \mathcal{S})$
2: $\quad do_traversal(r \leftarrow 0, traversal_type \leftarrow init_all)$
3: **procedure** VISIT$(\mathcal{G}, \mathcal{S}, vq)$ ▷ vq – (priority) message queue
4: $\quad do_update \leftarrow false$
5: $\quad$ **if** $d_1(src(v_j), v_j) = 0$ and $d_1(src(v_j), v_j) = r$ **then**
6: $\quad\quad t \leftarrow src(v_j)$; $r \leftarrow d_1(src(v_j), v_j)$; $do_update \leftarrow true$
7: $\quad$ **else if** $0 < r < d_1(src(v_j), v_j)$ **then**
8: $\quad\quad src(v_j) \leftarrow t$; $pred(v_j) \leftarrow v_p$; $d_1(src(v_j), v_j) \leftarrow r$
9: $\quad\quad do_update \leftarrow true$
10: $\quad$ **if** $do_update = true$ **then**
11: $\quad\quad$ **for all** $v_i \in adj(v_j)$ **do**
12: $\quad\quad\quad r \leftarrow r + d(v_i, v_j)$
13: $\quad\quad\quad vq.push$(VORONOI_CELL_VISITOR(v_i, v_j, t, r))
14: VORONOI_CELL_VISITOR
15: v_j – vertex that is being visited
16: v_p – vertex that sent the visitor to v_j
17: t – current source vertex of v_p
18: r – current distance of v_p from t

nel to potentially converge faster. The technique is light-weight and best-effort only; its effectiveness depends on timeliness of asynchronous message propagation within the network which is nondeterministic. The worst-case complexity is the same as that of Bellman-Ford's algorithm, however, even with message prioritization, it is not guaranteed the best-case complexity will match that of Dijkstra's algorithm. In §5.5, we demonstrate the benefits of using a priority message queue over a FIFO queue in practice.

GLOBAL_MIN_DIST_C-EDGE_ASYNC. Towards creating the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$, minimum distance cross-cell edges between Voronoi cells are identified. This is a hierarchical reduction procedure. First, the LO-

Algorithm 5 Minimum Distance Cross-Cell Edge

procedure LOCAL_MIN_DIST_C-EDGE_ASYNC $(\mathcal{G}, \mathcal{S}, \mathcal{E}'_N)$
    for every $(u, v) \in \mathcal{E}$ local to a partition of $\mathcal{G}$, computes
    $\hat{d}_{\mathcal{S}}(s, t) = min(d_1(s, u) + d(u, v) + d_1(v, t))$, where
    $u \in \mathcal{R}_s$, $v \in \mathcal{R}_t$, $s \neq t$; adds each corresponding
    min. distance cross-cell edge (u, v) to the local map, i.e.,
    $\mathcal{E}'_N : (s', t') \leftarrow (u, v, \hat{d}_{\mathcal{S}}(s, t))$, where $s' < t' \triangleright s'$ and t' are vertex identifiers of s and t, respectively, in the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$
procedure GLOBAL_MIN_DIST_C-EDGE_ASYNC $(\mathcal{E}'_N, \mathcal{E}_N)$
    each partition attempts to insert contents of
    the local map $\mathcal{E}'_N$ in the distributed map $\mathcal{E}_N$;
    for a *new* map insert request, if key (s', t')
    already exists in $\mathcal{E}_N$, the value of (s', t') is
    updated only if $\hat{d}_{\mathcal{S}}$ of the *new* request is smaller
    than the current $\hat{d}_{\mathcal{S}}$ of the key (s', t')

CAL_MIN_DIST_C-EDGE_ASYNC (Alg. 5) procedure identifies minimum distance cross-cell edges local to a partition and adds them to the local map $\mathcal{E}'_N$. This is an asynchronous procedure based on HavoqGT's vertex-centric abstraction. The distance information of a vertex computed in Alg. 4 is stored locally. To compute $\hat{d}_{\mathcal{S}}(s,t)$ (Alg. 5, line 3), a vertex u needs to receive $d_1(v,t)$ from v, often from a remote partition; hence, inter-vertex communication is required. GLOBAL_MIN_DIST_C-EDGE_ASYNC (Alg. 5) is an edge-centric procedure, operates on a distributed map which supports asynchronous updates: from the local minimum distance cross-cell edges stored in $\mathcal{E}'_N$, it identifies the global minimum distance cross-cell edges and store them in the distributed map $\mathcal{E}_N$. When this routine completes, $\mathcal{E}_N$ contains the edges of the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$ in distributed memory.

DISTRIBUTED_A-MST. The 2-approximate minimum spanning tree (A-MST) $\hat{\mathcal{T}}_{\mathcal{S}}$ of the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$ is computed using a YGM-based distributed approximate minimum spanning tree algorithm (Alg. 3, line 15). The DISTRIBUTED_A-MST, an edge-centric routine, ingests the distance graph from distributed memory edge list $\mathcal{E}_N$ and outputs a spanning tree $\hat{\mathcal{T}}_{\mathcal{S}}$, also, stored in distributed memory.

YGM (30; 31) is a communication optimized C++ library for developing distributed applications. The distributed A-MST is implemented using YGM's distributed disjoint set computation kernel (20). The A-MST algorithm offers configurable approximation bounds; the spanning tree returned is guaranteed to have a weight within a multiplicative factor of β for the provided parameter $\beta > 1$. A-MST identifies a spanning tree in a number of bulk synchronous rounds. In each round, computation follows asynchronous

processing. The number of bulk synchronous rounds required depends on the desired approximation bound; a stricter bound leads to a higher number of bulk synchronous rounds, thus, making it costly in terms of processing time. The distributed A-MST approach offers a unique opportunity for scalable 2-approximate Steiner minimal tree computation: in Appendix A, we provide theoretical guarantees that the distributed A-MST approach (for computing a spanning tree of the distance graph) is guaranteed to produce Steiner trees whose total distance is within a factor 2β of an actual minimal Steiner tree, where β is the approximation bound chosen for the A-MST algorithm.

Algorithm 6 Steiner Tree Edge

```
1: procedure TREE_EDGE_ASYNC (G, T̂_S, T)
2:     for all (u, v) → (s', t') ∈ T̂_S do
3:         E_T ← E_T ∪ (u, v); d_T(u, v) ← d(u, v)
4:         vq.push(TREE_EDGE_VISITOR(u))          ▷ vq – message queue
5:         vq.push(TREE_EDGE_VISITOR(v))
6:     do_traversal()
7: procedure VISIT(G, T, vq)
8:     if v_j ≠ src(v_j) then
9:         E_T ← E_T ∪ (pred(v_j), v_j)
10:        d_T(pred(v_j), v_j) ← d(pred(v_j), v_j)
11:        if pred(v_j) ≠ src(v_j) then
12:            vq.push(TREE_EDGE_VISITOR(pred(v_j))
13: TREE_EDGE_VISITOR
14: v_j – vertex that is being visited
```

TREE_EDGE_ASYNC. Edges in the final Steiner tree are identified

using a vertex-centric asynchronous routine (Alg. 6). Starting from each vertex u and v of every cross-cell edge $(u, v) \in \mathcal{E}$ present in the A-MST $\hat{\mathcal{T}}_{\mathcal{S}}$, tree edges within their respective Voronoi cells are identified by recursively visiting predecessors until the source vertex, e.g., $s : u \in \mathcal{R}_s$, has been reached. Note that the cross-cell edges in $\hat{\mathcal{T}}_{\mathcal{S}}$ also belong to the final Steiner tree $\mathcal{T}$ (line 3). Alg. 6 significantly reduces the number of messages communicated since often the number of Steiner tree edges $\mathcal{E}_{\mathcal{T}}$ is orders of magnitude smaller than number of non-tree edges, i.e., $|\mathcal{E}| - |\mathcal{E}_{\mathcal{T}}|$ (see Table 6 for empirical evidence).

4.4. Asymptotic Computational Complexity

Table 3 lists asymptotic computational complexity of each step (line 12–16) of the distributed Steiner tree algorithm D-2 (Alg. 3). The Voronoi cell (Alg. 4) computation based on Bellman-Ford's algorithm requires $O(|\mathcal{V}||\mathcal{E}|)$ time. The edges of the distance graph and their respective weights are identified in two steps: in LOCAL_MIN_DIST_C-EDGE_ASYNC (Alg. 5), a vertex communicates its weight to remote neighbors and each partition locally identifies minimum distance cross-cell edges by adding them to a local map. This step requires $O(|\mathcal{V}| + |\mathcal{E}|log|\mathcal{E}|)$ time – the cost of adding a single edge to a map is $O(log|\mathcal{E}|)$. From the cross-cell edges stored in partition local maps, using a distributed map, GLOBAL_MIN_DIST_C-EDGE_ASYNC (Alg. 5) computes globally unique minimum distance cross-cell edges, which requires $O(|\mathcal{E}|log|\mathcal{E}|)$ time. The time complexity of finding the 2-approximate MST (§Appendix A) of the distance graph is $O(|\hat{\mathcal{E}}_{\mathcal{S}}|log|\mathcal{S}|)$. Starting from the edges in the MST of the distance graph, identifying final Steiner tree edges (Alg. 6) requires no more than $O(|\mathcal{E}|)$ time.

