# Algorithms for some Steiner tree problems on Graphs

***Thesis submitted to the***
***Indian Institute of Technology Guwahati***
***for the award of the degree***

**of**

**Doctor of Philosophy**

in

**Computer Science and Engineering**

Submitted by

**Parikshit Saikia**

Under the guidance of

**Dr. Sushanta Karmakar**

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

November, 2020

***Dedicated to***

***My beloved Grandparents***

***and***

***Parents***

*Who always picked me up on time*
*and encouraged me to go on every adventure,*
*specially this one*

# Declaration

I certify that:

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor is not in a position to check for any possible instance of plagiarism within this submitted work.

Date: November 15, 2020
Place: Guwahati

Parikshit Saikia

**(Parikshit Saikia)**

# Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor *Dr. Sushanta Karmakar* for his consistent support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for his ethical beliefs and philosophy which made me mature not only as a scientific researcher but also as a human.

I am highly grateful to *Prof. Purandar Bhaduri* for his invaluable support and encouragement throughout my Ph.D. I would also like to thank the other members of my Doctoral Committee - *Dr. Pinaki Mitra* and *Dr. Gautam K. Das* for their insightful comments and suggestions which made me improve the quality and clarity of my work.

I would like to thank *Prof. Aris T. Pagourtzis* for the insightful comments and suggestions he made in reference to Chapter 5 of this work. I am also thankful to the anonymous reviewers of my research work in various forums for their critical comments which helped me to add quality to my work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IITG - *Prof. Diganta Goswami* and *Prof. S. V. Rao* for allowing me to use the facilities and the available resources. I acknowledge the Technical staff of the Department of Computer Science and Engineering - Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah, Mr. Hemanta Kumar Nath, Mr. Pranjitt Talukdar, Mr. Raktajit Pathak and Mr. Nava Kumar Boro for solving any engineering related issues. I am deeply thankful to - Mr. Monojit Bhattacharjee, Ms. Gauri Khuttiya Deori and Mr. Prabin Bharali for efficiently handling the administrative work. I am obliged to all the faculty members, the staff and security personnel for their constant help and support. I would also like to thank the satff at the Academic Affairs office who were supportive to process my applications and grant requests.

I would like to gratefully acknowledge *MHRD*, *Govt. of India* for the financial support rendered throughout my years of Ph.D. without which this research could not have taken shape. I would also like to acknowledge the department of Computer Science and Engineering and the Welfare Board of IITG for the travel grants which helped me to present my research work at the national and international levels.

I would like to specially mention *Dr. Sirumoni Baruah* for his persistent support and guidance from the very beginning of my Ph.D. life. I am indeed thankful to my fellow lab mates at the Research Lab 2. - Alakesh, Surajit, Arunagshu, Partha, Subrata, Madhurima, Neelakshi, Pawan, Swarup, Rakesh, Yashdeep, Hemraj, and Arijit for creating a wonderful experience at my workplace. The stimulating discussions, brainstorming and sleepless night working together contribute a significant portion towards my development as an independent researcher. I would also like to thank the research scholars of the department of Computer Science and Engineering at IITG for creating a warm atmosphere of mutual support and encouragement.

I am fortunate to have good friends - Manojit, Himangshu, Pranab, Kukil, Himangshu-pratim, Ashis, Dipanka, Dwijasish, Dhriti, Shrutidhara, Preetirekha, Chandana, Sampreet, Sanjoy, Bibhuti-ranjan, Joyprakash, Alakesh, Ado, and Pranay with whom I have shared some ineffaceable moments of my life at IITG. I am thankful to all my colleagues and friends during my journey as a Ph.D. scholar.

I want to thank the friends from my undergraduate and school days - Hemanta, Mehbub, Chamok, Mandit, Manash, Ratul, Kiran, and Linton for the beautiful memories and cherished moments spent with them. I am happy to mention my childhood friends - Alakesh and Bandana for their encouragement and support.

Finally yet importantly, I would like to thank Almighty God and my family - Ma, Baba, Kondai, Pehi, Dangarbaba, and my brothers and sisters for their unconditional love, support, caring, warmth and profound encouragement all these years. They never doubted my intentions and whole-heartedly supported me in all my endeavours. I fall short of words to express my gratitude to them.

# Certificate

This is to certify that this thesis entitled, **“Algorithms for some Steiner tree problems on Graphs”**, being submitted by **Parikshit Saikia**, to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a bonafide work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulation of the institute. To the best of my knowledge, it has not been submitted elsewhere for the award of the degree.

Date:
Place: Guwahati

..............................

**Dr. Sushanta Karmakar**
Associate Professor
Department of Computer Science and Engineering
IIT Guwahati

# Abstract

The *Steiner tree* (ST) problem is one of the classical problems in combinatorial optimization. Given a connected undirected graph with non-negative edge weights and a subset of terminal nodes, the goal of the ST problem is to find a minimum-cost tree spanning the terminals. It finds applications in network design, content distribution networks, phylogenetic tree reconstruction, and many more. In this thesis we study the ST problem in the distributed setting. In the first contribution we present a deterministic distributed algorithm for the ST problem (DST algorithm) in the CONGEST model that guarantees an approximation factor of $2(1 - 1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST. It has a round complexity of $O(S + \sqrt{n} \log^* n)$ and a message complexity of $O(mS + n^{3/2})$ for a graph of $n$ nodes and $m$ edges, where $S$ is the *shortest path diameter* of the graph. The DST algorithm improves the round complexity of the best distributed ST algorithm known so far, which is $\tilde{O}(S + \sqrt{\min\{St, n\}})$ [90], where $t$ is the number of terminal nodes. Note here that $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in $n$.

The ST problem is a generalized version of the minimum spanning tree (MST) problem. Until 2016, algorithms were known for the MST problem which are either *time-optimal* [38, 87] or *message-optimal* [5, 50], but not both simultaneously. Recently a few *singularly-optimal* distributed algorithms have been proposed for the MST problem [41, 62, 110] beating the long standing time-message trade-off. However, such a study has not been carried out for the ST problem in the distributed setting. In the second contribution of the thesis we investigate the round-message trade-off in distributed ST construction. We modify the DST algorithm and show that a $2(1 - 1/\ell)$-approximate ST can be deterministically computed using $\tilde{O}(S + \sqrt{n})$ rounds and $\tilde{O}(mS)$ messages in the CONGEST model. This algorithm improves the message complexity of the DST algorithm by dropping the additive term of $O(n^{3/2})$ at the expense of a logarithmic multiplicative factor in the round complexity. In case of $S = O(\log n)$, the round and message complexities of the modified DST algorithm are $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively, which almost coincide with the best known singular-optimality results of distributed MST construction in the CONGEST model due to [41, 62, 110] and the approximation ratio of the resultant ST is $2(1 - 1/\ell)$.

There has been a lot of progress in solving various problems in the CONGESTED CLIQUE model of distributed computing including MST, facility location, shortest paths and distances, sorting, routing etc. However, to the best of our knowledge, such a study has

not been carried out for the ST problem till date. In this thesis we study the ST problem in the CONGESTED CLIQUE model and propose two deterministic distributed algorithms: STCCM-A and STCCM-B. The first one computes an ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. The second one computes an ST using $O(S + \log\log n)$ rounds and $O(Sm + n^2)$ messages. Both the algorithms achieve an approximation ratio of $2(1 - 1/\ell)$. For graphs with $S = \omega(n^{1/3} \log n)$, the first algorithm performs better than the second one in terms of the round complexity. On the other hand, for graphs with $S = \tilde{o}(n^{1/3})$, the second algorithm outperforms the first one in terms of the round complexity.

In this thesis we also study a generalized version of the ST problem called *prize-collecting Steiner tree* (PCST). Given a connected undirected graph with non-negative edge weights and a non-negative prize value for each node, the goal of the PCST problem is to find a tree such that the sum of edge weights in the tree plus the prizes of nodes that are not in the tree is minimized. Problems such as MST, ST, Steiner forest, etc. which are related to the PCST, have been widely studied in the distributed setting. However, PCST has seen very little progress in the distributed setting (the only attempt seems to be a manuscript [122]), despite the potential applicability of the problem. In particular, distributed algorithms for PCST would be useful in distributed *ad hoc* (wireless) networks, such as MANETs and sensor networks, where typically nodes have small memory and very limited knowledge of the network.

