Massively Parallel Approximate Steiner Tree Algorithms

Chilei Wang, Qiang-Sheng Hua*, and Hai Jin

National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, People's Republic of China.
qshua@hust.edu.cn

Abstract. This work studies the approximate Steiner tree problem in the *Massively Parallel Computation* (MPC) model where each machine has $O(n^\sigma)$ memory and $\sigma \in (0, 1)$. n is the number of nodes in a graph. We focus on the undirected connected weighted graphs with shortest path diameter D and a terminal set S. The shortest path diameter is the minimum number of edges required for the shortest path constituting a weighted graph's diameter. The straightforward approach takes $O(n)$ rounds and $O(n^{3-\sigma/2})$ total memory, which is inefficient. To simplify the straightforward approach and reduce the round complexity, we design a constant-round subroutine to compute the routing table and combine algebraic strategies with recursive methods to compute the Steiner tree efficiently. By these techniques, we give the first parallel $2(1 - 1/|S|)$-approximate Steiner tree algorithm that requires $O(\sigma^{-1} \log n + D)$ rounds with the same memory size and the same approximation ratio. Moreover, we extend the straightforward approach to the MPC model with $O(n)$ memory per machine, which takes $O(\log n)$ rounds and significantly outperforms the existing algorithm [21] when $D \gg O(\log n)$.

Keywords: The MPC model · Steiner Tree · Round Complexity · SPF.

1 Introduction

Over the last several years, the study on graph algorithms in the *Massively Parallel Computation* (MPC) model has become the focus of growing interest, including graph connectivity [2], shortest paths [10, 11], and *minimum spanning tree* (MST) [9]. Furthermore, the significant development gap between storage capability and CPU performance causes communication to be a pivotal determinant of the algorithm's running speed. Hence, in the MPC model, reducing the communication complexity of graph algorithms is of utmost importance.

The *Steiner Tree* (ST) problem, a typical combinatorial optimization problem, has been widely applied in network design and computational biology [4].

* Corresponding author(qshua@hust.edu.cn); This work was supported in part by National Science and Technology Major Project 2022ZD0115301.

The ST problem is defined as finding a tree with the smallest edge weights, connecting all nodes in a specific set S of an undirected graph. Actually, the ST problem is inherently connected to the MST and shortest path problems. Concretely, it reduces to the MST problem when S contains all nodes and to the shortest path problem if S comprises only two nodes. However, unlike MST and shortest path problems that have polynomial-time solutions, the ST problem is NP-hard [12], which cannot be solved in polynomial time unless $P = NP$.

Thus, most researchers concentrate on the approximate solutions for the ST problem [22]. In the last four decades, there have been numerous studies on the *approximate Steiner tree* (AST) problem in the sequential model [17] and the parallel and distributed models [4], including the Congested Clique model [21] and the Congest model [6]. These highlight the necessity and importance of studying the ST problem.

Although the MPC algorithms designed for the MST or shortest path problems are fairly rich, as far as we know, no MPC algorithm has been proposed for the ST problem so far. In addition, the study of the ST problem can be extended to related optimization issues. For example, the traveling salesman problem, which is used extensively in logistics management and other fields [18]. Therefore, this paper focuses on the AST problem for undirected weighted dense graphs in the MPC model and aims to design an efficient MPC algorithm for it.

1.1 Our Contributions

In summary, our contributions are below:

- This paper proposes the first parallel AST algorithm in the MPC model (refer to Theorem 1 and Algorithm 4 in Section 5 for details).
- Compared with the straightforward approach (see Section 4.1), our parallel algorithm can greatly reduce the round complexity when the graph's shortest path diameter [21] is smaller than n, where n is the total number of nodes.
- Technically, we design a simpler subroutine to compute the routing table (see Definition 4 and Algorithm 2 in Section 5.1) and simplify the process of the straightforward approach by combining the algebraic methods and the recursive strategies, leading to a reduction in round complexity (see Section 5.2).

Next, we present the main results below, which can be readily proved by combining Lemma 3, Lemma 5, and the analysis in Section 6:

Theorem 1. *In the MPC model, each machine has $O(n^\sigma)$ memory ($\sigma \in (0,1)$ is a constant), given a graph $G = (V, E)$ with n nodes and a set S of terminals, there is a parallel algorithm that takes $O(\sigma^{-1} \log n + D)$ rounds and $O(n^{3-\sigma/2})$ total memory to solve the $2(1 - 1/|S|)$-approximate ST. Moreover, in the MPC model with $O(n)$ memory for each machine, the AST can be solved in $O(\log n)$ rounds and $O(n^{2.5})$ total memory.*

Then, we compare our algorithms with existing parallel AST algorithms. The results are outlined in Table 1. In the MPC model with $O(n^\sigma)$ memory assigned to each machine ($\sigma \in (0,1)$ is a constant), our parallel algorithm has a lower round complexity compared to the straightforward approach, particularly when

Table 1. Comparison between our parallel algorithms and the existing parallel approximate Steiner tree algorithms

Algo. (MPC($O(n^\sigma)$)†)	Round complexity	Total memory	Types of Algo.
The straightforward algorithm	$O(n)$	$O(n^{3-\sigma/2})$	Deterministic
Our work	$O(D+\sigma^{-1}\log n)$	$O(n^{3-\sigma/2})$	Deterministic
Algo. (MPC($O(n)$)‡)	Round complexity	Total memory	Types of Algo.
Saikia and Karmakar [21]	$O(\log\log n+D)$	$O(n^2)$	Deterministic
Our work	$O(\log n)$	$O(n^{2.5})$	Deterministic

Note: (1)† means the MPC model with $O(n^\sigma)$ memory per machine and $\sigma \in (0,1)$; (2) ‡ represents the MPC model with $O(n)$ memory for each machine.