Table 3: Asymptotic computational complexity of different steps of the proposed 2-approximate Steiner minimal tree algorithm (D-2).

Voronoi Cell	Local Min. Dist. Cross-cell Edge	Global Min. Dist. Cross-cell Edge	2-Approximate MST	Steiner Tree Edge
$O(\|\mathcal{V}\|\|\mathcal{E}\|)$	$O(\|\mathcal{V}\| + \|\mathcal{E}\|log\|\mathcal{E}\|)$	$O(\|\mathcal{E}\|log\|\mathcal{E}\|)$	$O(\|\hat{\mathcal{E}}_{\mathcal{S}}\|log\|\mathcal{S}\|)$	$O(\|\mathcal{E}\|)$

5. Evaluation

We demonstrate the ability of the extended solution to support up to one million seeds (§5.1), compare performance with the prior distributed solution, §5.2), measure result quality (§5.3), present strong scaling experiments using billion-edge, real-world graphs and up to 512 compute nodes (§5.4); evaluate the effectiveness of our design choices and optimizations (§5.5); and study sensitivity of our solution to problem parameters (§5.6, §5.7). Additional experimental results are available in (18).

Testbed. The testbed is Quartz at the Lawrence Livermore National Laboratory, a 2.6 petaFLOP cluster comprised of over 2K compute nodes using an Intel Omni-Path interconnect. Each node has two 18-core Intel Xeon E5-2695v4 @2.10GHz processors and 128GB of memory (32). All experiments uses 16 MPI processes per node (as observed, this configuration offers the best performance, since each process runs two threads).

Graph Datasets. Table 4 summarizes the main characteristics of the real-world datasets used for evaluation and shows their storage requirements in HavoqGT's binary graph format. The Web Data Commons 2012 (WDC), ClueWeb 2012 (CLW), and UK Web 2007-05 (UKW) are web graphs whose vertices are webpages and edges are hyperlinks (22). Friendster (FSR) (22)

Table 4: Characteristics of graph datasets used for evaluation.

	$\|\mathcal{V}\|$	$2\|\mathcal{E}\|$	Max. degree	Avg. degree	Edge weight	Size
WDC12	3.5B	257B	95M	72.3	[1, 500K]	5.7TB
ClueWeb12	978M	85B	75.6M	87	[1, 100K]	1.9TB
UKWeb07	105M	7.5B	975K	71	[1, 75K]	150GB
Friendster	66M	3.6B	5.2K	55.1	[1, 50K]	84GB
LiveJournal	4.8M	85.7M	20.3K	17.7	[1, 5K]	2.1GB
Patent	2.7M	28M	789	10.2	[1, 5K]	692MB
MiCo	100K	2.2M	1.4K	22	[1, 2K]	52MB
CiteSeer	3.3K	9.4K	99	3.6	[1, 1K]	328KB

and LiveJournal (LVJ) (33) are user-centric social media graphs. Patent (PTN) and CiteSeer (CTS) are citation graphs (33). MiCo (MCO) is a co-author graph created from the Microsoft research article repository (33). For each graph, we create symmetric edges ($2|\mathcal{E}|$ edges) with non-zero, positive edge weights in the range as listed in Table 4. The real-world applications of Steiner tree computation discussed in §1 commonly operate on graphs with non-negative edge weights. Both DIMACS (34) ans PACE (35) Steiner tree computation challenges used graph datasets with non-negative edge weights. In §5.7, we study how different edge weights influence runtime performance of Steiner tree computation.

Seed Vertex Selection. To ensure all seed vertices are present in the Steiner tree, we choose all seeds to be from the largest connected component of each graph. If a large number of seed vertices are directly connected, the Voronoi cell computation can potentially converge faster, altering performance characteristics. To avoid this scenario, we randomly select seed ver-

tices from different BFS levels, with a higher percentage of vertices selected from a level with more vertices. Note that the length of the weighted shortest path between a vertex pair is unlikely to be the same as in the BFS-tree. Also, sampling disjoint sets by the population size roughly converges to uniform random overall. In §5.6, we explore alternative seed selection methods, and study their impact on performance and characteristics of corresponding Steiner trees.

Experiment Methodology. Performance is measured by the time to identify Steiner tree edges. It does not including graph partitioning and loading times. Runtime numbers are averages over at least 10 runs. For strong scaling experiments, the smallest platform scale is the one that can accommodate a dataset and required algorithm states. Strong scaling experiments do not include results on a single node since this does not involve remote communication.

5.1. Number of Seed Vertices vs Runtime Performance

Solution D-2 addresses the high memory demands of D-1; this limitation prevents D-1 from accommodating a large number of seed vertices efficiently. D-2's fully scale out design eliminates D-1's shortcomings – D-2 does not impose any limitation on the size of the graph dataset and number of seed vertices it can efficiently support. Fig. 3 compares runtime performance of different number of seed verices, between one thousand and one million, using six graph datasets. The maximum number of seeds used in this paper (one million) is two orders of magnitude larger than the maximum number of seed vertices used in our previous work (18). For each dataset, the number of processes used are the same for all values of $|\mathcal{S}|$. Runtime is broken down

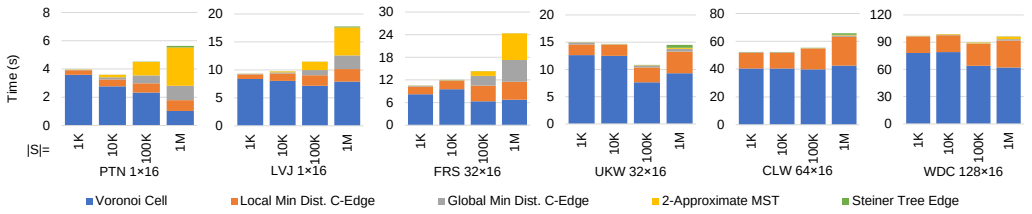


Figure 3: Runtime performance of different number of seed vertices $|\mathcal{S}|$: 1K, 10K, 100K and 1M, using six graph datasets. For each dataset, the number of processes used are the same for all values of $|\mathcal{S}|$, e.g., for the WDC graph we use 128 compute nodes, each node running 16 processes, total 2,048 processes. Runtime is broken down to the key computation steps.

to the key computation steps described in §4.

We make the following observations: *(i)* The computation time of Voronoi cells decreases with increasing $|\mathcal{S}|$. This is due to the convergence rate being accelerated in the presence of a large $\mathcal{S}$ (a data-dependent artifact). With increasing graph size, Voronoi cell computation dominates overall runtime. *(ii)* For all datasets, the time spent in computing the local min-distance cross-cell edges does not vary much with the seed set size. The global min-distance cross-cell edge computation time, however, often increases with seed set size, especially for the smaller graphs. *(iii)* The size of the distance graph $\hat{\mathcal{G}}_{\mathcal{S}}$ increases with the size of the seed sets (Table 5 lists the number of edges in the distance graph of respective graph datasets and corresponding seed sets). For example, in the LVJ graph, when $|\mathcal{S}| = 1\text{M}$, the distance graph has over 22.5M edges. For a large $\hat{\mathcal{G}}_{\mathcal{S}}$, A-MST computation often dominates runtime, particularly in the presence of smaller graphs. The small graphs end up with as many or more edges in their distance graphs that the A-MST is built from than are present in some of the large graphs. The exception

here is FRS, with as many as 689M edges in its distance graph, which is the only large graph where the A-MST time begins to dominate. *(iv)* The last step that constructs the final Steiner tree from the A-MST of the distance graph typically requires the least amount of time out of the five processing stages. Table 6 lists the number of edges in the final tree for each dataset and seed set combination, for both D-1 and D2.

Why do different graphs have varying sizes of distance graph and final Steiner tree for the same number of seeds? A large distance graph suggests abundance of inter-seed vertex paths, i.e., presence of a large number of cross-cell edges. While the FSR graph is significantly smaller than the hundred billion edge WDC graph, the distance graph of FSR is the largest in our experiments. The distance graphs of smaller LVJ and PTN graphs are considerably large given their respective sizes. One plausible explanation of this phenomenon is that this is inherent to the topology of social media graphs like FRS, LVJ and PTN. Topologically speaking, social media graphs are different from web graphs (e.g., WDC, CLW, and UKW). Social media graphs often contain larger, denser clusters compared to web graphs. This supports the observation that the social media graphs have larger distance graphs – even though the same approach was used for seed selection, when $|\mathcal{S}|$=1M, the probability of a shortest path existing between a pair of seed vertices is higher due to the presence of large, tight-knit clusters in social media graphs.