We present two deterministic distributed algorithms for the PCST problem in the CONGEST model: D-PCST and modified D-PCST. Both the algorithms are based on the primal-dual technique, preserve the dual constraints in a distributed manner, and achieve an approximation factor of $\left(2 - \frac{1}{n-1}\right)$. The D-PCST algorithm computes a PCST using $O(n^2)$ rounds and $O(mn)$ messages. The modified D-PCST algorithm computes a PCST using $O(Dn)$ rounds and $O(mn)$ messages, where $D$ is the *unweighted diameter* of the graph. For graphs with constant or small unweighted diameter ($D = o(n)$), the modified D-PCST algorithm performs better than the original D-PCST algorithm in terms of the round complexity. Both the algorithms require $O(\Delta \log n)$ bits of memory in each node, where $\Delta$ is the maximum degree of a node in the graph.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

"You have to dream before your dreams can come true."
A.P.J. Abdul Kalam (1931 - 2015)
Indian scientist and leader

Network optimization problems are prevalent in computer networks, distributed systems, and operations research. There are generally two kinds of network optimization problems in the distributed setting: *local* and *global*. Graph coloring [9, 85], maximal independent set [55, 123] etc. are a few examples of the *local* variety. A network optimization problem is considered as a *global* problem if solving it necessitates the collaborative computational effort of all the nodes, and information must travel to the farthest nodes in the network. In the last few decades solving network optimization problems has received remarkable attention in the CONGEST model of distributed computing in which the size of each message is restricted to $O(\log n)$ bits, where $n$ is the number of nodes in the network. A few global problems such as breadth first search (BFS) tree, spanning tree etc. require $\Omega(D)$ time (assuming that exchanging a message over a single edge costs one unit of time), where $D$ is the *unweighted diameter* of the network whereas for some other problems like finding the diameter [47], all-pairs shortest paths [40, 69] etc. it is even hard to achieve round complexities near linear in $n$. There is a significant amount of work for many other global problems such as minimum spanning tree (MST) [31, 83, 87], Steiner forest [76, 90] etc. in the CONGEST model of distributed computing.

In this thesis we focus on solving a global optimization problem called *Steiner tree* in the distributed setting. The problem is named in the honor of a famous Swiss mathematician Jacob Steiner (1796-1863), who solved and popularized the problem of joining three villages by a system of roads having minimum total length [61].

## 1.1 Steiner tree

The Steiner tree problem is a classical and fundamental problem in combinatorial optimization and is defined as follows.

**Definition 1.1.1** (Steiner tree (ST) problem). *Given a connected undirected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}^+$, and a set of vertices $Z \subseteq V$, known as the set of terminals, the goal of the ST problem is to find a tree $T' = (V', E')$ such that $\sum_{e \in E'} w_e$ is minimized subject to the conditions that $Z \subseteq V' \subseteq V$ and $E' \subseteq E$.*

The set $V \setminus Z$ is known as the set of *non-terminals* or *Steiner* nodes. Note that if $|Z| = 2$ then the ST problem reduces to the problem of finding *shortest-path* between two distinct nodes in the network. On the other hand if $|Z| = |V|$ then the ST problem becomes the MST problem. Specifically the ST problem is a generalized version of the MST problem. It is known that both MST and shortest path problems can be solved in polynomial time whereas the ST problem is one of the original 21 problems proved NP-complete by Karp [74] (in the centralized setting).

**Example 1.1.** An illustrative example of the ST problem on an arbitrary graph is given in Figure 1.1. The input graph is given in Figure 1.1(a), where $Z = \{A, C, E\}$ is the terminal set. There exist many STs (Steiner trees) with the given terminal set $Z$. Some of them are $T_1 = (\{A, B, C, E\}, \{(A, E), (E, B), (B, C)\})$ as in shown Figure 1.1(b), $T_2 = (\{A, B, C, D, E\}, \{(A, D), (D, E), (E, B), (B, C)\})$ as shown in Figure 1.1(c), and $T_3 = (\{A, B, C, E\}, \{(A, B), (B, C), (B, E)\})$ as shown in Figure 1.1(d). The costs of $T_1$, $T_2$, and $T_3$ are 8, 7, and 9 respectively. Among all the possible STs, $T_2$ is the ST with the minimum edge weights (optimal ST). Note that $T_2$ includes the set $\{B, D\}$ as the set of Steiner nodes.

There are many variations of the ST problem such as directed Steiner tree, metric Steiner tree, euclidean Steiner tree, rectilinear Steiner tree, and so on. Hauptmann and Karpinski

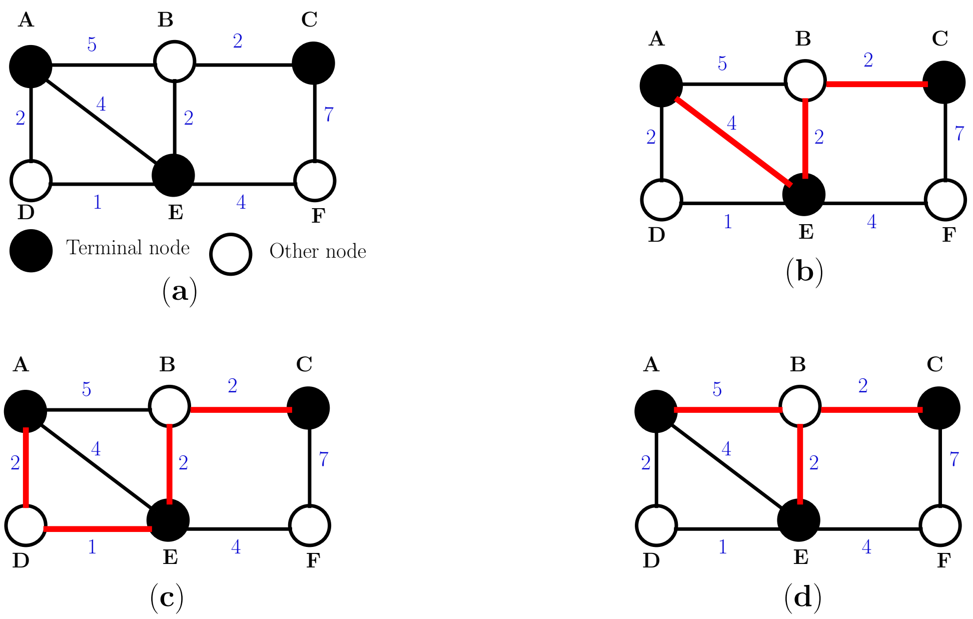


**Figure 1.1:** An illustrating example of the ST problem. (a) a weighted graph $G$ with a terminal set $Z = \{A, C, E\}$. (b), (c), and (d) are some feasible solutions among which (c) is the optimal solution.

[64] provide a website with continuously updated state of the art results for many variants of the problem. The ST problem appears as a subproblem or as a special case of many other problems in network design such as Steiner forest, prize-collecting Steiner tree (PCST) etc. In this thesis we also study the PCST problem, which is defined as follows.

**Definition 1.1.2** (PCST problem)**.** *Given a connected undirected weighted graph* $G = (V, E, p, w)$ *where* $V$ *is the set of vertices,* $E$ *is the set of edges,* $p : V \to \mathbb{R}^+$ *is a vertex prize function and* $w : E \to \mathbb{R}^+$ *is an edge weight function, the goal is to find a tree* $T = (V', E')$*, where* $V' \subseteq V$ *and* $E' \subseteq E$ *that minimizes the following quantity:*

$$W(T) = \sum_{e \in E'} w_e + \sum_{v \in V \setminus V'} p_v$$

This minimization problem is known as the unrooted PCST problem. A feasible solution to the PCST problem partitions the set of nodes into two parts, namely *Steiner* and *Penalty*. A node is in the Steiner part if it is covered by $T$ (i.e. it belongs to $V'$), otherwise it is in the Penalty part. Note that the ST problem is in fact a special case of PCST problem, where we set the prize of terminals to $\infty$ and the prize of all other nodes to 0; therefore PCST is NP-hard and at least as hard to approximate as ST.