$D \ll n$. Furthermore, when $D = O(\log n)$, the round complexity required by our algorithm is $O(\log n)$, which matches the best deterministic MST algorithm with $O(n^2)$ total memory in [9]. In the MPC model with $O(n)$ memory for each machine, when $D \gg O(\log n)$, our algorithm outperforms that of Saikia and Karmakar [21][1], with the cost of increasing the total memory. However, it still has a certain gap compared to the best MST algorithm in [19], which takes only constant rounds.

2 Related Work

Over the past thirty years, designing parallel and distributed AST algorithms [1] has been primarily based on the sequential algorithms from [17] and [22]. Among these, the algorithms designed for the Congested Clique model and the Congest model are most relevant to those in the MPC model.

In the Congest model, to solve the AST problem, Chen et al. [7] proposed the first distributed algorithm, which takes $O(n^2)$ rounds[2] and has an approximation ratio of $2(1-1/|S|)$. Then, Chatermosook et al. [6] presented a distributed 2-approximate ST algorithm with $O(n\log n)$ rounds. The two algorithms above are deterministic. Later, Khan et al. [16] designed a randomized ST algorithm with $O(\log n)$-approximation ratio, using $O(D\log^2 n)$ rounds. Next, Saikia and Karmakar [20] presented a deterministic $2(1-1/|S|)$-approximate ST algorithm that requires $O(D+\sqrt{n}\log^* n)$ rounds, combining the methods from [17] and [22]. Following this, for the ST problem with the same approximation ratio in the Congested Clique model, Saikia and Karmakar [21] designed the first deterministic algorithm with $O(D+\log\log n)$ rounds, using the steps in [20]. Recently, Kerger et al. [15] extended the $2(1-1/|S|)$-approximate ST problem to the quantum Congested Clique model and designed a randomized distributed algorithm using $O(n^{1/4})$ rounds.

[1] The algorithms designed for the Congested Clique model with $O(n)$ restricted memory per machine can be applied directly in the MPC model with $O(n)$ memory for each machine [3].

[2] We only show the rounds required for these distributed algorithms since the first goal in the MPC model is reducing the round complexity.

Compared to the nearly absent research on the AST problem in the MPC model, the MST problem has been well studied. For instance, in the MPC model with $O(n^\sigma)$ memory for each machine, Coy and Czumaj [9] proposed a deterministic parallel MST algorithm taking $O(\log n)$ rounds and $O(n+m)$ total memory or $O(\log D)$ rounds and $O((n+m)^{1+\Omega(1)})$ memory, where m is the number of edges in a graph. Moreover, the latter matches $\Omega(\log D)$ lower bound [9].

3 Preliminaries

In this section, we will first introduce the MPC model and the problem definitions. Then, we will describe the parallel *all-pairs shortest path* (APSP) algorithm designed by Hajiaghayi et al. [14] in Section 3.3, which is an essential component of our parallel AST algorithm.

3.1 The MPC Model

As a theoretical computation model, the MPC model evolves from the MapReduce model [13]. It contains three key parameters: the number P of machines required, the memory M of each machine, and the size N of the input, such that $P = \tilde{O}(N/M)$ ($\tilde{O}(\cdot)$ hides a logarithmic factor). The MPC algorithms are executed in synchronous rounds. Initially, the input data are distributed across P machines. Within each round, machines perform local computations without inter-machine communication. At the end of each round, machines send or receive messages with each other to obtain the data for the computation in next round. The data size sent or received from other machines is no larger than M. Machines communicate with each other in a pairwise interconnected manner.

Generally, the MPC model contains three memory settings [11]: 1) the strongly superlinear memory ($M = O(n^{1+\sigma})$, where $\sigma > 0$ is a constant); 2) the near-linear memory ($M = O(n)$); 3) the strongly sublinear memory ($M = O(n^\sigma)$, where $\sigma \in (0, 1)$ is a constant). This paper considers the latter two memory settings. In particular, we focus on designing parallel algorithms in the MPC model with strongly sublinear memory, which is more difficult and more scalable.

Complexity Measurements: In the MPC model, the first and most critical objective is to minimize an algorithm's round complexity. Reducing the total memory required by the algorithm is the secondary goal.

There are many useful and efficient subroutines in the MPC model with strongly sublinear memory, such as sorting an ordered set, broadcasting messages from one machine to other machines, and finding the maximum or minimum in a set, which all take only $O(\sigma^{-1})$ rounds [10, 13].

3.2 The Problem Definitions

This paper mainly considers the undirected connected weighted dense graphs $G = (V, E, W)$. Specifically, V denotes the set of n nodes, E represents the set of m edges, and $W : E \to \mathbb{R}^+$ is the weight function. When $m = \Theta(n^2)$, it is a

dense graph. For any two nodes $u_1, u_2 \in V$, $d(u_1, u_2)$ denotes the distance from u_1 to u_2. In addition, in the MPC model, the IDs of the nodes in V are marked as $1, 2, \cdots, n$, and the edges related to a node, ordered by the ID of another endpoint, are stored in a continuous set of machines. Namely, the node and its edges with smaller IDs will be stored in the machines with smaller IDs [10].

Definition 1 (The ST problem). *Considering the graphs above, given a node (or terminal) subset $S \subseteq V$, the ST problem is defined as finding a tree $T_0 = (V_0, E_0)$ that minimizes $\sum_{e \in E_0} w(e)$, where $S \subseteq V_0 \subseteq V$ and $E_0 \subseteq E$.*

It is worth mentioning that the nodes belonging to $V \setminus S$ are called the non-terminals and the nodes that belong to $V_0 \setminus S$ are known as Steiner nodes. Next, we introduce some important definitions for computing the AST.