We observe that, with increasing number of seed vertices, the larger graphs tend to produce larger Steiner trees – the average path length between seed vertices are longer (i.e., the trees have more Steiner vertices).

Table 5: Number of edges in the distance graph of different graph datasets and seed sets, when using the solution technique D-2.

$\|\mathcal{S}\|$	10	100	1K	10K	100K	1M
WDC	9	103	1,559	26,721	4,146,477	64,587,719
CLW	9	118	2,801	91,806	1,858,527	36,938,582
UKW	9	464	7,673	112,885	1,177,873	12,011,186
FRS	12	280	19,756	502,503	52,262,567	689,677,348
LVJ	13	379	45,695	1,152,360	7,618,849	22,583,967
PTN	14	237	69,969	1,116,442	4,604,626	9,675,462

Table 6: Total number of edges in the output Steiner trees of different graphs and seed vertex sets. The table lists results for both D-1 and D-2.

$\|\mathcal{S}\|$	WDC		CLW		UKW		FRS		LVJ		PTN	
	D-1	D-2	D-1	D-2	D-1	D-2	D-1	D-2	D-1	D-2	D-1	D-2
10	326	517	152	166	184	184	149	149	105	105	125	144
100	1,953	2,727	1,265	1,439	1,542	1,591	1,485	1,499	1,108	1,127	1,112	1,204
1K	12,488	18,209	6,909	7,917	10,644	10,768	11,639	11,921	7,193	7,485	8,075	8,930
10K	85,586	129,228	36,397	41,971	42,782	43,977	85,211	86,997	50,530	51,807	51,988	56,587

Randomly selected seeds from a large connected component have higher likelihood of being far apart. Also, the path length can be influenced by both edge weights and graph topology. The diameter of a graph influences the computation time of Voronoi cell – web graphs are known to have larger diameters than social media graphs.

5.2. Resources Usage Analysis

Fig. 4 compares runtime and resources requirements of D-1 and D-2 using six different graphs and 10K seed vertices. D-1 uses the minimum number of compute nodes required to run each experiment (listed on the X-axis in the

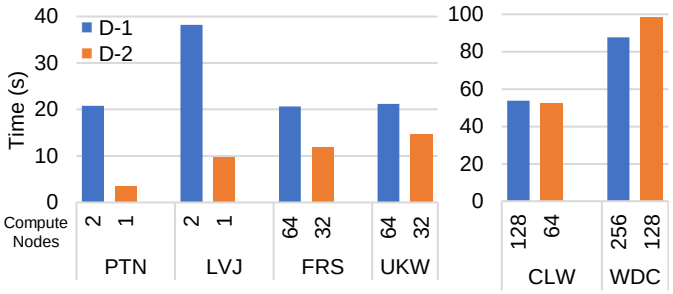


Figure 4: Comparison of D-1 and D-2 with respect to end-to-end runtime and number of compute nodes used (top X-axis labels) in each experiment. D-1 uses the minimum number of compute nodes required to run each experiment. In all experiments, $|\mathcal{S}| = 10$K.

chart). In the same graph, D-2 can successfully identify a Steiner tree using half the number of compute nodes (thus, half the memory) required by D-1. Using twice as many compute nodes D-1 achieves better time-to-solution in some of the experiments.

To further explain resource efficiency enabled by D-2, we measure cluster-wide memory usages of D-1 and D-2 solutions. Fig. 5 shows cluster-wide peak memory usages by the two solutions for three graphs and 10K seed vertices. For each experiment, total memory usage is broken down to memory required for the in-memory binary graph and algorithm states (which includes memory used by communication buffers). For LVJ, CLW and WDC graphs, D-2 uses 1, 64, and 128 compute nodes, respectively. For D-1, we had to double the number of compute nodes to meet its memory demands. For the smaller LVJ, memory usage is dominated by algorithm states; D-1 consumes 22.7× more memory than D-2. For the larger CLW and WDC graphs, D-1 uses 6× and 3.7× more memory, respectively. Note that in D-1, high memory usage is due to large buffers required to perform MPI collective operations on $\mathcal{E}_N$, the edge list of the distance graph. When $|\mathcal{S}| = 10$K, the length of $\mathcal{E}_N$ is

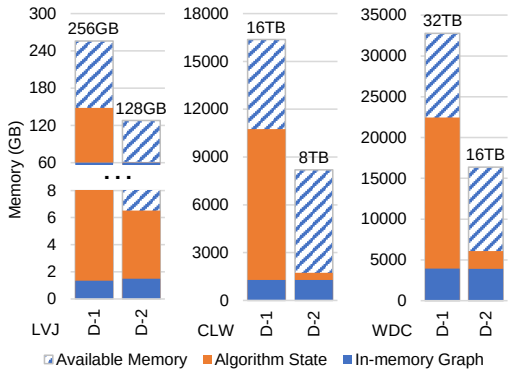


Figure 5: Cluster-wide peak memory usages by D-1 and D-2 for three graphs and 10K seed vertices. The total system memory is listed on top of respective stacked bars. For D-1, the number of compute nodes is doubled to meet its high memory demands.

$\sim$50M. D-2's superior scale out design liberates it from memory limitations of D-1 (§4.2). Fig. 3 shows that, using the same number of nodes, D-2 can identify Steiner trees from $|\mathcal{S}| = 1$K all the way to 1M seeds – a significant improvement in resource efficiency over D-1.

5.3. Comparison with Related Work: Runtime and Result Quality

We compare our distributed solution with two sequential 2-approximate algorithms WWW (15) and Mehlhorn (17), and sequential exact solution SCIP-JACK (21). ParaSCIP (36) enables SCIP-Jack to harness distributed clusters by replicating the data graph on all processes. Unfortunately, ParaSCIP did not yield any performance advantage in our experiments: the authors of SCIP-Jack reproduced our experiments and concluded that branch-and-bound search was lacking parallelism for the datasets we used (explained in (21)); hence, we present results obtained using sequential SCIP-JACK 2.0 (reproduced by original authors as well). Unfortunately, we do not have access to another parallel, distributed implementation that solves the same

Table 7: Runtime comparison between our distributed solutions (D-1 and D-2) – using 16 processes on a single compute node, and three related work: exact solution SCIP-Jack (S) (21), and 2-approximation algorithms WWW (W) (15) and Mehlhorn (M) (17). Time units: hour (h), minute (m), second (s), and millisecond (ms). (Reproduced from (18).)

$\|\mathcal{S}\|$	10					100					1000				
	S	W	M	D-1	D-2	S	W	M	D-1	D-2	S	W	M	D-1	D-2
LVJ	9.4m	27.8s	25.1s	5.5s	5.7s	10.1m	28.4s	40.5s	6.4s	6.5s	45.8m	28.4s	1.9m	9.3s	9.3s
PTN	7.3m	8.4s	14.8s	4.6s	5.3s	11.0m	8.3s	28.7s	4.0s	4.5s	1.0h	8.4s	1.5m	3.9s	4.0s
MCO	8.8s	0.3s	0.2s	0.3s	0.2s	9.8s	0.3s	0.4s	0.3s	0.2s	53.5s	0.3s	1.6s	0.5s	0.2s
CTS	<1s	0.5ms	0.9ms	5ms	14ms	<1s	0.7ms	4.9ms	6ms	14.5ms	<1s	2.2ms	0.1s	0.2s	15ms

problem that we could use for direct comparison. We implement the WWW and Mehlhorn algorithms in C++ based on a cache friendly CSR graph data structure. We use four small graphs in Table 4 and up to 1K seed vertices.

Table 7 presents a comparison of the runtime performance. While for the smaller CTS and MCO graphs, work efficient WWW performances slightly better, the advantage of our parallel solution becomes apparent for the larger PTN and LVJ graphs: when using 16 processes on a single node, our solution is up to 27× faster than Mehlhorn and 5× faster than WWW.

Table 8 shows the quality of approximation of our distributed solutions. We compare the total weight of a Steiner tree identified by our algorithm with that of the exact Steiner minimal tree produced by SCIP-Jack, i.e., we measure $D(\mathcal{T})/D_{min}(\mathcal{G})$. The empirical results complement the 2-approximation bound that our solution guarantees: on average, the total distance $D(\mathcal{T})$ of the Steiner tree identified by our first solution D-1 is 1.0527 times greater than that of the Steiner minimal tree (i.e., $D_{min}(\mathcal{G})$) which is well within the the-

Table 8: The quality of approximation, i.e., $D(\mathcal{T})/D_{min}(\mathcal{G})$, of distributed solutions: on the left, the approximation ratio of a Steiner tree identified by D-1; on the right, the approximation ratio of a Steiner tree identified by D-2. Steiner minimal trees were computed using SCIP-Jack (21).