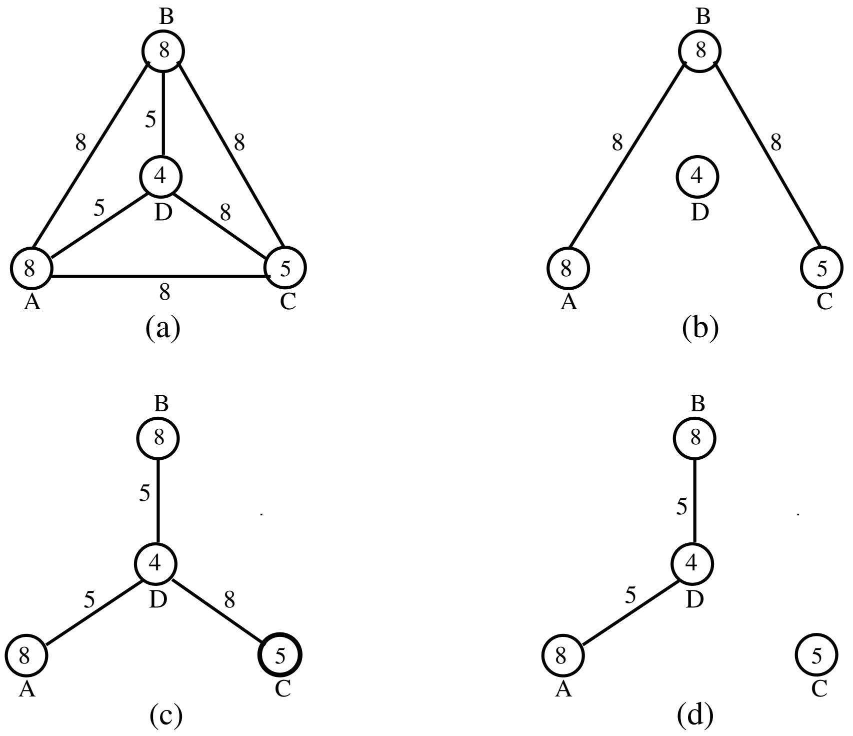


**Figure 1.2:** An illustrating example of the PCST problem. (a) a weighted graph $G$ in which each node has a prize value that is shown inside the node. (b), (c), and (d) are some feasible PCSTs of $G$ among which (d) is the optimal one.

**Example 1.2.** An illustrative example of the PCST problem is shown in Figure 1.2. An arbitrary undirected weighted graph $G$ is given in Figure 1.2(a). Each node in $G$ has a prize value which is shown inside the node. For example node $A$ has a prize value of 8. There exist many feasible PCSTs of $G$. Let $(Steiner, Penalty)$ represent a PCST of $G$. Some of the PCSTs are $T_1 = \big(\{(A,B),(B,C)\},\{D\}\big)$ as in shown Figure 1.2(b), $T_2 = \big(\{(A,D),(B,D),(C,D)\},\phi\big)$ as shown in Figure 1.2(c), and $T_3 = \big(\{(A,D),(B,D)\},\{C\}\big)$ as shown in Figure 1.2(d). The costs of $T_1$, $T_2$, and $T_3$ are 20, 18, and 15 respectively. Among all the possible PCSTs, $T_3$ has the minimum cost.

## 1.2 Applications

The ST problem finds applications in numerous areas such as Very-Large-Scale-Integration (VLSI) layout design [88], communication networks [72], transportation networks [99], content distribution (video on demand, streaming multicast) networks [8], phylogenetic tree reconstruction in computational biology [13] etc. Below we give a brief description of some of the applications.

**VLSI layout design:** VLSI design is a process of placing thousands of transistors on a single chip to create an integrated circuit. From small electronic devices like pen-drive, cell phone, smart phone, camera to the large ones like super computers have thousands of functionalities due to VLSI technology. Rapid growth of functionalities and memory elements inside a single chip demands efficient algorithms and routing policies to route information from one element to another. Also good positioning of elements are essential to reduce power consumption, wire components, electronic interference, and heating of the chip. All these benefits make the chip smaller in size and lower the production costs. The optimization criteria here is to minimize the wire length, and thereby to reduce the total circuit area and the amount of production cost. In this case ST can be used to model the problem.

**Communication networks:** The ST problem is widely used in modeling communication networks. Some of the applications are:

- *Multicast routing*: In a multicast tree, a source and a set of given nodes termed as destinations are connected with minimum link distances. Information is distributed efficiently among the group of collaborating nodes (tree nodes). For example, in teleconferencing the data from one user is to be sent to a selected number of other users. The minimum connected tree can be constructed by forming an ST among the collaborating nodes.

- *Ad-hoc wireless networks*: These networks are dynamic in nature where topology changes over time. Multi-hop communication, limited resources (bandwidth, CPU, battery) and limited security are some other properties of such networks. In such networks each node participates in routing by forwarding data for other nodes. A message sent by a node can be received by all of its neighbours. The decision about which node does the actual forwarding is determined dynamically, based on the network connectivity. Ad-hoc networks are used in areas where network topology is dynamic, rapid network reconfiguration is needed and wired network is not available or difficult to setup. For example military battlefield, disaster management, mobile conferencing etc. are a few areas of applications of ad-hoc network. In all these applications, an ST topology can be constructed over the ad-hoc network, making the information sending and receiving faster with less number of messages.

**Computational biology:** A phylogenetic tree depicts the evolutionary relationships among a set of genes, species etc. that are believed to have a common ancestor. The reconstruction of such a tree is one of the fundamental problems in computational biology. In such a tree leaves represent species, the length of an edge represents the estimated evolution time, and each internal node represents an event for which corresponding ancestor has to evolve to two or more different child species. Forming an ST among these nodes gives the best evolutionary tree to predict the evolution chain of the related species. The detailed description on the evolutionary tree construction can be found in [45, 97].

Similarly the PCST problem has a number of practical applications. Real-world problems in network design (e.g. optical fiber networks), content distribution networks (video on demand, streaming multicast) [8], protein-protein interaction networks [33], leakage detection systems [117], phylogenetic tree reconstruction [97], to name but a few, can be modelled as cases of the PCST problem. In general, PCST applies in situations where various demand points (nodes) need to form a structure with minimum total connection cost. Each demand point has some non-negative prize associated with it. If some of the demand points are too expensive to connect then it might be better not to include them in the structure and instead lose their prize—or, equivalently, pay a penalty, equal or proportional to their prize value.

## 1.3 Motivation

The LOCAL and the CONGEST are two fundamental models in distributed computing [114].[1] The LOCAL model mainly focuses on *locality* and ignores the *congestion* by allowing messages of unlimited sizes to be transmitted [93, 115]. In distributed computing locality means to what extent a global solution to a computational problem can be obtained from locally available data [93]. The main issue of locality is that nodes (processors) are restricted in collecting data from others which are at a distance of $x$ hops in $x$ time units. The CONGEST model on the other hand equally considers both congestion (by bounding the transmitted message size) and locality.

[1]In this thesis, unless it is specified, CONGEST model means the synchronous CONGEST model.

Elkin [39] showed that approximating MST within any constant factor on graphs of small unweighted diameter ($D = O(\sqrt{n})$) requires $\Omega(\sqrt{n/B})$ rounds (assuming $B$ bits can be sent through each edge in each round).[1] Das Sarma et al. [31] achieved an unconditional lower bound on round complexity of the MST problem and showed that approximating MST within any constant factor requires $\Omega(D + \sqrt{n/(B \log n)})$ rounds. The deterministic lower bounds on round and message complexities of the MST problem in the CONGEST model are $\Omega(D + \sqrt{n}/\log n)$ [115] and $\Omega(m)$ [86] respectively.[2] Since the ST problem is a generalization of the MST problem, the lower bounds on the round and message complexities of the MST problem also apply for the ST problem in the CONGEST model of distributed computing. It is therefore highly desirable to obtain a round or message optimal distributed algorithm which computes a near optimal ST. The best round complexity known so far for solving the ST problem in the distributed setting (in the CONGEST model) is due to Lenzen and Patt-Shamir [90] which takes $\tilde{O}(S + \sqrt{\min\{St, n\}})$ rounds and the result is optimal upto a factor of $2 + o(1)$. Here $S$ and $t$ are the *shortest path diameter* (the definition is deferred to Chapter 3, Section 3.1) and the number of terminals respectively in the given input graph. On the other hand the best deterministic round complexity for solving the MST problem in the CONGEST model is $O(D + \sqrt{n} \log^* n)$ due to Garay, Kutten, and Peleg (GKP algorithm) [51, 87]. Note that there is quite a gap in the round complexity between the best known MST algorithm [51,87] and the best known ST algorithm (both in the CONGEST model). Nevertheless the following interesting question is still open:

**Question 1.** *What is the best approximation factor that can be achieved in solving the ST problem while incurring a round complexity cost that is close to that of the best known MST algorithm?*

Until 2016, algorithms were known for the MST problem which are either *time-optimal* [38, 87] or *message-optimal* [5, 50], but not both simultaneously. Recently a few *singularly-optimal* distributed algorithms have been proposed for the MST problem [41,62,110] beating the long standing time-message trade-off. Note that a distributed algorithm is said to be

[1]In literature, round complexity is typically called time complexity. Here we use time and round interchangeably.