Definition 2 (The *Shortest Path Forest* (SPF) [7, 21]). *Given a graph $G = (V, E, W)$ above and a node subset S in Definition 1, the SPF for S, denoted as $G_{SPF} = (V, E_{SPF}, w)$, is a subgraph of G that comprises $|S|$ disjoint trees $T_j = (V_j, E_j, W)$, where $j \in \{1, 2, \cdots, |S|\}$. Moreover, these trees satisfy the following conditions: 1) Each V_j contains only one node s_j from S; 2) For any node $v \in V_j$, s_j is the unique source of v, where $s_j \in S$; 3) $\bigcup_{j=1}^{|S|} V_j = V$ and different node subsets are pairwise disjoint; 4) $\bigcup_{j=1}^{|S|} E_j = E_{SPF} \subseteq E$; 5) The shortest path from $v \in V_j$ to $s_j \in V_j$ in T_j is the shortest path from v to s_j in G.*

Definition 3 (The *Complete Distance Graph* (CDG) [17, 22]). *Considering the graphs above and the subset S in Definition 1, the CDG, denoted by $K_S = (V, E')$, consists of all nodes in G and is connected, undirected, and weighted. The weight of each edge $(u_1, u_2) \in E'$ is the corresponding shortest distance between u_1 and u_2 in the original graph G.*

Definition 4 (The *Routing Table* (RT) [5]). *Given a graph $G = (V, E)$ above and its adjacency matrix A_0 of size $n \times n$, the RT R is an $n \times n$ matrix. Each element $R[u_1, u_2] = z$ in R, where $u_1, u_2, z \in V$, satisfies that $(u_1, z) \in E$ and z is the first intermediate node on a shortest path from u_1 to u_2.*

3.3 A Known Parallel APSP Algorithm

Next, we introduce the parallel exact APSP algorithm from [14], which is Algorithm 1 below. The primary idea of Algorithm 1 is to iteratively compute the product $A \star A$ for $\lceil \log n \rceil$ times, where A is the $n \times n$ adjacency matrix of the graph $G = (V, E)$. In addition, $\star$ represents the multiplication over semiring $(\min, +)$. That is to say, for any nodes u_1, u_2 in V, $(A \star A)_{u_1, u_2} = \min_{z \in V}\{A_{u_1, z} + A_{z, u_2}\}$ (see lines 4 - 5 of Algorithm 1). After $\lceil \log n \rceil$ iterations (see line 1 of Algorithm 1), we obtain the distance matrix A^n (replaced by *Dist* in line 7 of Algorithm 1). Lemma 1 below presents the rounds required by Algorithm 1.

Lemma 1 (Algorithm 1 [14]). *In the MPC model with $O(n^\sigma)$ memory for each machine ($\sigma \in (0, 1]$ is a constant), a graph $G = (V, E)$ that has n nodes and its adjacency matrix A, Algorithm 1 computes the distance matrix Dist using $O(\sigma^{-1} \log n)$ rounds and $O(n^{3-\sigma/2})$ total memory.*

Algorithm 1 The Parallel APSP algorithm [14]

Input: An $n \times n$ matrix A distributed among $O(n^{2-\sigma})$ machines numbered as $P_{i,j,1}$, each of which has a memory size $O(n^{\sigma})$, where $i, j \in \{1, 2, \cdots, n^{1-\sigma/2}\}$.
Output: The distance matrix $Dist$
1: **for** $h = 1, 2, \cdots, \lceil \log n \rceil$ **do**
2: The machine $P_{i,j,1}$ broadcasts the sub-matrix $A_{i,j}^{2^{h-1}}$ with size $n^{\sigma/2} \times n^{\sigma/2}$ to other $O(n^{1-\sigma/2})$ machines numbered by $P_{i,j,k}$, where $k \in \{2, 3, \cdots, n^{1-\sigma/2}\}$. ▷ If $h = 1$, then $A_{i,j}^{1} = A_{i,j}$
3: The machine $P_{i,j,1}$ broadcasts the sub-matrix $A_{i,j}^{2^{h-1}}$ to other $O(n^{1-\sigma/2})$ machines numbered by $P_{k,i,j}$, where $k \in \{1, 2, \cdots, n^{1-\sigma/2}\}$.
4: Each machine $P_{i,j,k}$ computes $A_{i,k}^{2^h} = A_{i,j}^{2^{h-1}} \star A_{j,k}^{2^{h-1}}$, where $i, j, k \in \{1, 2, \cdots, n^{1-\sigma/2}\}$. ▷ $A_{i,j}^{2^{h-1}} \star A_{j,k}^{2^{h-1}}$ represents the multiplication of two matrices $A_{i,j}^{2^{h-1}}$ and $A_{j,k}^{2^{h-1}}$ over semiring $(\min, +)$
5: Execute the broadcast operation among machines numbered by $P_{i,j,k}$ to compute partial sums of sub-matrix $A_{i,k}^{2^h}$ to obtain $A_{i,k}^{2^h}$ $(i, j, k \in \{1, 2, \cdots, n^{1-\sigma/2}\})$.
6: **end for**
7: Return $Dist = A^n$

Proof. As we can see, in only one round, Algorithm 1 can compute all the partial sums of all pairs of nodes (refer to line 4 in Algorithm 1). Since there are only broadcast operations in each iteration (refer to lines 2, 3, and 5 in Algorithm 1), it takes only constant rounds. Therefore, we obtain the round complexity for Algorithm 1, as shown in Lemma 1.