	D-1			D-2		
$\|S\|$	10	100	1000	10	100	1000
LVJ	1.0112	1.0110	1.0183	1.0112	1.0264	1.0583
PTN	1.1684	1.0859	1.0790	1.3068	1.2630	1.2725
MCO	1.0375	1.0668	1.0435	1.1797	1.1534	1.1191
CTS	1.0526	1.0438	1.0138	1.0526	1.0655	1.0398

oretical bound of $\leq 2(1-1/l)$. The second solution D-2, which computes an approximate minimum spanning tree of the distance graph, identifies Steiner trees with total distance, on average, 1.1290 times greater than that of the Steiner minimal tree; in other words, on average total distance is only $\sim$6.8% greater than the total distance of a Steiner tree produced by D-1.

Table 9 lists ratios of the total distance of Steiner trees identified by D-2 and D-1 solutions, i.e., $D(\mathcal{T}_2)/D(\mathcal{T}_1)$, for different graphs and seed sets. Since D-2 computes an approximate minimum tree, for the same seed set, the total distance of a Steiner tree identified by D-2 can be greater than the distance of a tree found by D-1. The results in table 9 suggest, when using the D-2 solution, smaller graphs and seed sets incurs low approximation error compared to their larger counterparts. In all observed cases, however, the ratio is always ≤ 2 which is within the theoretical bounds we provide in the appendix. Note the ratio could be as bad as 4 for $\beta = 2$, in a case where D-1 happens to provide an exact Steiner minimal tree and D-2 is arbitraily close

Table 9: Ratio of total distance of Steiner trees identified by D-2 and D-1. Each cell in the table lists $D(\mathcal{T}_2)/D(\mathcal{T}_1)$ for the corresponding graph and seed set.

$\lvert\mathcal{S}\rvert$	WDC	CLW	UKW	FRS	LVJ	PTN
10	1.8267	1.2667	1.0000	1.0000	1.0000	1.1185
100	1.7895	1.5416	1.0169	1.0144	1.0153	1.1631
1K	1.7648	1.4081	1.0182	1.0358	1.0393	1.1793
10K	1.7789	1.3387	1.0400	1.0431	1.0303	1.1529

to the derived worst-case 2β approximation bound.

5.4. Strong Scaling

Fig. 6 presents results of strong scaling experiments for D-1 using the four largest graphs in Table 4, 100 and 1K seed vertices, and up to 512 compute nodes. Runtime is broken down to the key computation steps described in §4. For each dataset, the smallest platform scale (e.g., 32 nodes for FRS) is the one that can accommodate the graph and required algorithm states (each compute node has 128GB memory). For all datasets, the majority of the runtime is spent in computing Voronoi cells, followed by local min-distance edge computation, whose scalability improves almost linearly when the platform size is doubled. More time is spent in local min-distance edge computation when the number of seeds is increased by an order of magnitude. Time spent in the remaining distributed computation phases are insignificant compared to the total runtime (hence, scalability). Moving from 100 to 1K seeds, for a dataset, no major deviation is observed in scaling performance. Overall, we observe decent scaling – the larger CLW and WDC graphs demonstrate better scaling (up to 90% efficient scaling) compared to the smaller FRS and UKW datasets. The Voronoi cell computation, whose performance is most

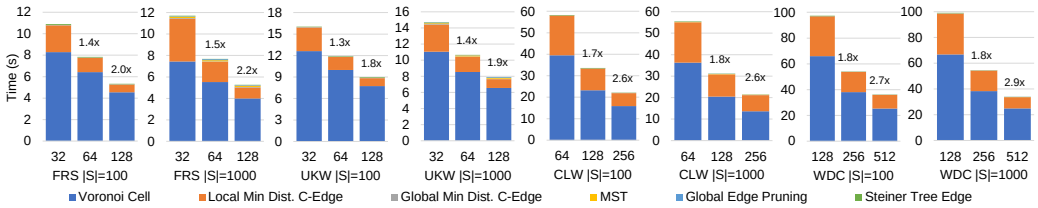


Figure 6: Strong scaling experiment results of D-1 using the FRS, UKW, CLW and WDC datasets. Runtime is broken down to the key computation steps (identified in chart legends). For each experiment, X-axis labels in the first row indicates the platform size (number of compute nodes). Speedup achieved over the smallest scale is shown on top of respective stacked bar plots. (Reproduced from (18).)

affected by the irregular graph topology, is the main scalabaility bottleneck.

Fig. 7 presents results of strong scaling experiments for D-2. Here, the experiment setup is similar to the one used for D-1 earlier; here, we use 100, 1K, 100K and 1M seed vertices. Performance of D-2 is comparable to the scaling efficiency of D-1 (for the most parts). Unfortunately, in D-2, YGM-based A-MST is the bottleneck in strong scaling. While it improves resource efficiency, the A-MST routine often shows subpar strong scaling performance. Our investigation suggests, the A-MST implementation suffers from significant load imbalance; scaling efficiency degrades with increasing node count (e.g., Fig. 7, WDC $|\mathcal{S}| = 1\text{M}$). We conclude that, to improve scaling efficiency of distributed A-MST, we would require a more balanced partitioning scheme for the distance graph beyond what YGM currently offers (A-MST relies on YGM's distributed disjoint set container).

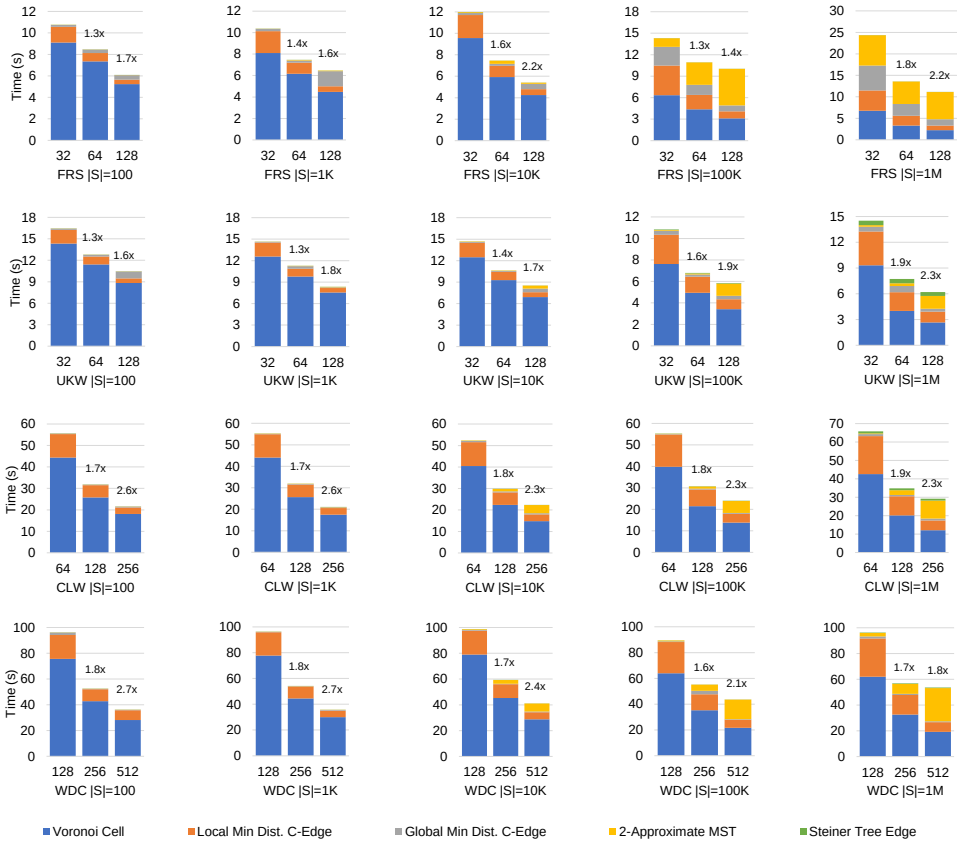


Figure 7: Strong scaling experiment results of D-2 using the FRS, UKW, CLW and WDC datasets. Runtime is broken down to the key computation steps (identified in chart legends). For each experiment, X-axis labels in the first row indicates the platform size (number of compute nodes).