[2]Kutten et al. [86] established that $\Omega(m)$ is the message lower bound for leader election in the $KT_0$ model (i.e. **K**nowledge **T**ill radius **0**) which holds for both the deterministic as well as randomized (Monte Carlo) algorithms even if the network parameters $D$, $n$, and $m$ are known, and all nodes wake up simultaneously. Since a distributed MST algorithm can be used to elect a leader, the above message lower bound in the $KT_0$ model also applies to the distributed MST construction.

*singularly-optimal* if it achieves the optimal round and message complexities simultaneously. It is known that the round and message complexities of the singularly-optimal MST algorithms are $\tilde{O}(D+\sqrt{n})$ and $\tilde{O}(m)$ (here $m$ is the number edges in the graph) respectively and both are optimal upto a polylogarithmic factors in $n$ [41, 62, 110]. However, such a study has not been carried out for the ST problem in the distributed setting. Therefore another intriguing question is:

**Question 2.** *Can we achieve a distributed algorithm for the ST problem in the CONGEST model whose round and message complexities coincide with that of the singular optimality results of the MST algorithms [41, 62, 110] while maintaining an approximation factor of at most* $2$*?*

In distributed computing one of the special cases of the CONGEST model is the CONGESTED CLIQUE model (CCM) that was first introduced by Lotker et al. [96]. This model takes locality out of the picture and solely focuses on congestion. Specifically, in CCM, nodes can communicate with each other via an underlying communication network, which is a *clique*. Communication happens in synchronous rounds and a pair of nodes can exchange $b$ bits in a round. Following the convention it is assumed that $b = O(\log n)$. Since, in CCM the hop diameter is *one*, nodes can directly communicate with each other and in each round all nodes can together exchange $O(n^2 \log n)$ bits.

The main motivation behind the study of the CCM is to understand the role of congestion in distributed computing. There has been a lot of progress in solving various problems in the CCM including MST [56, 67, 73, 96, 107], facility location [15, 53], shortest paths and distances [24, 68, 104], subgraph detection [36], triangle finding [34, 36], sorting [89, 112], routing [89], and ruling sets [15, 67]. Despite the significant progress in solving various problems in the CCM, to the best of our knowledge, such a study has not been carried out for the ST problem. Therefore an intriguing question is:

**Question 3.** *What is the best round complexity that can be achieved in solving the ST problem in the CCM while maintaining an approximation factor of at most* $2$*?*

Furthermore, problems such as MST, ST, Steiner forest, etc. which are related to the PCST, have been widely studied in the distributed setting. However, to the best of our knowledge, PCST has seen very little progress in distributed setting (the only attempt seems to be a manuscript [122]), despite the potential applicability of the problem. In particular, distributed algorithms for PCST would be useful in distributed *ad hoc* (wireless) networks, such as MANETs and sensor networks, where typically nodes have small memory and very limited knowledge of the network. Therefore the following is an interesting question:

**Question 4.** *Can we design a non-trivial distributed algorithm for the PCST problem while maintaining a constant approximation factor?*

We know that a rooted BFS tree can be computed using $O(D)$ rounds and $O(m)$ messages in both LOCAL and CONGEST model [114].[1] In LOCAL model one can trivially solve the PCST problem by simply collecting the entire network topology at the root of the BFS tree, which takes $O(D)$ rounds, centrally compute the PCST using the best known centralized PCST algorithm (the algorithm proposed by Aarcher et al. [4]), and finally distribute the solution in the network using $O(D)$ rounds. However, in this trivial approach, the memory requirement per node is $O(m \log n)$ bits since information concerning all $m$ edges needs to be stored at the root; note also that a large amount of information must be stored at nodes close to the root. Similarly in the CONGEST model one can solve the PCST problem by simply using the gather and solve method leveraging on a rooted BFS tree using $O(m)$ rounds and the memory requirement per node is again $O(m \log n)$ bits. This implies that the trivial approaches do not work well in resource-constrained networks, specially in networks where nodes have limited memory space. Therefore, a natural question arises–

**Question 5.** *Can we improve the per node memory requirement for distributed PCST computation with a guaranteed constant approximation factor?*

## 1.4 Objectives

Recently Bacrach et al. [6] came with a lower bound round complexity for exact ST computation in the CONGEST model, which is $\Omega(n^2/log^2 n)$. On the other hand approximating (for any constant factor $\alpha > 0$) MST requires $\Omega(D + \sqrt{n}/\log n)$ rounds in the CONGEST model due to Das Sarma et al. [31]. It is also known that the lower bound message complexity for MST construction is $\Omega(m)$ [86]. Since the ST problem is a generalized version of

[1]The LOCAL model allows messages of unlimited size, whereas the CONGEST model allows messages of size $O(\log n)$ bits only. In both LOCAL and CONGEST model it is assumed that the computation time at a node is negligible in comparison to the message communication delay between any two distinct nodes in the network.

the MST problem, we believe that in the approximation sense the lower bound results for the MST construction also hold for the ST construction. Therefore the first two objectives of our work are as follows.

> **Objective I.** We will design a deterministic distributed approximation algorithm in the CONGEST model that computes an ST of a given connected undirected weighted graph with the round complexity that is close to the lower bound round complexity for MST construction subject to the condition that the resultant ST maintains an approximation factor of at most 2.

> **Objective II.** We will investigate the round-message trade-off in distributed ST construction in the CONGEST model.

Regarding the ST problem in the CCM, to the best of our knowledge, no result is known till date. However, in CCM one can trivially compute an ST that has an approximation factor of at most 2 using $O(n)$ rounds as follows. One can collect the entire topology of the input graph in a special node $r$, which takes $O(n)$ rounds. Then we can compute an ST (locally) by applying one of the best known centralized ST algorithms [20, 84, 130] whose approximation factor is at most 2, and finally inform each of the nodes involved with the resultant ST. Note that the resultant ST can consist of at most $n-1$ edge information which can be decomposed into $O(n)$ messages. Therefore $r$ can perform the final step using $O(1)$ rounds by sending each edge of the resultant ST to a different intermediate node, which will eventually send to the destined node. Now the third objective of our work is as follows.

> **Objective III.** We will design a non-trivial deterministic distributed approximation algorithm for the ST problem in the CCM which guarantees the following properties.
>
> - Asymptotically better round complexity than the trivial one.
> - No need to collect the entire topology at a special node.
> - The approximation factor of the resultant ST is at most 2.

The fourth objective of our work is related to the PCST problem. Note that in literature there exist a few primal-dual based distributed approximation algorithms for problems like facility location [102, 109], vertex cover [59, 60], set cover [42] etc. One common aspect of these algorithms is that they are derived in a systematic fashion from their respective sequential primal-dual counterparts. Notably, while there exist many sequential primal-dual based algorithms for PCST [1,4,44,58,72], to the best of our knowledge, such an attempt is not known in the distributed setting (the only known manuscript is due to Rossetti [122]). Therefore the fourth objective of our work is as follows.

> **Objective IV.** We will design a primal-dual based deterministic distributed algorithm for the PCST problem while maintaining an approximation factor of at most 2.

## 1.5 Contributions

The contributions of the thesis are summarized in the following subsections.

### 1.5.1 Improved distributed approximation for ST in the CONGEST model

#### 1.5.1.1 DST algorithm: A $2(1-1/\ell)$-factor distributed ST algorithm using $O(S+\sqrt{n}\log^* n)$ rounds

In this contribution we present a deterministic distributed algorithm for the ST problem (DST algorithm) in the CONGEST model [114] that guarantees an approximation factor of $2(1-1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST. The DST algorithm has a round complexity of $O(S+\sqrt{n}\log^* n)$, which is better than the round complexity of the best distributed ST algorithm known so far, which is $\tilde{O}(S+\sqrt{\min\{St, n\}})$ [90]. It also significantly reduces the round complexity gap between solving the ST problem and the MST problem in the CONGEST model. The message complexity of the DST algorithm is $O(mS+n^{3/2})$. We also propose a deterministic distributed *shortest path forest* (SPF) algorithm in the CONGEST model that computes a SPF using $O(S)$ rounds and $O(Sm)$ messages. The SPF algorithm is used as a subroutine in the DST algorithm. The first contribution of the thesis is stated in the following theorem.