4 Techniques

4.1 A Straightforward Parallel AST Algorithm

The general steps of addressing the AST problem in the Congested Clique model [21] and the Congest model [20] are below: **1) Construct the SPF; 2) Modify the weights of edges in the graph; 3) Construct the MST on the modified graph; 4) Delete the non-terminal leaves from the MST.** When exact algorithms are used for each step [15, 20, 21], the approximation ratio of the ST is $2(1 - 1/|S|)$. Through these steps, we obtain a *straightforward AST* (SAST) algorithm in the MPC model with strongly sublinear memory below:

Step 1: Initially, we compute the CDG (see Definition 3 in Section 3.2) using Algorithm 1. Then, we solve the RT (refer to Definition 4 in Section 3.2). Specifically, we denote the RT for $A^{2^j} = A^{2^{j-1}} \star A^{2^{j-1}}$ as R_j $(j \in \{1, 2, \cdots, \lceil \log n \rceil\})$. According to [5], if R_j is known, then for $A^{2^{j+1}} = A^{2^j} \star A^{2^j}$, $R_{j+1}[u_1, u_2] = R_{j+1}[u_1, R_j[u_1, u_2]]$, where $u_1, u_2 \in V$. The resulting RT is $R = R_{\lceil \log n \rceil}$. This is the same as computing the distance matrix in Algorithm 1. Thus, the rounds and total memory needed in this step are $O(\sigma^{-1} \log n)$ and $O(n^{3-\sigma/2})$, respectively.

Finally, we construct the SPF (the $|S|$ disjoint trees $T_j = (V_j, E_j)$, where $j \in \{1, 2, \cdots, |S|\}$, as defined in Definition 2). Specifically, we determine the

source of every node $v \in V$ by comparing all distances from v to the terminals in S, using the distance matrix $Dist$. It requires $O(\sigma^{-1})$ rounds and $O(|S|n)$ total memory. Next, we can determine the edges in the tree T_j by the RT R in only one round (for each node v in T_j, we find and mark its connecting edge (u, v), where u is the direct predecessor.). This is implied by the fact that the subpath of the shortest path is also the shortest path [8], so u has the same source as v.

Step 2: We modify the edge weights of the original graph G as follows: 1) Set all tree edges with a weight of 0; 2) Set the weights of all non-tree edges whose endpoints have the same source to ∞; 3) For any edge (u_1, u_2) whose endpoints have different sources, we set its weight to $w'(u_1, u_2) = d(s_i, u_1) + w(u_1, u_2) + d(u_2, s_j)$, where s_i and s_j are the sources of u_1 and u_2 ($i, j \in \{1, 2, \cdots, |S|\}$ and $i \neq j$), respectively. Next, we analyze the round complexity required in this step. Since each node must modify its edge weights using information from all other nodes, which is at most n. As the memory of each machine is $O(n^\sigma)$ ($\sigma \in (0, 1)$), it takes $O(n^{1-\sigma}/\sigma)$ rounds and $O(n^2)$ total memory using the broadcast operation.

Step 3: We solve the MST on the modified graph with an existing deterministic MPC algorithm [9], which takes $O(\log n)$ rounds and $O(n^2)$ total memory.

Step 4: We prune the MST computed in Step 3. This results in at most n updates since deleting a non-terminal leaf will cause another new non-terminal leaf [1]. Thus, it takes up to $O(n)$ rounds and $O(n)$ total memory.

In summary, this algorithm takes $O(n)$ rounds and $O(n^{3-\sigma/2})$ total memory. In addition, the SAST algorithm computes the exact result for the $2(1 - 1/|S|)$-approximate Steiner tree without incurring any additional approximation ratio.

4.2 Challenges and Solutions

The challenges in reducing the rounds of the SAST algorithm are below:

Firstly, Step 2 is too complex and causes a high round complexity, which is inefficient. Simplifying this step and reducing the required rounds is important.

Secondly, the round complexity of deleting the non-terminal leaves in Step 4 is too high. It is a key challenge to reduce the round complexity.

The solutions to the challenges above are as follows:

To tackle the first challenge, we combine steps 2 and 3 in Section 4.1 into one step. We combine the recursive strategy with algebraic methods to connect these $|S|$ disjoint trees and then obtain the required MST. This simplifies both Step 2 and Step 3 and reduces the round complexity for computing the MST.

To address the second challenge, we mark the endpoints of these connecting edges that connect these $|S|$ trees. Then, using these endpoints and the routing table, we find the Steiner nodes, which reduces the round complexity when a graph's shortest path diameter is smaller than n.

Additionally, we design a simpler and novel algorithm to calculate the RT, which takes only constant rounds. This reduces the round complexity of the parallel algorithm that computes the RT in Section 4.1.

5 The Parallel Approximate Steiner Tree Algorithm

According to the strategies in Section 4.2, this section first computes the SPF in Section 5.1. Then, in Section 5.2, we address the AST problem.

5.1 The Construction of the SPF

The process of computing the SPF is similar to Step 1 in Section 4.1. The key difference is a new parallel algorithm designed for the RT.

Algorithm 2 The parallel RT algorithm

Input: The $n \times n$ distance matrix $Dist$, a matrix A with size $n \times n$, and the predecessor matrix $\prod^A$ of A, distributed among $O(n^{2-\sigma})$ machines numbered as $P_{i,j,1}$, where $i, j \in \{1, 2, \cdots, n^{1-\sigma/2}\}$, each of which has a memory size $O(n^\sigma)$.

Output: The routing table $R_{i,k}$ with size $n \times n$.