5.5. FIFO vs Priority Queue for Message Propagation

In §4, we discussed employing a priority queue for message propagation within HavoqGT, specifically to accelerate the convergence of the Voronoi cell computation. Here, we demonstrate the advantage of this design optimization. Fig. 8 compares runtime performance of using a priority queue

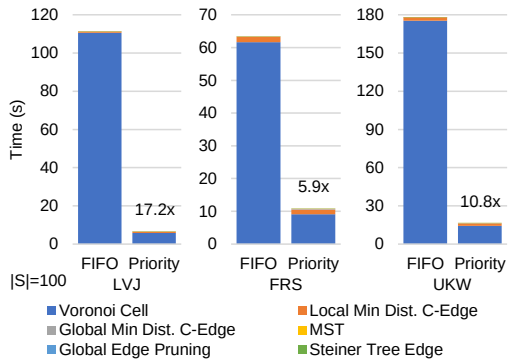


Figure 8: Runtime comparison of using FIFO and priority queues using a fixed number of seed vertices and cluster sizes (one node for LVJ, and 32 nodes for FRS and UKW). These results are for solution D-2. Runtime is broken down to the key computation steps (identified in chart legends). Speedup achieved as a result of using the priority queue is shown on top of respective stacked bar plots.

with that of a FIFO queue (default in HavoqGT). The advantage of the priority queue is significant: for FRS, the priority queue offers 5.9× speedup, while for LVJ, the improvement is over 17×. In (18), we also demonstrated the role of the priority queue in improving generated message volume.

5.6. Studying Seed Selection Alternatives

Earlier in the section, we discussed the seed selection approach used for evaluating this work (we call it BFS-level). In this section, we explore three additional seed selection alternatives and study their influence on performance and characteristics of the output tree. *(i)* Uniform random – we randomly select $|\mathcal{S}|$ vertices from the largest connected component. *(i)* Eccentric – we ensure seed vertices are far away from each other. We use a technique inspired by the k-BFS heuristic (37) – the algorithm iteratively identifies ($k = |\mathcal{S}|$) BFS sources which are then used as seeds for Steiner tree computation. Starting with a random vertex in the largest connected compo-

Table 10: Comparison of seed selection strategies using the LVJ graph. For each technique, the table lists runtime, distance $D(\mathcal{T})$ of the output Steiner tree $\mathcal{T}$ and number of the edges $|\mathcal{E}_\mathcal{T}|$ in $\mathcal{T}$. (Reproduced from (18).)

	BFS-level			Uniform Random		
$\|\mathcal{S}\|$	Time	$D(\mathcal{T})$	$\|\mathcal{E}_\mathcal{T}\|$	Time	$D(\mathcal{T})$	$\|\mathcal{E}_\mathcal{T}\|$
100	6.4s	426.9K	1,108	5.5s	207.5K	889
1K	9.3s	2,840.9K	7,193	9.0s	1,845.5K	7,202
10K	38.2s	28,903.3K	50,530	37.5s	18,123.5K	49,755
	Eccentric			Proximate		
$\|\mathcal{S}\|$	Time	$D(\mathcal{G}_\mathcal{T})$	$\|\mathcal{E}_\mathcal{T}\|$	Time	$D(\mathcal{G}_\mathcal{T})$	$\|\mathcal{E}_\mathcal{T}\|$
100	6.1s	412.3K	1,115	7.6s	16.0K	272
1K	6.3s	6,091.5K	6,548	8.1s	101.0K	1,699
10K	31.4s	49,644.5K	49,691	39.1s	1,105.9K	16,624

nent as the BFS source, each of the subsequent $k-1$ BFS sources is identified based on BFS-levels computed in the previous $k-n$ rounds ($1 \leq n \leq k-1$). The following equation computes the BFS source for the $k-n+1$'th round:

$$u_{k-n+1} = max(\{\textstyle\sum_{j=0}^{k-n} l_j(v_i)\})$$

Here, $u_{k-n+1}, v_i \in \mathcal{V}$, $u_{k-n+1} \neq v_i$, $l_j(v_i)$ is the BFS-level of v_i in the j'th round. *(iii)* Proximate – seed vertices are selected such that they are close to each other, following the same approach in the eccentric case. The following equation computes the BFS source for the $k-n+1$'th round:

$$u_{k-n+1} = min(\{\textstyle\sum_{j=0}^{k-n} l_j(v_i)\})$$

Table 10 presents the results. In summary, we do not observe notable differences in performance between techniques. Compared to other techniques,

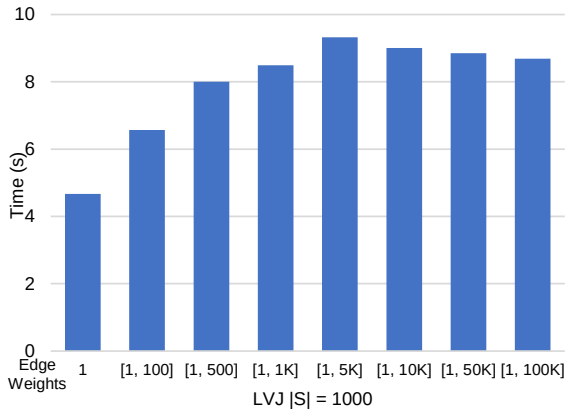


Figure 9: Influence of edge weight distribution on end-to-end runtime performance: X-axis labels are the edge weight range (inclusive). We compare two cases, using FIFO and priority queues for message propagation. Here, $\mathcal{S}$ is fixed, 1K seeds, and the LVJ graph uses a single node for all experiments.

proximate produces significantly smaller trees (which we tried to avoid in the evaluation of our work).

5.7. Edge Weight Distribution vs Runtime Performance

Edge weight distribution is known to influence the convergence time of shortest path computation (28; 25). Fig. 9 studies this phenomena using the LVJ graph for a fixed $\mathcal{S}$ but different edge weight distributions, from [1, 100] up to [1, 100K]. Also, included in the chart, when all edges have the same weight, 1. Edge distribution does impact runtime, especially Voronoi cell computation. The scenario where all edges have the same weight, achieves the fastest time, followed by the weight distribution [1, 100]. The weight range [1, 5K], which has been used in all experiments in this work that use the LVJ graph, is the slowest. The Steiner tree computation time is directly influenced by the volume of edge relaxation operations, i.e., the number of times a vertex's distance is updated, and the vertex visits its immediate

neighbors during Voronoi cell computation.

6. Related Work

The Steiner minimal tree problem is one of Karp's 21 NP-complete problems (1). Hakimi (3) is often credited for proposing the first exact solution, followed by a significant number of contributions over the decades – a comprehensive survey is available in (38). Often practical applications solve a particular variant of the problem, such as the Steiner arborescence, Euclidean and rectilinear minimum tree, group, prize-collecting, and node-weighted Steiner tree problem (38; 39). GeoSteiner (40) is a well known solver for exact Euclidean and rectilinear Steiner minimal tree problems. SCIP-Jack (21; 41) is the state-of-the-art general purpose exact solver, winner of the 11th DIMACS challenge in the rooted prize-collecting problem category (34). SCIP-Jack follows a branch-and-cut approach which enables tight linear programming (LP) relaxation.

For many years, there has been continuing interest in polynomial-time algorithms with tight approximation bounds. In §1 – 3, we discussed key 2-approximation algorithms. Winter and Smith (42) proposed heuristics with an approximation bound less than 2. They do not, however, improve the runtime over (14), and their algorithm is not a good candidate for parallelization. More recently, Hougardy and Prömel improved the approximation ratio to 1.598 (43), followed by Robins and Zelikovsky further improving the ratio to 1.55 (44). To the best of our knowledge, the current best bound is given by an LP-based approximation algorithm using iterative randomized rounding by Byrka et al. (45), which offers an approximation bound of $ln(4)+\varepsilon \leq 1.39$.

Often algorithms with approximation ratios less than 2 iteratively refine a base-solution which is commonly computed using a 2-approximation algorithm (46; 35). STAR (47) and SketchLS (48), although lacking guarantees of polynomial runtime and a tight approximation bound, were evaluated using real-world, scale-free graphs. Genetic algorithm (49), Swarm optimization (50), Physarum optimization (51) heuristics have been used for approximating Steiner minimal trees – they only offer probabilistic solutions and their result quality suffers with increasing graph size.

We are not aware of any parallel, distributed solution that can compete with the scalability demonstrated in this paper. Several papers discuss parallel, distributed Steiner tree computation from the theoretical perspective only (46; 52; 53; 54). PARSTEINER94 (55), algorithmically similar to GeoSteiner, is an early parallel exact solver. ParaSCIP (36) is an MPI library enabling SCIP-Jack to harness distributed clusters. It replicates the data graph on all processes and performs parallel branch-and-bound search. A distributed dual ascent (42) heuristic is presented in (56) whose performance was demonstrated using graphs with less than 10K edges. A GPU implementation of the STAR heuristic (47) is presented in (57). Group Steiner tree computation on the GPU with application to VLSI routing is presented in (58). Curious readers are encouraged to check out the survey paper by Bezensek and Robic (59) – it presents an extensive survey of distributed Steiner tree computation efforts.