**Theorem 1.5.1.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes an ST using $O(S+\sqrt{n}\log^* n)$ rounds in the CONGEST model with an approximation factor of $2(1-1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST.*

#### 1.5.1.2 Round-Message Trade-off in distributed ST construction

This contribution is related to the investigation of the round-message trade-off in computing ST in the distributed setting. We modify the DST algorithm which helps us achieve the round and message complexities of $\tilde{O}(S+\sqrt{n})$ and $\tilde{O}(mS)$ respectively. The modified DST algorithm has the same approximation factor as that of the original DST algorithm. The polylogarithmic factors involved with the round and message complexities of the modified DST algorithm are $O(\log n)$ and $O(\log^2 n)$ respectively. The modified DST algorithm improves the message complexity of the original DST algorithm by dropping the additive term of $O(n^{3/2})$ at the expense of a logarithmic multiplicative factor in the round complexity. Specifically we prove the following theorem.

**Theorem 1.5.2.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes an ST using $\tilde{O}(S + \sqrt{n})$ rounds and $\tilde{O}(Sm)$ messages in the CONGEST model with an approximation factor of $2(1 - 1/\ell)$.*

As a by-product of the above theorem, for networks with constant or sufficiently small shortest path diameter ($S = O(\log n)$) the following corollary holds.

**Corollary 1.5.2.1.** *If $S = O(\log n)$ then a $2(1 - 1/\ell)$-approximate ST can be deterministically computed with the round and message complexities of $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively in the CONGEST model.*

Note that in case of $S = O(\log n)$, the round and message complexities of the modified DST algorithm are $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively, which almost coincide with the best known singular-optimality results of distributed MST construction in the CONGEST model due to [41, 62, 110] and the approximation factor of the resultant ST is at most $2(1 - 1/\ell)$.

### 1.5.2 Distributed approximation algorithms for ST in the CCM

In this contribution we present two non-trivial deterministic distributed approximation algorithms for the ST problem in the CCM: STCCM-A and STCCM-B. Both the algorithms perform better than the trivial one in two aspects: (i) asymptotically better round complexity and (ii) no need to collect the entire topology at a special node.

#### 1.5.2.1 STCCM-A: A $2(1-1/\ell)$-factor ST algorithm using $\tilde{O}(n^{1/3})$ rounds in the CCM

The STCCM-A algorithm computes a $2(1-1/\ell)$-approximate ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. We also propose a deterministic distributed SPF algorithm in the CCM that computes a SPF using $O(n^{1/3}\log n)$ rounds and $O(n^{7/3}\log n)$ messages. The SPF algorithm is used as a subroutine in the STCCM-A algorithm. Specifically we prove the following theorem.

**Theorem 1.5.3.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes a $2(1-1/\ell)$-approximate ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages in the CCM.*

#### 1.5.2.2 STCCM-B: A $2(1-1/\ell)$-factor ST algorithm using $O(S+\log\log n)$ rounds in the CCM

We also propose STCCM-B, another deterministic distributed approximation algorithm for the ST problem in the CCM, that computes a $2(1-1/\ell)$-approximate ST using $O(S + \log\log n)$ rounds and $O(Sm+n^2)$ messages. Furthermore, we propose another SPF algorithm in the CCM that computes a SPF using $O(S)$ rounds and $O(Sm)$ messages. This version of the SPF algorithm is used as a subroutine in the STCCM-B algorithm. The above results are summarized in the following theorem.

**Theorem 1.5.4.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes a $2(1-1/\ell)$-approximate ST using $O(S + \log\log n)$ rounds and $O(Sm + n^2)$ messages in the CCM.*

As a by-product of the above theorem, for networks with constant or sufficiently small shortest path diameter (where $S = O(\log\log n)$) the following corollary holds.

**Corollary 1.5.4.1.** *If $S = O(\log\log n)$ then a $2(1-1/\ell)$-approximate ST can be deterministically computed using $O(\log\log n)$ rounds and $\tilde{O}(n^2)$ messages in the CCM.*

Note that in case of $S = O(\log\log n)$, the round and message complexities of the STCCM-B algorithm are $O(\log\log n)$ and $\tilde{O}(n^2)$ respectively, which almost coincide with the best known deterministic results for MST construction in the CCM due to Lotker et al. [96] and the approximation factor of the resultant ST is at most $2(1-1/\ell)$ of the optimal.

### 1.5.3 Primal-dual based distributed approximation algorithm for PCST

In this contribution first we present a deterministic distributed algorithm for the PCST (D-PCST algorithm) problem that computes a $\left(2 - \frac{1}{n-1}\right)$-approximate PCST for a given connected undirected graph of $n$ nodes with non-negative edge weights and a non-negative prize value for each node. The D-PCST algorithm is inspired by a sequential algorithm proposed by Goemans and Williamson (GW-algorithm [58]), which is one of the most elegant algorithms for the PCST problem, providing a constant approximation ratio. For an input graph $G = (V, E)$, the round and message complexities of the D-PCST algorithm in the CONGEST model are $O(n^2)$ and $O(mn)$ respectively, where $n = |V|$ and $m = |E|$. Furthermore, we modify the D-PCST algorithm and show that a $(2 - \frac{1}{n-1})$-approximate PCST can be deterministically computed using $O(Dn)$ rounds and $O(mn)$ messages in the CONGEST model. For networks with small unweighted diameter ($D = o(n)$), the modified D-PCST algorithm performs better than the original one in terms of the round complexity. The overall performance of the modified D-PCST algorithm is summarized in the following theorem.

**Theorem 1.5.5.** *Given a connected weighted graph $G = (V, E, p, w)$ where $V$ is the set of vertices, $E$ is the set of edges, $p : V \rightarrow \mathbb{R}^+$ is a vertex prize function and $w : E \rightarrow \mathbb{R}^+$ is an edge weight function, a $(2 - \frac{1}{n-1})$-approximate PCST can be deterministically computed using $O(Dn)$ rounds and $O(mn)$ messages in the CONGEST model of distributed computing, where $n = |V|$, $m = |E|$, and $D$ is the unweighted diameter of $G$.*

Our PCST algorithms are based on the primal-dual method and uses a technique of preserving dual constraints in a distributed manner, without relying on knowledge of the global structure of the network. Both the algorithms require $O(\Delta \log n)$ bits of memory in each node, where $\Delta$ is the maximum degree of a node in the graph. In contrast, none of the earlier primal-dual based distributed approximation algorithms [42, 59, 60, 102, 109] discuss about the memory efficiency of their proposed solutions. Indeed, one can design a trivial, essentially centralized, distributed algorithm for the PCST problem by 'black-box' use of the GW-algorithm. In such a case there should be a specified node (leader) to gather the entire graph information and compute the PCST solution in a centralized manner. Assuming polynomially bounded edge weights and node prizes, the leader node would need to store

**Table 1.1:** *Summary of contributions of the thesis. Here CM = CONGEST model, CCM = CONGESTED CLIQUE model, DT = deterministic, $n = |V|$, $m = |E|$, $\ell$ is the number of leaf nodes in the optimal ST, $D$ and $S$ are the unweighted diameter and the shortest path diameter of a given connected undirected weighted graph $G$ respectively.*

| **Problems** | **Type** | **Model** | **Round complexity** | **Message complexity** | **Approximation** |
|---|---|---|---|---|---|
| ST | DT | CM | $O(S + \sqrt{n}\log^* n)$ | $O(mS + n^{3/2})$ | $2(1 - 1/\ell)$ |
| | | | $\tilde{O}(S + \sqrt{n})$ | $\tilde{O}(mS)$ | $2(1 - 1/\ell)$ |
| | | CCM | $\tilde{O}(n^{1/3})$ | $\tilde{O}(n^{7/3})$ | $2(1 - 1/\ell)$ |
| | | | $O(S + \log\log n)$ | $O(Sm + n^2)$ | $2(1 - 1/\ell)$ |
| PCST | | CM | $O(n^2)$ | $O(mn)$ | $2 - \frac{1}{n-1}$ |
| | | | $O(Dn)$ | $O(mn)$ | $2 - \frac{1}{n-1}$ |

$O(m \log n)$ bits in its memory. In contrast, in our approach only local information is stored in each node.

The main contributions of the thesis are summarized in Table 1.1.

## Road map

The rest of the thesis is organized as follows.

**Chapter 2.** This chapter provides a survey of the previous related works that are needed to get the idea of the state-of-art works.