1: The machine $P_{i,j,1}$ broadcasts the sub-matrix $A_{i,j}$ with size $n^{\sigma/2} \times n^{\sigma/2}$ to other $O(n^{1-\sigma/2})$ machines numbered by $P_{i,j,k}$, where $k \in \{2, 3, \cdots, n^{1-\sigma/2}\}$.
2: The machine $P_{i,j,1}$ broadcasts the sub-matrix $Dist_{i,j}$ to other $O(n^{1-\sigma/2})$ machines numbered by $P_{k,i,j}$, where $k \in \{1, 2, \cdots, n^{1-\sigma/2}\}$
3: The machine $P_{i,j,1}$ broadcasts the $n^{\sigma/2} \times n^{\sigma/2}$ sub-matrices $\prod^A_{i,j}$ and $Dist_{i,j}$ to other $O(n^{1-\sigma/2})$ machines numbered by $P_{i,k,j}$, where $k \in \{2, 3, \cdots, n^{1-\sigma/2}\}$.
4: **for** all machines $P_{i,j,k}$ $(i, j, k \in \{1, 2, \cdots, n^{1-\sigma/2}\})$ execute locally in parallel **do**
5: Set an empty matrix R with size $n^{\sigma/2} \times n^{\sigma/2}$
6: **for** $i' = 1, 2, \cdots, n^{\sigma/2}$ **do**
7: **for** $k' = 1, 2, \cdots, n^{\sigma/2}$ **do**
8: **for** $j' = 1, 2, \cdots, n^{\sigma/2}$ **do**
9: **if** $i' = k'$ **then** Set $R_{i,k}[i', k'] = \prod^A_{i,k}[i', k']$ and stop
10: **end if**
11: $d(i', k') = A_{i,j}(i', j') + Dist_{j,k}(j', k')$
12: **if** $d(i', k') = Dist_{i,k}(i', k')$ and $i' \neq j'$ **then** Set $R_{i,k}[i', k'] = j'$ and stop ▷ when $i' = j'$, this avoids the case of $Dist_{i,k}(i', k') = Dist_{j,k}(j', k')$, indicating equal distances between i'/j' and k'.
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: **end for**
18: Execute the broadcast operation among machines numbered by $P_{i,j,k}, j \in \{1, 2, \cdots, n^{1-\sigma/2}\}$ to obtain the final predecessor matrix $R_{i,k}$.
19: Return R ▷ R is made up of $n^{2-\sigma}$ submatrices $R_{i,k}$ of size $n^{\sigma/2} \times n^{\sigma/2}$.

A Simpler Algorithm for the RT: We compute the RT only once, using the distance matrix $Dist$ and the adjacency matrix A of G (Algorithm 2), instead of computing it $\lceil \log n \rceil$ times as in Step 1 in Section 4.1. First, we initialize the predecessor matrix $\prod^A$ of A such that for $u_1, u_2 \in V$, if $e(u_1, u_2) \in E$,

then $\prod^A[u_1, u_2] = u_2$, otherwise, $\prod^A[u_1, u_2] = NIL$. Also, we set $\prod^A[u_1, u_1] = NIL$. Second, we compute $A \star Dist$ (line 12 in Algorithm 2), modifying the elements of $\prod^A$ accordingly. The resulting predecessor matrix $R = \prod^A$ is the RT. The principle behind this is simple: 1) $Dist = A \star Dist$ means that the shortest distances between nodes will not change anymore; 2) $(A \star D)_{u_1,u_2} = \min_{z \in V}\{w(u_1, z) + d(z, u_2)\}$ determines u_1's predecessor. The concrete process is shown in Algorithm 2. Moreover, we give a simple example in Fig. 1 to better understand Algorithm 2. Next, we analyze the rounds of Algorithm 2.

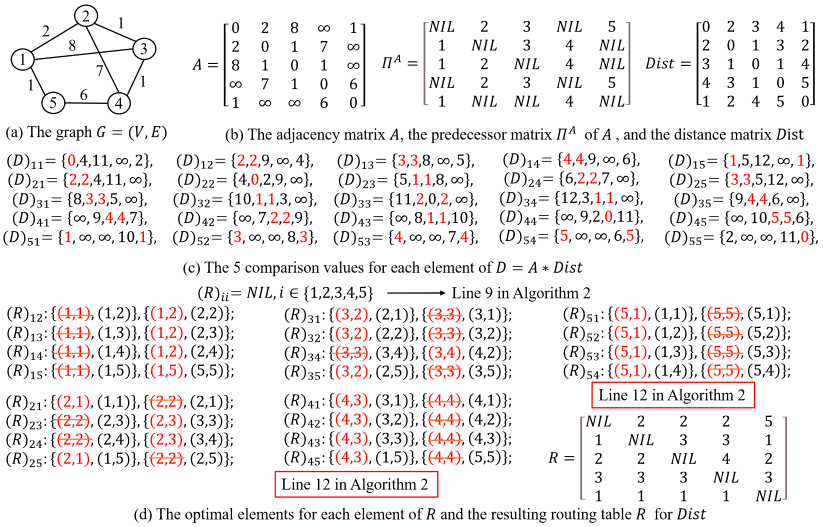
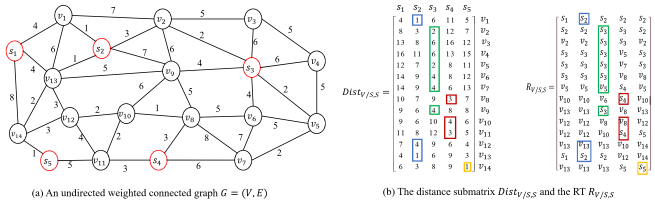


Fig. 1. A concrete example of Algorithm 2

Lemma 2 (Algorithm 2). *In the MPC model with $O(n^\sigma)$ memory ($\sigma \in (0, 1]$) for each machine, for a graph $G = (V, E)$ that has n nodes, the adjacency matrix A, A's predecessor matrix $\prod^A$, and the distance matrix $Dist$, Algorithm 2 computes the RT R of $Dist$ using $O(\sigma^{-1})$ rounds and $O(n^{3-\sigma/2})$ total memory.*

Proof. As shown in Algorithm 2, there are only broadcast operations in lines 1 - 3 and line 20, which cost $O(\sigma^{-1})$ rounds. There is no communication between machines in lines 4 - 19 since each machine executes local computation. Similar to the computation of APSP in Algorithm 1, Algorithm 2 requires $O(n^{3-\sigma/2})$.