A literature review of Steiner tree solvers would be incomplete without discussing the 11th DIMACS Implementation Challenge 2014 (34) and 3rd PACE 2018 Steiner tree computation challenge (35). The 11th DI-

MACS challenge considers the classical exact Steiner tree problem and its variants, e.g., Euclidean, rectilinear, prize-collecting, node-weighted, group, height- or hop- constrained rooted Steiner trees, as well as low-cost heuristics and stochastic alternatives. The 3rd PACE challenge is divided in to three categories: Tracks 1 and 2 target exact solutions, more specifically, fixed-parameter tractable (FPT) algorithms. Track 1 instances have a small number of seed vertices (10^2), while Track 2 instances have a low treewidth. Track 3 is dedicated to heuristics, hence closely related to our work. Also, this track considers instances that are large, have many seed vertices and high treewidth: the average number of graph edges is 48K, and the average number of seeds is 1,114; and the optimum tree distance of several instances is unknown. Both DIMACS and PACE challenges consider popular datasets used for evaluation in the past, SteinLib and Vienna being among the largest. The datasets used in the PACE challenge have at most 10^5 edges and 10^4 seeds. The DIMACS challenge also considers synthetically generated EFST datasets, to the best of our knowledge, the largest instance used contains 100,000 seeds. The Stockholm dataset, used in the hop-constrained directed Steiner tree problem category, has about 20M edges but few seed vertices.

We took two of the 3rd PACE Track 3 instances, instance#197 with the largest number of edges, 366,093, and instance#200 which has the most seeds, 17,832, and run our D-2 solution on them. On a single compute node and using 16 MPI processes, D-2 was able to identify Steiner trees in both instances in under two seconds. The $\frac{optimum}{solution}$ score (35) of instance#197 is 0.820 and instance#200 is 0.998. In Track 3, the average score of top four teams was 0.997. CIMAT, the winning team in the heuristic track developed

an evolutionary algorithm. The authors of SCIP-Jack took the runner-up spot in the same track (35). Earlier we discussed SCIP-Jack, winner of the 11th DIMACS challenge in the rooted prize-collecting exact problem category. Other key winners of the 11th DIMACS includes PUW (60) – first place in the heuristics for the classical problem category; Staynerd (61) – winner of the prize-collecting heuristics category. mozartballs, by the authors of Staynerd, won in multiple exact problem categories. Since 11th DIMACS, SCIP-Jack has been further improved and can support much larger instances (see §5.3).

7. Conclusion

We have demonstrated the effectiveness of the design choices aimed at addressing the resource inefficiencies and scalability limitations of our prior distributed Steiner tree solution in (18). However, there is room for further improving this work. While they improve resource efficiency, the YGM-based distributed A-MST suffers from load imbalance and often show subpar strong scaling performance – an area where we are investing research efforts. We are interested in investigating potential benefits of GPU acceleration to solve the 2-approximate Steiner minimal tree problem. In the future, we would like to extend our distributed design to support other popular variants of the Steiner tree problem, such as the Steiner arborescence, group, prize-collecting, and node-weighted Steiner tree problem (38; 39). We want to explore avenues for distributed solutions that improve the effective approximation quality: there are existing techniques that starts from the 2-approximate solution then improve it with local search (46; 35); an attractive future direction for

our research.

Appendix A. Approximation Quality

We present a relatively simple approximation bound that shows our distributed approach is guranteed to produce Steiner trees whose total distance is within a factor 2β of an actual minimal Steiner tree, where β is a configurable parameter determining the approximation bound on the approximate minimum spanning tree procedure. The ratio is often much lower in practice, even for massive graphs, very large seed sets, and a very large number of compute nodes/processors (see Table 8)

We utilize several fundamental definitions from graph theory, including walks, paths, and trees, and follow several previous works on Steiner trees (2; 3; 13; 14; 15; 16; 17; 42) to define various maps between these objects living on $\mathcal{G}$ and $\mathcal{G}_{\mathcal{S}}$ to obtain the bound.

Definition 1. (Walks and Properties)

(i) **Walk.** *A walk $\mathcal{W}$ in $\mathcal{G}(\mathcal{V}, \mathcal{E}, d)$ is sequence of connected edges within $\mathcal{G}$. Implicitly, the vertices that are incident to each edge and connect each sequential pair of edges are said to be in the sequence, in order. The edges in $\mathcal{W}$ can repeat. The total weight of $\mathcal{W}$ is the sum of the edge weight $D(\mathcal{W})$.*

(ii) **Walk Visitation.** *Given vertex set $\mathcal{S}$, a walk $\mathcal{W}$ visits all vertices in $\mathcal{S}$, if for each $i \in \mathcal{S}$ there is one or more edges in $\mathcal{W}$ that are incident to i.*

(iii) **Complete Walk.** *If a walk $\mathcal{W}$ visits all of $\mathcal{S}$, we say it is a complete walk for $\mathcal{S}$. The set of all complete walks is $\mathbf{W}_{\mathcal{G}}(\mathcal{S})$.*

(iv) **Optimal Complete Walk (OCW).** *An OCW $\mathcal{W}^\star$ satisfies*

$$D(\mathcal{W}^\star) = \min_{\mathcal{W} \in \mathbf{W}_{\mathcal{G}}(\mathcal{S})} D(\mathcal{W}^\star).$$

This is related to the traveling salesperson problem.

Definition 2. (PATHS)

(i) **Path.** *A path $\mathcal{P}$ is a walk with no repeating vertices. For vertices $i, j \in \mathcal{V}$, path $\mathcal{P}(i,j)$ starts with an edge incident to i and ends with an edge incident to j. The set of all paths between i and j is $\mathbf{P}_{(i,j)}$.*

(ii) **Shortest Path.** *For vertices $i, j \in \mathcal{V}$, a shortest path $\mathcal{P}^\star(i,j)$ satisfies*

$$D(\mathcal{P}^\star(i,j)) = \min_{\mathcal{P} \in \mathbf{P}_{(i,j)}} D(\mathcal{P}).$$

(iii) **Subset Closure Graph.** *Given $\mathcal{S} \subset \mathcal{V}$, define the subgraph $\mathcal{G}_{\mathcal{S}}(\mathcal{S}, \mathcal{E}_{\mathcal{S}}, d_{\mathcal{S}})$, where $\mathcal{E}_{\mathcal{S}} = \mathcal{S} \times \mathcal{S}$ and the closure distance is the length of shortest path between all vertex pairs,*

$$d_{\mathcal{S}}(i,j) := D\left(\mathcal{P}^\star(i,j)\right).$$

(iii) **Subset Voronoi Graph (17).** *For each $i \in \mathcal{S}$, let $\mathcal{R}_i$ be the Voronoi set of all vertices closest to i,*

$$\mathcal{R}_i := \left\{ u \in \mathcal{V} \,:\, d(u,i) = \min_{j \in \mathcal{S}} d(u,j) \right\}.$$

Note that ties in distance are broken in a consistent way. Define the subgraph $\mathcal{G}_{\mathcal{S}}(\mathcal{S}, \hat{\mathcal{E}}_{\mathcal{S}}, \hat{d}_{\mathcal{S}})$, where $\hat{\mathcal{E}}_{\mathcal{S}} \subset \mathcal{E}_{\mathcal{S}}$ is defined as containing (i,j)

if and only if there exists $u \in \mathcal{R}_i$ *and* $v \in \mathcal{R}_j$ *such that* $(u, v) \in \mathcal{E}$. *Distance is defined as*

$$\hat{d}_{\mathcal{S}}(i,j) := \min_{u \in \mathcal{R}_i, v \in \mathcal{R}_j} \{d(i,u) + d(u,v) + d(v,j)\}$$

In general, $\hat{d}_{\mathcal{S}}(i,j)$ *does not always equal* $d(i,j)$. *Consider* (i,j) *for which all shortest paths go through a third Voronoi set(17).*

Definition 3. (Trees)

(i) **Tree.** *A tree* $\mathcal{T}(\mathcal{V}_{\mathcal{T}}, \mathcal{E}_{\mathcal{T}})$ *in* $\mathcal{G}$ *is a connected acyclic subgraph of* $\mathcal{G}$.

(ii) **Rooted Tree.** *A tree* $\mathcal{T}(\mathcal{V}_{\mathcal{T}}, \mathcal{E}_{\mathcal{T}})$ *that is rooted has a single vertex* $i \in \mathcal{V}_{\mathcal{T}}$ *designated as the root.*

(iii) **Spanning Tree.** *A* spanning tree $\mathcal{T}$ *in a connected graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ *has at least one edge incident to every vertex in* $\mathcal{V}$. *Therefore,* $\mathcal{V}_{\mathcal{T}} \equiv \mathcal{V}$.