**Chapter 3.** In this chapter we present two deterministic distributed algorithms for the ST problem in the CONGEST model. The proof of correctness of both the algorithms are given. The first one computes a $2(1 - 1/\ell)$-approximate ST using $O(S + \sqrt{n}\log^* n)$ rounds and $O(mS + n^{3/2})$ messages. The second one is a modified version of the first one, which computes a $2(1-1/\ell)$-approximate ST using $\tilde{O}(S+\sqrt{n})$ rounds and $\tilde{O}(mS)$ messages.

**Chapter 4.** This chapter presents two deterministic distributed algorithms for the ST problem in the CCM. Both the algorithms achieve an approximation factor of $2(1-1/\ell)$. Proof of correctness are given for both the algorithms. The first one computes an ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. The second one does the same using $O(S+\log\log n)$ rounds and $O(Sm+n^2)$ messages.

**Chapter 5.** This chapter presents two deterministic distributed algorithms for the PCST problem in the CONGEST model. The first one computes a $\left(2-\frac{1}{n-1}\right)$-approximate PCST using $O(n^2)$ rounds and $O(mn)$ messages. The second one computes a $\left(2-\frac{1}{n-1}\right)$-approximate PCST using $O(Dn)$ rounds and $O(mn)$ messages. The proof of correctness of both the algorithms are also given.

**Chapter 6.** This is the final chapter. It contains the summary of work done, the highlight of the contributions, and the directions for possible future work.

# 2
# Related work

In this chapter we present the state of art results for both the Steiner tree (ST) and the prize-collecting Steiner tree (PCST) in both sequential and distributed settings.

## 2.1 Sequential ST algorithms

### 2.1.1 Exact algorithms

An *exact algorithm* is used to obtain the exact solution of a problem. It is believed that solving an NP-hard problem exactly (optimally) requires exponential time unless $P = NP$. However, some algorithms with exponential running time have been considered to be acceptable if the size of the problem is not too large. Dreyfus and Wagner [35] proposed the first exact sequential algorithm for the ST problem with the running time of $O(3^t n + 2^t n^2 + n^2 \log n + mn)$, where $m$, $n$, and $t$ are the number of edges, the number of nodes, and the number of terminals respectively in the given input graph. Fuchs et al. [49] improved the time complexity to $O(2.684^t \cdot n^{O(1)})$ and Mölle et al. [100] improved the time complexity to $O((2 + \epsilon)^t \cdot n^{f(e^{-1})})$ for a constant factor $\epsilon > 0$. Later on Fuchs et al. [48] improved the exponent to $O((\frac{\epsilon}{-ln\epsilon})^{-c})$ for any constant $c > 1/2$, which has a time complexity of $O(2.5^t n^{14.2})$ or $O(2.1^t n^{57.6})$. By using the subset convolution and MöBius inversion

Bjorklund et al. [17] presented an exact algorithm for computing a minimum ST with the time complexity of $\tilde{O}(2^t n^2 + mn)$, where $\tilde{O}(\cdot)$ notation hides the polylogarthmic factors in $n$. The summary of related work of exact ST algorithms is provided in Table 2.1.

**Table 2.1:** *Summary of related work of exact ST algorithms. Here $n$, $m$, and $t$ are the number of nodes, number of edges, and number of terminals respectively in a given graph.*

| **References** | **Time complexity** |
|---|---|
| Dreyfus and Wagner [35] | $O(3^t n + 2^t n^2 + n^2 \log n + mn)$ |
| Fuchs et al. [49] | $O(2.684^t \cdot n^{O(1)})$ |
| Mölle et al. [100] | $O((2+\epsilon)^t \cdot n^{f(e^{-1})})$ |
| Fuchs et al. [48] | $O(2.5^t n^{14.2})$ or $O(2.1^t n^{57.6})$ |
| Bjorklund et al. [17] | $\tilde{O}(2^t n^2 + mn)$ |

### 2.1.2 Approximation algorithms

As of today, the NP-hard problems can not be solved optimally in polynomial time unless $P = NP$. Since the ST problem is also an NP-hard, this also holds for it. Nevertheless, the ST problem has numerous practical applications. Therefore an efficient algorithm[1] which computes a near-optimal ST is widely accepted. There are numerous ways to compute a near-optimal ST in polynomial time:– approximation, randomization, parameterization, heuristics etc. In this section we mainly focus on the approximation algorithms for the ST problem exist in literature. Note that the goal of an approximation algorithm [126] is to compute a near-optimal solution of an NP-hard problem as fast as possible. An $\alpha$-approximation algorithm is defined as follows.

**Definition 2.1.1.** *An $\alpha$-approximation algorithm for an optimization problem is a polynomial time algorithm such that for any instances of the problem it computes a solution whose value is within a factor of $\alpha$ of the value of an optimal solution.*

To the best of our knowledge, the first sequential approximation algorithm for the ST problem was proposed by Edward Forrest Moore (1925-2003) which was published by Gilbert

[1]Here the efficient algorithm denotes an algorithm which solves a problem in polynomial time.

and Pollak [57] in the year 1968 (by referring Moore). This algorithm, which is based on the MST heuristic, has an approximation ratio of 2. A Prim's based shortest path heuristic (P-SPH) approximate ST algorithm and a Krushkal's based shortest path heuristic (K-SPH) approximate ST algorithm were proposed by Takahashi and Matsuyama [125] in the year 1980 and Kou et al. [84] in the year 1981 respectively. The approximation ratio guaranteed by both the algorithms is 2 and remained the best known approximation ratio until Zelikovsky's 11/6-approximation algorithm [132] proposed in 1993, which became the first algorithm to beat the 2-approximation algorithm for the ST problem. The total running time of this algorithm is $O(nm + |Z|^4)$. Berman and Ramaiyer [14] improved Zelikovsky's 11/6 bound to 16/9 by showing that 3 element subset of Zelikovsky's consideration can be further increased to 4 element subset. Based on the Zelikovsky's central idea, sequence of improved approximation algorithms appeared in literature. The bound of $16/9 = 1.764$ was improved to 1.693 by Zelikovsky in 1996 [133], then to 1.667 by Promel and Steger in 1997 [118], to 1.644 by Karpinski and Zelikovsky in 1997 [75], to 1.598 by Hougardy and Promel in 1999 [70] and to 1.550 (actually $1 + \frac{1}{2}\ln 3 + \epsilon$ for any arbitrary small constant $\epsilon > 0$) by Robins and Zelikovsky in 2000 [121].

The sequence of improved approximation ratios for the ST problem in [14, 70, 75, 118, 121, 132, 133] are mainly based on the following idea. A full Steiner component (or full component for short) of an ST is a subtree whose internal vertices are Steiner vertices, and whose leaves are terminals. The edge set of any ST can be partitioned into full components by splitting the tree at terminals. An $r$-(restricted)-ST is defined to be an ST all of whose full components have at most $r$ terminals. Borchers and Du [18] showed that in order to obtain a good approximation factor, it is sufficient to restrict our attention to $r$-restricted STs. Let $c(opt)$ and $c(opt_r)$ denote the costs of the optimal ST and an optimal $r$-restricted ST respectively. Then the following theorem holds.

**Theorem 2.1.1.** *( [18]) Let $p_r$ be the $r$-Steiner ratio, i.e. the supremum of the ratio $c(opt_r)/c(opt)$. Then $q_r = \frac{(a+1)2^a+b}{a2^a+b} \leq 1 + \frac{1}{[\log_2 r]}$, where $a$ and $b$ are non-negative integers such that $r = 2^a + b$ and $b < 2^a$.*

Till date, the best known (polynomial time) approximation ratio for solving the ST problem in the sequential setting is $\ln(4) + \epsilon \approx 1.386 + \epsilon$, for any constant factor $\epsilon > 0$ due to Byrka et al. [20]. Byrka et al. achieved this result by using the *iterative randomized rounding* technique. It is also known that in general graphs the ST problem can not be solved in polynomial time with an approximation factor $\leq \frac{96}{95}$ [28] unless $P = NP$.