The Parallel SPF Algorithm: Algorithm 3 computes the SPF in two steps. First, it computes the CDG (or $Dist$) in line 1 and the related RT R in line 2. Second, it finds the source of each node $v \in V/S$ by selecting the minimum distance from v to S in line 3. Using the RT, it constructs the tree T_j rooted



(a) An undirected weighted connected graph $G = (V, E)$

(b) The distance submatrix $Dist_{V/S,S}$ and the RT $R_{V/S,S}$

Fig. 2. An illustrative instance for Algorithm 3

at $s_j \in S$ ($j \in \{1, 2, \cdots, |S|\}$) in line 4. For clarity, we provide an illustrating example for Algorithm 3 in Fig. 2.

Algorithm 3 The parallel SPF algorithm

Input: A given undirected connected graph $G = (V, E, W)$ and the terminal set S.
Output: The SPF $G_{SPF} = (V, E_{SPF}, w)$.

1: Execute Algorithm 1: compute the distance matrix $Dist$.
2: Execute Algorithm 2: compute the RT R of the distance matrix $Dist$.
3: The machines storing the distances of each $v \in V/S$ and nodes in S find v's source.
4: The machine that stores the tree edge of $v \in V/S$ whose source is $s \in S$, sends the tree edge and the distance $d(s, u)$ to the machine that stores $s \in S$.

Lemma 3. *In the MPC model, each machine has $O(n^\sigma)$ memory and $\sigma \in (0, 1)$, for a graph G with n nodes and a terminal set S, Algorithm 3 computes the SPF $G_{SPF} = (V, E_{SPF})$ taking $O(\sigma^{-1} \log n)$ rounds and $O(n^{3-\sigma/2})$ total memory.*

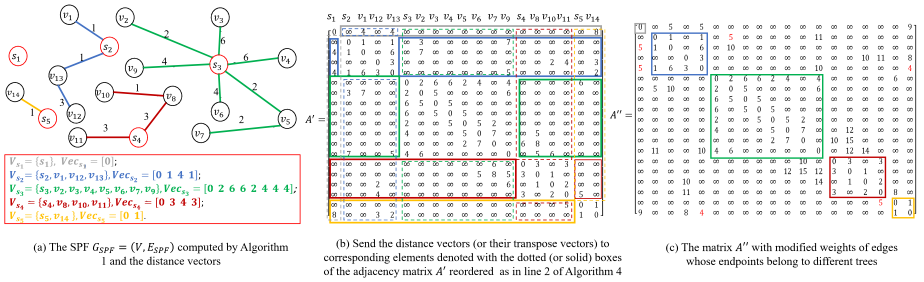
Proof. By Lemmas 1 - 2, it requires $O(\sigma^{-1} \log n)$ rounds and $O(n^{3-\sigma/2})$ memory to run lines 1 - 2 in Algorithm 3. Sorting these distances and finding the minimum in line 3 takes $O(1/\sigma)$ rounds. Line 4 requires $O(1)$ rounds to send messages.

5.2 The Computation of the AST

This subsection computes the AST in the MPC model using the first two strategies proposed in Section 4.2, which will be detailed in Algorithm 4.

In Algorithm 4, we first label the states of all nodes in line 1, which are used to identify the Steiner nodes later. Then, we reorder the nodes whose IDs are the row and column indices of the adjacency matrix A such that querying the edges with endpoints in different trees (line 2) is more convenient. Next, we modify the weights of edges with endpoints in distinct trees in lines 3 - 5 (see Fig.3(b)), the same as in Section 4.1. We select the edge with minimum weight connecting T_i ($i \in \{1, 2, \cdots, |S|\}$) to its closest tree (not T_i) in line 6 (see Fig.3(c)). In lines 7 - 9 (refer to Fig. 4(a)), we delete the heaviest edge that causes a cycle or a duplicate edge among the $|S|$ trees. We find the necessary connecting edges (line 10 and Fig. 4(b)) and modify the states of non-terminals that are Steiner

nodes (line 11). Since we utilize the RT, there is no resource contention between machines when multiple edges are selected for a tree T. Finally, we delete the non-terminal leaves in G_{SPF} in line 12 and obtain the AST (refer to Fig. 4(c)). Now, we analyze the correctness and round complexity for Algorithm 4.

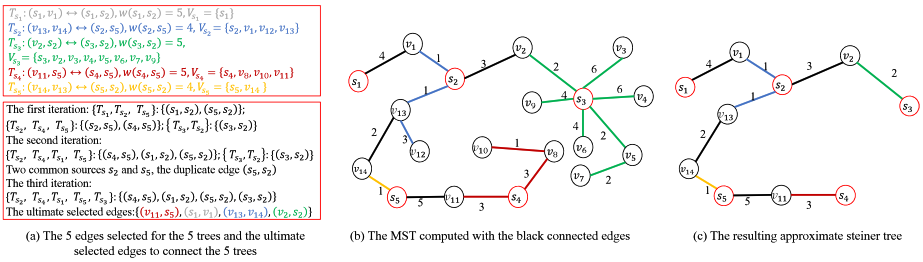


(a) The SPF $G_{SPF} = (V, E_{SPF})$ computed by Algorithm 1 and the distance vectors

(b) Send the distance vectors (or their transpose vectors) to corresponding elements denoted with the dotted (or solid) boxes of the adjacency matrix A' reordered as in line 2 of Algorithm 4

(c) The matrix A'' with modified weights of edges whose endpoints belong to different trees

Fig. 3. A straightforward example of the process of lines 2 - 6 in Algorithm 4



(a) The 5 edges selected for the 5 trees and the ultimate selected edges to connect the 5 trees

(b) The MST computed with the black connected edges

(c) The resulting approximate steiner tree

Fig. 4. A concrete instance of the process of lines 8 - 12 in Algorithm 4

Lemma 4. *Algorithm 4 can compute an MST correctly without line 10 in Algorithm 4. Additionally, the approximation ratio of the ST computed is* $2(1-1/|S|)$.