Definition 4. (Walk of a Tree) *Let* $\omega(\cdot)$ *be a map that takes any tree* $\mathcal{T}$ *within* $\mathcal{G}$ *and maps it to* $\omega(\mathcal{T})$, *which is a complete walk of* $\mathcal{S}$ *that visits all edges with* $\mathcal{T}$ *twice, yielding*

$$d\left(\omega\left(\mathcal{W}\right)\right) \leq 2D(\mathcal{T}).$$

Map $\omega(\cdot)$ *is defined constructively by choosing a* root *vertex and performing depth-first search to enumerate a sequence of edges such that returning edges from children to parents are also stored. This is easily made deterministic with vertex and edge orderings.*

Definition 5. (Walk Restriction) *Let* $\theta(\cdot)$ *be a map that takes any walk* $\mathcal{W}$ *visiting all of* $\mathcal{S}$ *within* $\mathcal{G}$ *and maps it to* $\theta(\mathcal{W})$, *which is a complete walk*

on all of $\mathcal{G}_{\mathcal{S}}$*, such that*

$$D_{\mathcal{S}}\left(\theta\left(\mathcal{W}\right)\right) \leq D(\mathcal{W}).$$

Map $\theta(\cdot)$ *is defined constructively and deterministically by pruning all edges before and after all vertices from* $\mathcal{S}$*, followed by replacing each sub-sequence of edges between any two vertices* $i, j \in \mathcal{S}$ *with the associated edge* $(i, j) \in \mathcal{E}_{\mathcal{S}}$*. If two sequential vertices in* $\mathcal{S}$ *repeat, no edge from* $\mathcal{E}_{\mathcal{S}}$ *is added. The inequality above is straightforward, as* $d_{\mathcal{S}}(i, j) = D(\mathcal{P}^{\star}(i, j))$*.*

Definition 6. (Tree Lifting) *Let* $\phi(\cdot)$ *be a map that takes any spanning tree* $\mathcal{T}_{\mathcal{S}}$ *within* $\mathcal{G}_{\mathcal{S}}$ *and maps it to* $\phi(\mathcal{T}_{\mathcal{S}})$*, which is a Steiner tree with respect to* $\mathcal{S}$ *satisfying*

$$D\left(\phi\left(\mathcal{T}_{\mathcal{S}}\right)\right) \leq D_{\mathcal{S}}\left(\mathcal{T}_{\mathcal{S}}\right).$$

Map $\phi(\cdot)$ *is defined constructively by sequentially adding a shortest path* $\mathcal{P}(i, j)$ *for each* $(i, j) \in \mathcal{E}_{\mathcal{S}}$*, pruning for paths that have two or more vertices already included in the tree being built up by removing portions of* $\mathcal{P}(i, j)$ *in between the first and last redundant vertices* $u, v \in \mathcal{V}$*. This can be made deterministic simply by using a vertex and an edge ordering to choose specific shortest paths and sort them in a specific order. The inequality above is straightforward, as* $d_{\mathcal{S}}(i, j) = D(\mathcal{P}(i, j))$ *and the edges added to the tree being built up are a subset of* $\mathcal{P}(i, j)$*.*

For $\beta > 1$, let $\mathcal{T}^{(\beta)}$ be an approximate minimum spanning tree of $\mathcal{G}$, given by binning edge distances multiplicatively by factor β and computing connected components via distributed union-find iteratively, starting with the bins in the smallest edges and adding larger bins, one at a time. For $\beta = 1$, let $\mathcal{T}^{(\beta)}$ be a true minimum spanning tree of $\mathcal{G}$.

Theorem 1. (Distributed MST Approximation (20)) *Let $\mathcal{T}^\star$ be a minimal spanning tree. Given the approach outlined above to find $\mathcal{T}^{(\beta)}$, we have*

$$D(\mathcal{T}^{(\beta)}) \leq \beta * D(\mathcal{T}^\star).$$

Theorem 1 shows existence of a parallelizable approach to compute a near-minimal-weight spanning tree $\mathcal{T}_{\mathcal{S}}^{(\beta)}$ of $\hat{\mathcal{G}}_{\mathcal{S}}$ with

$$d\left(\mathcal{T}_{\mathcal{S}}^{(\beta)}\right) \leq \beta * D(\mathcal{T}_{\mathcal{S}}^\star).$$

Theorem 2. (Approximation Quality of Voronoi-β-MST Steiner Trees) *Let $\mathcal{T}^\star$ be a minimal Steiner in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with respect to $\mathcal{S} \subset \mathcal{V}$. Let $\mathcal{T} = \phi(\mathcal{T}_{\mathcal{S}}^{(\beta)})$ be the approximate minimal Steiner treee computed via approach D-2. We have*

$$D(\mathcal{T}) \leq 2\beta * D(\mathcal{T}^\star)$$

Proof.

$$\begin{aligned}
2\beta * D(\mathcal{T}^\star) &\geq \beta * D(\omega(\mathcal{T}^\star)) && \text{(Def. 4)} && \text{(A.1)}\\
&\geq \beta * D(\mathcal{W}^\star) && \text{(OCW)} && \text{(A.2)}\\
&\geq \beta * D_{\mathcal{S}}(\theta(\mathcal{W}^\star)) && \text{(Def. 5)} && \text{(A.3)}\\
&\geq \beta * D_{\mathcal{S}}(\mathcal{T}_{\mathcal{S}}^\star) && \text{(MST)} && \text{(A.4)}\\
&= \beta * D_{\mathcal{S}}(\hat{\mathcal{T}}_{\mathcal{S}}^\star) && \text{((17))} && \text{(A.5)}\\
&\geq D_{\mathcal{S}}\left(\mathcal{T}_{\mathcal{S}}^{(\beta)}\right) && \text{(Thm. 1)} && \text{(A.6)}\\
&\geq D\left(\phi\left(\mathcal{T}_{\mathcal{S}}^{(\beta)}\right)\right) && \text{(Def. 6)} && \text{(A.7)}
\end{aligned}$$

□

References

[1] R. M. Karp, Reducibility among Combinatorial Problems, Springer US, Boston, MA, 1972, pp. 85–103.

[2] E. N. Gilbert, H. O. Pollak, Steiner minimal trees, SIAM Journal on Applied Mathematics 16 (1) (1968) 1–29.

[3] S. L. Hakimi, Steiner's problem in graphs and its implications, Networks 1 (1971) 113–133.

[4] E. Ihler, G. Reich, P. Widmayer, Class steiner trees and vlsi-design, Discrete Appl. Math. 90 (1–3) (1999) 173–194.

[5] A. Caldwell, A. Kahng, S. Mantik, I. Markov, A. Zelikovsky, On wirelength estimations for row-based placement, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (9) (1999) 1265–1278.

[6] Y. Sun, M. Brazil, D. Thomas, S. Halgamuge, The fast heuristic algorithms and post-processing techniques to design large and low-cost communication networks, IEEE/ACM Trans. Netw. 27 (1) (2019) 375–388.

[7] H. Gong, L. Zhao, K. Wang, W. Wu, X. Wang, A distributed algorithm to construct multicast trees in wsns: An approximate steiner tree approach, in: Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '15, ACM, New York, NY, USA, 2015, p. 347–356.

[8] Y. Sun, C. Ma, S. Halgamuge, The node-weighted steiner tree approach to identify elements of cancer-related signaling pathways, BMC Bioinformatics 18 (12 2017).

[9] C. L. Lu, C. Y. Tang, R. C.-T. Lee, The full steiner tree problem in phylogeny, in: O. H. Ibarra, L. Zhang (Eds.), Computing and Combinatorics, Springer Berlin Heidelberg, 2002, pp. 107–116.

[10] W. Lee, W.-K. Loh, M. M. Sohn, Searching steiner trees for web graph query, Comput. Ind. Eng. 62 (3) (2012) 732–739.

[11] R.-H. Li, L. Qin, J. X. Yu, R. Mao, Efficient and progressive group steiner tree search, in: Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 91–106.

[12] M. R. Garey, R. L. Graham, D. S. Johnson, Some np-complete geometric problems, in: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, STOC '76, Association for Computing Machinery, New York, NY, USA, 1976, p. 10–22.

[13] H. Takahashi, et al., An approximate solution for the steiner problem in graphs, Math. Jap. 24 (1980) 573–577.

[14] L. Kou, G. Markowsky, L. Berman, A fast algorithm for steiner trees, Acta Inf. 15 (2) (1981) 141–145.

[15] Y.-F. Wu, P. Widmayer, C.-K. Wong, A faster approximation algorithm for the steiner problem in graphs, Acta Informatica 23 (1986) 223–229.

[16] P. Widmayer, On approximation algorithms for steiner's problem in graphs, in: G. Tinhofer, G. Schmidt (Eds.), Graph-Theoretic Concepts in Computer Science, Springer Berlin Heidelberg, 1987, pp. 17–28.

[17] K. Mehlhorn, A faster approximation algorithm for the steiner problem in graphs, Info. Processing Letters 27 (3) (1988) 125–128.

[18] T. Reza, G. Sanders, R. Pearce, Towards distributed 2-approximation steiner minimal trees in billion-edge graphs, in: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022, pp. 549–559. doi:10.1109/IPDPS53621.2022.00060.