**Table 2.2:** *Summary of related work of sequential ST algorithms for general graphs. P-SPH: Prim's based shortest path heuristic, K-SPH: Krushkal's based SPH, LP: linear programming, $\epsilon > 0$ is any constant, "*" means particular result is not discussed in that work.*

| **Authors** | **Techniques** | **Approx. Ratio** | **Time Complexity** |
|---|---|---|---|
| Gilbert and Pollak [57] | Moore Theorem | 2 | $O(nt^2)$ |
| Takahashi and Matasuyama [125] | P-SPH | 2 | * |
| Kou et al. [84] | K-SPH | 2 | * |
| Goemans and Williamson [58] | primal-dual | 2 | $O(n^2 \log n)$ |
| Jain [71] | Iterative rounding | 2 | $O(n^{10}m^7)$ |
| Zelikovsky [132] | $r$-restricted ST | 1.834 | $O(mn + t^4)$ |
| Berman and Ramaiyer [14] | $r$-restricted ST | 1.734 | $O(n^5)$ |
| Zelikovsky [133] | $r$-restricted ST | 1.694 | $O(n^3)$ |
| Promel and Steger [118] | $r$-restricted ST | 1.67 | $O(n^3)$ |
| Karpinski and Zelikovsky [75] | $r$-restricted ST | 1.644 | $O(n^3)$ |
| Hougardy and Promel [70] | $r$-restricted ST | 1.598 | * |
| Robins and Zelikovsky [121] | $r$-restricted ST | 1.55 | $O(mn^2)$ |
| Byrka et al. [20] | LP and Iterative randomization | $\ln(4) + \epsilon$ | $\tilde{O}(n^6)$ |

One of the known standard approaches to solve the ST problem is the use of proper LP relaxation. A natural formulation of the problem is *undirected cut formulation* [58]. In this formulation, a variable is assigned for each edge of the graph and a constraint is considered for each cut separating the set of terminals. Each constraint helps to pick at least one edge crossing the corresponding cut. Considering the LP relaxation, 2-approximation algorithms can be obtained by using the primal-dual framework developed by Goemans and Williamson [58] or the iterative rounding technique due to Jain [71].

To the best of our knowledge, the best known *integrality gap* of LP relaxation for the ST problem in general graphs is 2. However there exists integrality gap better than 2 for special class of graphs, known as *quasi-bipartite* graphs.[1] For such graphs Rajagopalan and Vazirani [119] gave an upper bound on the integrality gap, which is 3/2. This was improved to 4/3 by Chakrabarty et al. [25]. Robins and Zelikovsky [121] showed that a 1.28-approximate ST can be computed if input graphs are restricted to quasi-bipartite graphs. With a different LP formulation Könemann et al. [82] showed that for $b$-quasi-bipartite

[1]A graph is called *quasi-bipartite* if all of its Steiner nodes form an independent set.

graphs[1] the integrality gap is upper-bounded by $\frac{2b+1}{b+1}$. Moreover, the best known hardness result for the ST problem in this class of graphs is $\frac{128}{127}$ [28]. For planar graphs Borradaile et al. [19] proposed a PTAS for ST problem whose running time is $O(n \log n)$. Recently Byrka et al. [21] presented a PTAS for the ST problem which holds for *map graphs*.[2] A summary of performances of known sequential algorithms in special class of graphs for ST problem is listed in Table 2.3.

**Table 2.3:** *Summary of related work of sequential ST algorithms for special class of graphs.* $\epsilon > 0$ *and "*" means particular result is not discussed in that work.*

| **Authors** | **Graphs** | **Approx. Ratio** | **Time Complexity** |
|---|---|---|---|
| Rajagopalan and Vazirani [119] | Quasi-bipartite | 3/2 | $O(\frac{1}{\epsilon} mn \log n)$ |
| Chakrabarty et al. [25] | Quasi-bipartite | 4/3 | * |
| Robins and Zelikovsky [121] | Quasi-bipartite | 1.28 | $O(mn^2)$ |
| Könemann et al. [82] | b-quasi-bipartite | $\frac{2b+1}{b+1}$ | * |
| Borradaile et al. [19] | Planar | PTAS | $O(n \log n)$ |
| Byrka et al. [21] | Map | PTAS | * |

## 2.2 Distributed ST algorithms

In literature many heuristic algorithms for the ST problem have been proposed in the sequential setting. However only a few of them are suitable to adapt to the distributed setting. This is because in distributed setting nodes have limited knowledge about the network topology. Regarding distributed algorithms for the ST problem, there are quite a few related works in literature which are mainly based on MST heuristic, shortest path heuristic (SPH), and average distance heuristic (ADH).

In MST heuristic, initially an MST is constructed spanning all the given terminals. After that a pruning operation is performed to delete all the subtrees from the MST which contain no terminal node. Basically, there are two types of distributed MST algorithm. One is based

[1] A graph is $b$-quasi-bipartite if on deleting all required vertices (terminal nodes), the largest size of any component is at most $b$.

[2] The map graph has edges whenever regions share at least a single point, whereas the planar graph has edges between two regions if they share a border.

on Prim's MST algorithm, where the tree is initialized with the source node and then grows by successively adding the next closest node to the tree, until all nodes are in the tree. The other type is based on Kruskal's MST algorithm. This type of algorithm initializes each of the nodes as a subtree and joins subtrees pairwise repeatedly until all the nodes are in a single tree.

In SPH based distributed algorithms, it is assumed that each node knows in advance the distances[1] to all other nodes in the network. Basically there are two types of SPH algorithms namely Prim's based SPH (P-SPH) and Krushkal's based SPH (K-SPH). In P-SPH (aka cheapest insertion heuristic) [81] a single tree grows at a time. At each step a least cost path is added from the existing partially built tree to the destination node that has not yet been connected. In K-SPH algorithms [12, 106, 124] on the other hand, initially each multi-cast member node starts out as a fragment. The leader of each fragment attempts to merge with its closest neighbouring fragment in parallel.

A more generalized version of the K-SPH is the ADH [129]. The execution of an ADH based algorithm begins with a set of components, initially each one consists of a terminal node. In each iteration, two components $A$ and $B$ are merged by a path that crosses a node that has a minimum average distance to each of the components $A$ and $B$. The ADH based algorithm terminates with a single tree, spanning all the terminal nodes. The distributed version of the ADH was proposed by Gatani et al. [52] whose approximation factor is 2 and its time and message complexities are $O(tD)$ and $O(m + tn)$ respectively, where $D$ is the unweighted diameter of a graph. Moreover there exist some other algorithms for the ST problem in the distributed setting which are based on the techniques of tree embedding (TE) [76], all pairs shortest paths (APSP) [90] construction etc.

To the best of our knowledge, Chen et al. [27] proposed the first distributed algorithm for the ST problem in the asynchronous CONGEST model and achieved an approximation factor of $2(1 - 1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST. The round and message complexities of this algorithm are $O(n(n - t))$ and $O(m + n(n - t + \log n))$ respectively.

Kompella et al. [81] addressed the ST problem as *constrained ST or delay bounded ST* problem. They proposed two distributed algorithms to compute multi-cast trees, in which a source node and a set of destinations nodes are given as the set of terminals. Such multi-cast

[1]The distance between two vertices in a graph is the length of the shortest path connecting them.

**Table 2.4:** *Summary of related work of the ST problem in distributed setting. Here DT: deterministic, RM: randomized, CM: CONGEST model, LM: LOCAL model, MST: minimum spanning tree, SPH: shortest path heuristic, P-SPH: Prim's based SPH, K-SPH: Krushkal's based SPH, ADH: average distance heuristic, TE: tree embedding, APSP: all pairs shortest paths, "–" indicates that the corresponding result is verified by simulation and "*" indicates that the corresponding result is not discussed in that work.*

| **Authors** | **Model** | **Heuristics** | **Type** | **Time Complexity** | **Mess. Complexity** | **Approx.** |
|---|---|---|---|---|---|---|
| Chen et al. [27] | CM | MST | DT | $O(n(n-t))$ | $O(m + n((n-t) + \log n))$ | $2(1 - 1/\ell)$ |
| Kompella et al. [81] | LM | MST | DT | $O(n^3)$ | $O(n^3)$ | – |
| Bauer and Varma [12] | LM | K-SPH | DT | $O(Dn)$ | $O(nt)$ | – |
| Bauer and Varma [12] | LM | SPH | DT | $O(Dt)$ | $O(nt)$ | – |
| Novak and Rugelj [105] | LM | SPH | DT | $O(Dt)$ | $O(nt)$ | – |
| Singh and Vellanki [124] | LM | K-SPH | DT | $O(Dt)$ | $O(n \log t)$ | $*$ |
| Chalermsook et al. [26] | CM | MST | DT | $O(n \log n)$ | $O(tn^2)$ | 2 |
| Gatani et al. [52] | LM | ADH | DT | $O(Dt)$ | $O(m + nt)$ | – |
| Khan et al. [76] | CM | TE | RM | $O(S \log^2 n)$ | $O(Sn \log n)$ | $O(\log n)$ |
| Lenzen et al. [90] | CM | APSP | DT | $\tilde{O}(S + \sqrt{\min\{St, n\}})$ | $*$ | $2 + o(1)$ |
|  |  |  | RM | $\tilde{O}(D + \sqrt{n})$ | $*$ | $2 + o(1)$ |