Proof. The core ideas behind the SAST algorithm and Algorithm 4 for computing the MST are the same. Firstly, the MSTs computed by the two algorithms both contain all the tree edges: the SAST algorithm sets the weights of all edges in G_{SPF} to 0 and Algorithm 4 retains the edges in G_{SPF}. Secondly, the two algorithms both modify the weights of edges whose endpoints are in different trees in the same way. The difference is as follows: each time, the SAST algorithm chooses the minimum weight edge. However, Algorithm 4 selects the minimum weight edge between each tree $T_i (i \in \{1, 2, \cdots, |S|\})$ and its closest tree. Consequently, we have $|S|$ selected edges, which may cause a cycle. However, Algorithm 4 either deletes the edge with maximum weight in the cycle or the duplicate edge

in lines 7 - 9. This shows that Algorithm 4 computes the MST correctly. Thus, the approximation ratio of the ST solved by Algorithm 4 is $2(1-1/|S|)$, which is the same as that of the SAST algorithm.

Algorithm 4 The parallel AST algorithm

Input: The SPF $G_{SPF} = (V, E_{SPF}, w) = \bigcup_{j=1}^{|S|}(T_j = (V_j, E_j))$ obtained by Algorithm 3, the adjacency matrix A, the RT R, and the terminal set S.

Output: The AST T_0.

1: Label the states of the terminals in S and nodes in V/S as $true$ and $false$, respectively.
2: Reorder the row and column indices in A according to the ID values of terminals in S, each of which is followed by the ordered IDs of its child nodes. ▷ see Fig. 3(b)
3: The machines that store the $1 \times |V_{T_i}|$ distance vectors from $s_i \in S$ to nodes in the tree T_i ($i \in \{1, 2, \cdots, |S|\}$), broadcast them to the machines that store the $|V_{T_i}| \times |V_{T_j}|$ submatrix $A[V_{T_i}, V_{T_j}]$, where $j \neq i$ and $j \in \{1, 2, \cdots, |S|\}$. ▷ $A[V_{T_i}, V_{T_j}]$ represents the submatrix of A with row indices in V_{T_i} and column indices in V_{T_j}.
4: The machines that store the $|V_{T_i}| \times 1$ distance vectors from nodes in the tree T_i ($i \in \{1, 2, \cdots, |S|\}$) to $s_i \in S$, broadcast them to the machines that store the $|V_{T_j}| \times |V_{T_i}|$ submatrix $A[V_{T_j}, V_{T_i}]$, where $j \neq i$ and $j \in \{1, 2, \cdots, |S|\}$.
5: The machines that store $A[V_{T_i}, V_{T_j}]$, add the elements of $A[V_{T_i}, V_{T_j}]$ with the corresponding elements in the distance vectors, where $i, j \in \{1, 2, \cdots, |S|\}$ and $i \neq j$. ▷ Modify the weights of edges whose endpoints belong to different trees.
6: The machines that store the modified submatrix $A[V_{T_i}, V/V_{T_i}]$ choose the minimum value of $A[V_{T_i}, V/V_{T_i}]$. ▷ Find the connecting edge for each tree to its closest tree.
7: Invoke other $O(|S|/n^{\sigma})$ machines (or one machine if $|S| \leq n^{\sigma}$) to receive the $|S|$ selected edges for the $|S|$ disjoint trees sent by the machines that store them.
8: The invoked machines modify the endpoints of each selected edge to their sources and then match these edges that have a common endpoint recursively, until there are two common sources. ▷ Find the duplicate edge or the cycle among the trees.
9: The invoked machines either delete the duplicate edge or compare all edge weights in the cycle and find the heaviest edge $e = (s_i, s_j)$ and delete the corresponding edge $e = (i, j)$. ▷ Delete the heaviest edge in the cycle or the duplicate edge.
10: The machines that store the selected edges $e = (i, j)$, send these edges to the machines that store the tree T_i and its closest tree T_j, where $i, j \in \{1, 2, \cdots, |S|\}$ and $i \neq j$. ▷ $e = (i, j)$ connects T_i and T_j, with $i \in T_i$ and $j \in T_j$, respectively.
11: The machines that store the tree T_i (T_j), find the nodes on the path from i (j) to the root of T_i (T_j, where $i, j \in \{1, 2, \cdots, |S|\}$ and $i \neq j$), using R. Meanwhile, these machines modify the states of these nodes to $true$. ▷ Find the Steiner nodes.
12: Delete all nodes in G_{SPF} whose states are $false$ and their corresponding edges. ▷ Delete the leaves in G_{SPF} that are non-terminals.
13: Return $T_0 = G_{SPF}$

Lemma 5. *In the MPC model, each machine has $O(n^{\sigma})$ ($\sigma \in (0, 1)$) memory, for the graph $G = (V, E)$ that has n nodes, a terminal set S, the adjacency*

matrix A, the RT R, and the SPF $G_{SPF} = (V, E_{SPF})$ computed by Algorithm 3, Algorithm 4 solves the AST in $O(\log |S| + D)$ rounds and $O(n^2)$ total memory.

Proof. In Algorithm 4, executing line 1 takes only one round because each terminal in S is a root of a tree in $G_{SPF} = (V, E_{SPF})$. There is only a sorting operation in line 2 of Algorithm 4, which takes $O(\sigma^{-1})$ rounds. In lines 3 - 4 and 10, the broadcast operations require $O(\sigma^{-1})$ rounds. Finding the minimum and maximum weight edges in lines 5, 6, and 9 takes $O(\sigma^{-1})$ rounds. Dealing with $|S|$ edges recursively in line 8 requires $O(\log |S|)$ rounds. It takes only one round to send messages to the corresponding machines in line 10. The most expensive operation is to find the nodes on a path in line 11 of Algorithm 4, which requires $O(D)$ rounds since the graph's shortest path diameter is D. The total memory for this algorithm is $O(n^2)$, as the size of the adjacency matrix A is $O(n^2)$.