[19] D. Bader, G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs, in: 18th IEEE International Parallel and Distributed Processing Symposium, 2004, pp. 39–.

[20] T. Steil, R. Pearce, Leveraging disjoint sets in distributed memory to approximate minimum spanning trees, Tech. Rep. 1-6, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory (2022).
URL https://computing.llnl.gov/casc

[21] D. Rehfeldt, Y. Shinano, T. Koch, Scip-jack: An exact high performance solver for steiner tree problems in graphs and related problems, in: H. G. Bock, W. Jäger, E. Kostina, H. X. Phu (Eds.), Modeling, Simulation and Optimization of Complex Processes HPSC 2018, Springer International Publishing, Cham, 2021, pp. 201–223.

[22] R. Dathathri, G. Gill, L. Hoang, V. Jatala, K. Pingali, V. K. Nandivada, H.-V. Dang, M. Snir, Gluon-async: A bulk-asynchronous system for distributed and heterogeneous graph analytics, in: 28th International Conf. on Parallel Architectures and Compilation Tech. (PACT), 2019.

[23] K. Pingali, Lonestar minimum spanning tree (2018).
URL `https://iss.oden.utexas.edu/?p=projects/galois/analytics/mst`

[24] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, I. Stoica, Graphx: Graph processing in a distributed dataflow framework, in: Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 599–613.

[25] T. Aasawat, T. Reza, K. Yoshizoe, M. Ripeanu, Hygn: Hybrid graph engine for numa, in: 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 383–390.

[26] M. Ceccarello, A. Pietracaprina, G. Pucci, E. Upfal, A practical parallel algorithm for diameter approximation of massive weighted graphs, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016, pp. 12–21. `doi:10.1109/IPDPS.2016.61`.

[27] K. Wang, D. Fussell, C. Lin, A fast work-efficient sssp algorithm for gpus, in: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 133–146.

doi:10.1145/3437801.3441605.
URL https://doi.org/10.1145/3437801.3441605

[28] C. Xie, R. Chen, H. Guan, B. Zang, H. Chen, Sync or async: Time to fuse for distributed graph-parallel computation, SIGPLAN Not. 50 (8) (2015) 194–204.

[29] R. Pearce, M. Gokhale, N. M. Amato, Faster parallel traversal of scale free graphs at extreme scale with vertex delegates, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 549–559.

[30] LLNL, Ygm (2022).
URL https://github.com/LLNL/ygm

[31] T. Steil, T. Reza, K. Iwabuchi, B. W. Priest, G. Sanders, R. Pearce, Tripoll: Computing surveys of triangles in massive-scale temporal graphs with metadata, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3458817.3476200.
URL https://doi.org/10.1145/3458817.3476200

[32] LLNL, Quartz (2017).
URL https://hpc.llnl.gov/hardware/platforms/Quartz

[33] T. Reza, M. Ripeanu, G. Sanders, R. Pearce, Approximate pattern

matching in massive graphs with precision and recall guarantees, in: ACM Inter. Conf. on Management of Data, SIGMOD '20, 2020.

[34] D. S. Johnson, T. Koch, R. F. Werneck, M. Zachariasen, 11th dimacs implementation challenge in collaboration with icerm: Steiner tree problems (2014).
URL https://dimacs11.zib.de/

[35] É. Bonnet, F. Sikora, The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration, in: C. Paul, M. Pilipczuk (Eds.), 13th International Symposium on Parameterized and Exact Computation (IPEC 2018), Vol. 115 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019, pp. 26:1–26:15. doi:10.4230/LIPIcs.IPEC.2018.26.
URL http://drops.dagstuhl.de/opus/volltexte/2019/10227

[36] G. Gamrath, T. Koch, D. Rehfeldt, Y. Shinano, Scip-jack - a massively parallel stp solver, Tech. Rep. 14-35, ZIB, Berlin (2014).

[37] K. Iwabuchi, G. Sanders, K. Henderson, R. Pearce, Computing exact vertex eccentricity on massive-scale distributed graphs, in: 2018 IEEE Inter. Conf. on Cluster Computing (CLUSTER), 2018. doi:10.1109/CLUSTER.2018.00043.

[38] M. Hauptmann, M. K. (eds.), A Compendium on Steiner Tree Problems, Hausdorff Center for Mathematics, Univ. of Bonn, 2013.

[39] D. Rehfeldt, A generic approach to solving the steiner tree problem and variants, Tech. rep., The Technical University of Berlin (2015).

[40] D. M. Warme, P. Winter, M. Zachariasen, Exact Algorithms for Plane Steiner Tree Problems: A Computational Study, Springer US, Boston, MA, 2000, pp. 81–116.

[41] D. Rehfeldt, T. Koch, Implications, conflicts, and reductions for steiner trees, in: M. Singh, D. P. Williamson (Eds.), Integer Programming and Combinatorial Optimization, Springer, 2021, pp. 473–487.

[42] P. Winter, J. Smith, Path-distance heuristics for the steiner problem in undirected networks, Algorithmica (7) (1992) 309–327.

[43] S. Hougardy, H. J. Prömel, A 1.598 approximation algorithm for the steiner problem in graphs, in: Proceedings of the 10th Annual Acm-Siam Symposium on Discrete Algorithms, Soda, 1999, pp. 448–453.

[44] G. Robins, A. Zelikovsky, Improved steiner tree approximation in graphs, in: In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.

[45] J. Byrka, F. Grandoni, T. Rothvoß, L. Sanità, An improved lp-based approximation for steiner tree, in: Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 583–592.

[46] J.-S. Park, W. W. Ro, H. Lee, N. Park, Parallel algorithms for steiner tree problem, in: 3rd International Conference on Convergence and Hybrid Information Technology, Vol. 1, 2008, pp. 453–455.

[47] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, G. Weikum, Star: Steiner-tree approximation in relationship graphs, in: Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09, IEEE Computer Society, USA, 2009, p. 868–879.

[48] A. Gubichev, T. Neumann, Fast approximation of steiner trees in large graphs, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM '12, ACM, New York, NY, USA, 2012, p. 1497–1501.

[49] A. Kapsalis, V. Rayward-Smith, G. Smith, Solving the graphical steiner tree problem using genetic algorithms, Journal of the Operational Research Society 44 (04 1993).

[50] X. Ma, Q. Liu, A particle swarm optimization for steiner tree problem, in: 6th Inter. Conf. on Natural Comp., Vol. 5, 2010.

[51] Y. Sun, S. Halgamuge, Fast algorithms inspired by physarum polycephalum for node weighted steiner tree problem with multiple terminals, in: IEEE Congress on Evol. Comp. (CEC), 2016, p. 3254–3260.

[52] K. Makki, K. Been, N. Pissinou, A parallel algorithm for the steiner tree problem, in: Proceedings of ICCI'93: 5th International Conference on Computing and Information, 1993, pp. 380–384.

[53] P. Saikia, S. Karmakar, Distributed approximation algorithms for steiner tree in the congested clique, International Journal of Foundations of Computer Science 31 (07) (2020) 941–968.

[54] H. Akbari, Z. Iranmanesh, M. Ghodsi, Parallel minimum spanning tree heuristic for the steiner problem in graphs, in: 2007 International Conference on Parallel and Distributed Systems, 2007, pp. 1–8.

[55] F. C. Harris, Steiner Minimal Trees: An Introduction, Parallel Computation, and Future Work, Springer US, Boston, MA, 1998, pp. 851–903.

[56] L. M. A. Drummond, M. Santos, E. Uchoa, A distributed dual ascent algorithm for steiner problems in multicast routing, Netw. 53 (2) (2009) 170–183.

[57] C. Mathieu, M. Klusch, Accelerated steiner tree problem solving on gpu with cuda, in: G. Wang, A. Zomaya, G. Martinez, K. Li (Eds.), Algorithms and Architectures for Parallel Processing, Springer International Publishing, Cham, 2015, pp. 444–457.

[58] V. Maringanti, B. Imana, P. Yoon, Gpu-accelerated vlsi routing using group steiner trees, The Journal of Computational Science Education 8 (2017) 16–19.

[59] M. Bezensek, B. Robic, A survey of parallel and distributed algorithms for the steiner tree problem, International Journal of Parallel Programming (04 2014).

[60] T. Pajor, E. Uchoa, R. F. Werneck, A robust and scalable algorithm for the steiner problem in graphs, Mathematical Programming Computation 10 (1) (2018) 69–118. doi:10.1007/s12532-017-0123-4.
URL https://doi.org/10.1007/s12532-017-0123-4

[61] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, M. Sinnl, Thinning out steiner trees: a node-based model for uniform edge costs, Mathematical Programming Computation 9 (2) (2017) 203–229. doi:10.1007/s12532-016-0111-0.
URL https://doi.org/10.1007/s12532-016-0111-0