6 Extension to the Near-linear Memory Setting

Now, we extend the straightforward method of computing AST to the MPC model with $O(n)$ memory for each machine. The four-step process remains unchanged: 1) In Step 1, we obtain the CDG and the RT R using Algorithm 1 and Algorithm 2, respectively. The subsequent process follows Section 4.1. We can store the edges of the $|S|$ disjoint trees in a single machine since the number of edges in these trees is less than n. By Lemmas 1 - 2, Step 1 takes $O(\log n)$ rounds and $O(n^{2.5})$ total memory. 2) For Step 2, since each node contains up to n edges, we can modify the edge weights of each node in a single machine, which costs only one round. As there are n nodes, the total memory needed is $O(n^2)$. 3) In Step 3, we utilize the existing fastest parallel MST algorithm proposed by Coy and Czumaj [9], requiring $O(\log n)$ rounds and $O(n^2)$ total memory. 4) Step 4 can be completed in one round, as a single machine is sufficient to store all edges of the MST.

Hence, we obtain a parallel AST algorithm in the MPC model with near-linear memory using $O(\log n)$ rounds and $O(n^{2.5})$ total memory. This greatly improves the result in [21], which takes $O(\log \log n + D)$ rounds when $D \gg \log n$.

7 Conclusion

This paper studies the approximate Steiner tree problem in the MPC model with $O(n^\sigma)$ memory for each machine and $\sigma \in (0, 1)$. Specifically, to reduce the round complexity of the straightforward approximate Steiner tree algorithm, we use a simpler method to compute the routing table that takes only constant rounds and combine the recursive strategy with the algebraic methods to solve the approximate Steiner tree more efficiently. Then, we obtain a parallel $2(1 - 1/|S|)$-approximate Steiner tree algorithm that takes $O(D + \sigma^{-1} \log n)$ rounds and $O(n^{3-\sigma/2})$ total memory. Furthermore, we extend the straightforward approximate Steiner tree algorithm to the MPC model with $O(n)$ memory for each machine, requiring $O(\log n)$ rounds. This reduction in round complexity is highly beneficial when $D = \Omega(\log n)$.

References

1. Akbari, H., Iranmanesh, Z., Ghodsi, M.: Parallel minimum spanning tree heuristic for the steiner problem in graphs. In: Proc. ICPADS. pp. 1–8. IEEE (2007)
2. Andoni, A., Song, Z., Stein, C., Wang, Z., Zhong, P.: Parallel graph connectivity in log diameter rounds. In: Proc. FOCS. pp. 674–685. IEEE (2018)
3. Behnezhad, S., Derakhshan, M., Hajiaghayi, M.: Brief announcement: Semi-mapreduce meets congested clique. CoRR **abs/1802.10297** (2018)
4. Bezensek, M., Robic, B.: A survey of parallel and distributed algorithms for the steiner tree problem. Int. J. Parallel Program. **42**(2), 287–319 (2014)
5. Censor-Hillel, K., Kaski, P., Korhonen, J.H., Lenzen, C., Paz, A., Suomela, J.: Algebraic methods in the congested clique. In: Proc. PODC. pp. 143–152. ACM (2015)
6. Chalermsook, P., Fakcharoenphol, J.: Simple distributed algorithms for approximating minimum steiner trees. In: Proc. COCOON. pp. 380–389. Springer (2005)
7. Chen, G., Houle, M.E., Kuo, M.: The steiner problem in distributed computing systems. Inf. Sci. **74**(1-2), 73–96 (1993)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press (2009)
9. Coy, S., Czumaj, A.: Deterministic massively parallel connectivity. In: Proc. STOC. pp. 162–175. ACM (2022)
10. Dinitz, M., Nazari, Y.: Massively parallel approximate distance sketches. In: Proc. OPODIS. pp. 35:1–35:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2019)
11. Dory, M., Matar, S.: Massively parallel algorithms for approximate shortest paths. In: Proc. SPAA. pp. 415–426. ACM (2024)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
13. Goodrich, M.T., Sitchinava, N., Zhang, Q.: Sorting, searching, and simulation in the mapreduce framework. In: Proc. ISAAC. pp. 374–383 (2011)
14. Hajiaghayi, M., Lattanzi, S., Seddighin, S., Stein, C.: Mapreduce meets fine-grained complexity: Mapreduce algorithms for apsp, matrix multiplication, 3-sum, and beyond. CoRR **abs/1905.01748** (2019)
15. Kerger, P.A., Neira, D.E.B., Izquierdo, Z.G., Rieffel, E.G.: Quantum distributed algorithms for approximate steiner trees and directed minimum spanning trees. In: Proc. QCE. pp. 1249–1259. IEEE (2023)
16. Khan, M., Kuhn, F., Malkhi, D., Pandurangan, G., Talwar, K.: Efficient distributed approximation algorithms via probabilistic tree embeddings. In: Proc. PODC. p. 263–272. ACM (2008)
17. Kou, L.T., Markowsky, G., Berman, L.: A fast algorithm for steiner trees. Acta Informatica **15**, 141–145 (1981)
18. Lenzen, C., Patt-Shamir, B.: Improved distributed steiner forest construction. In: Proc. PODC. pp. 262–271. ACM (2014)
19. Nowicki, K.: A deterministic algorithm for the MST problem in constant rounds of congested clique. In: Proc. STOC. pp. 1154–1165. ACM (2021)
20. Saikia, P., Karmakar, S.: A simple $2(1\text{-}1/l)$ factor distributed approximation algorithm for steiner tree in the congest model. In: Proc. ICDCN. pp. 41–50. ACM (2019)
21. Saikia, P., Karmakar, S.: Distributed approximation algorithms for steiner tree in the congested clique. Int. J. Found. Comput. Sci. **31**(7), 941–968 (2020)
22. Wu, Y., Widmayer, P., Wong, C.K.: A faster approximation algorithm for the steiner problem in graphs. Acta Informatica **23**(2), 223–229 (1986)