trees are useful in multi-casting multimedia information in bounded delay. The basic idea of the two algorithms is same; the only difference is in the edge selection criteria they apply to select edges. Specifically the algorithms are mainly based on the Prim's based shortest path heuristic (P-SPH). Usually in distributed MST algorithms the time delay along the path is not considered. The lack of consideration of time delay may lead to the failure of meeting the delay bound by a grown tree constructed by usual distributed MST algorithms. To mitigate this problem Kompella et al. introduce the time delay along with the edge cost to find more realistic solution for the multi-cast tree construction. The algorithm, which is based on P-SPH, builds multi-cast tree by selecting the cheapest outgoing edge from the subtree constructed so far. This ensures that all destinations are either reachable or have been reached within the bounded delay. The second algorithm focuses on the time delay by considering the residual delay (largest remaining delay on the path) which improves the cost of the computed multi-cast tree. Both the algorithms have the same time and message complexities, which is $O(n^3)$. The approximation ratios of both the algorithms are verified empirically by comparing with the approximation ratio of known centralized algorithms. Specifically it is shown that the approximation ratio is $15 - 30\%$ better than that of the centralized algorithms.

One of the drawbacks suffered by the algorithms proposed by Kompella et al. [81] is that all nodes in the network have to participate in the process of computing the required multi-cast tree. It has been shown in [127] that the quality of the trees suffer as the participation of the number of non multi-cast nodes increases. This may be impractical in large networks with sparse multi-cast groups. Moreover, the theoretical upper bound on *competitiveness* of a pruned MST to that of an optimal ST has been shown to be $y + 1$, where $y$ is the number of non-multi-cast nodes [127]. Here the competitiveness is defined as the ratio of the sum of the edge weights of the tree constructed by the heuristic to that of an optimal tree.

Bauer and Varma [12] proposed two distributed algorithms to construct a multi-cast tree over a set of nodes using the SPH and the K-SPH. In both the algorithms only the multi-cast nodes (terminals) participate in the multi-cast tree construction. The K-SPH algorithm begins in parallel at all the terminals. Subtrees containing at least one terminal are called fragments. Each fragment contains a fragment leader, which coordinates all the activities of the fragment. The identity (ID) of a fragment leader is unique and is known to each of the nodes in the fragment. Initially each fragment consists of one terminal. Fragments grow by merging (pairwise) with other fragments. The algorithm relies on the shortest path information available at each node and the information gathered by the fragment leader. Note here that the shortest path information consists of the next hope and next to last hope on the path. Whenever two fragments merge, the leader with the smaller ID between the two fragments becomes the new leader of the resultant fragment. Merging of two fragments are accomplished in two steps *1) discovery* and *2) connection*. During the discovery step, each fragment leader updates the information about the other neighbouring fragments. The leader node of a fragment finds the closest neighbouring fragment from the gathered information with whom it merges with. Discovery step is depicted in Figure 2.1.

During the connection step a fragment leader sends a merge request to the leader of the closest fragment and if the connection request is accepted then the leader node with the lowest ID of the two fragments initiates the merging operation. After the successful merging leader with the lowest ID becomes the new fragment leader. The round and message complexities of the K-SPH based algorithm due to Bauer and Varma [12] are $O(Dn)$ and $O(nt)$ respectively.

The second distributed algorithm due to Bauer and Varma [12], which is based on the SPH is a special case of the distributed K-SPH algorithm. The SPH is inherently a sequential

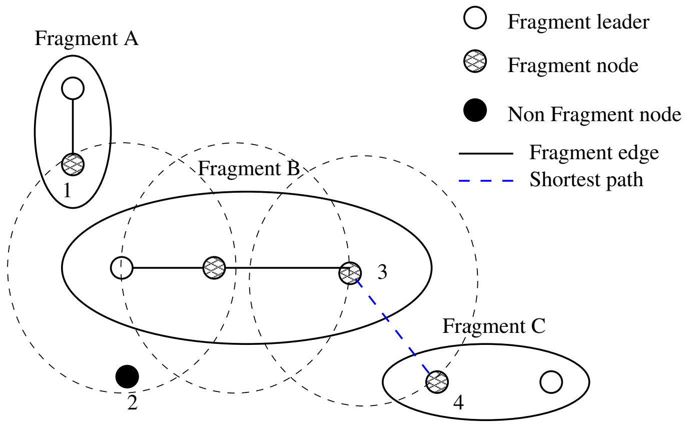


**Figure 2.1:** *An example of the discovery step in the distributed K-SPH due to Bauer and Varma [12]. In the current state of the algorithm, the leader node of the fragment $B$ believes that fragment $C$ is the closest fragment. During the discovery step, the leader node of fragment $B$ instructs its all fragment's nodes to query those nodes which are closer than the fragment $C$. The distance between the fragment $B$ and the fragment $C$ is the distance between node 3 and 4, and it is marked by dotted circles around each node in fragment $B$. Since nodes 1 and 2 fall within one such circle, they receive queries and the leader of the fragment $B$ discovers that the fragment $A$ is the closest one.*

algorithm, since there is only one subtree expanding itself at any time during the execution of the algorithm and nodes must join the tree serially. The K-SPH, on the other hand, allows many of the join operations to proceed in parallel. The time and message complexities of the SPH based algorithm are $O(Dt)$ and $O(nt)$ respectively.

The simulation implemented in [12] showed that both the heuristics, i.e. K-SPH and SPH, produce multi-cast trees whose quality is within 4% of that of the best solution found by many other heuristics (both centralized and decentralized) in 90% of the test cases.

Novak and Rugelj [106] improved the algorithms proposed in [12] in terms of the time and the approximation factor. They showed the correctness of their results by simulation; however, the worst-case complexities remain the same as that of [12].

Singh and Vellanki [124] studied the distributed algorithms for constructing multi-cast trees (Steiner trees), and presented a K-SPH based algorithm which improved (by simulation) the message complexity to $O(n \log t)$. Note that all the algorithms proposed in [12, 81, 106, 124] consider the following system environments.

- The LOCAL model of distributed computing.
- Nodes have the prior information of distances to all other nodes in a network.

Chalermsook et al. [26] proposed a 2-approximate distributed algorithm for the ST problem which is based on the distributed MST algorithm due to Galleger et al. [50]. Initially the input graph $G$ is decomposed into clusters containing a terminal as a central node in each of the clusters. The set of clusters are considered as a set of nodes and the sequence of edges on a path that connect two clusters is considered as an edge. These set of nodes and edges form a new weighted graph $G'$. The weight of an edge in $G'$ is computed by considering the distance between the two corresponding terminals in the original graph $G$. On graph $G'$ the algorithm follows the steps of the algorithm due to Gallager et. al. [50] but starts with clusters instead of single nodes. This constructs an MST spanning all the clusters which also implies the MST containing the same set of terminals. Also a distance metric $d'$ is calculated (which gives a complete graph $K_Z$) on weighted graph $G'$. It is shown that the MST on $K_Z$ induced by metric space $d'$ and the MST on the terminal set of $G$ induced by the metric space $d$ is same. This MST heuristic algorithm has an approximation ratio of 2 for the ST problem in distributed setting. The round and message complexities of this algorithm are $O(n \log n)$ and $O(tn^2)$ respectively.

Gatani et al. [52] proposed a distributed ST algorithm based on ADH in the LOCAL model. In this work it is assumed that each node $i$ knows the routing table (which is provided by the network layer protocol) that provides the minimum cost paths to each of the destinations $j$ and also the next hop in the path from $i$ to $j$. Using this routing table a node sends messages via the minimum cost path to any destination. Initially any node may receive a list of IDs of multi-cast members from the external user. The receiving node (considered as the root node in the first phase) builds a data structure with the multi-cast members and their IDs (unique). The root initially constructs a spanning tree of the whole network. Using this spanning tree root broadcasts the information of multi-cast forest. On receiving this information from the root and with the locally available routing table, each node calculates the path distance to each of the external fragments and select the minimum path distance (denoted as $f$). Calculated value $f$ is converged cast towards the root node. After receiving all the $f$ values from its child nodes, the root node selects the minimum $f$ value and sends it to node say $v$ that computed this value. Whenever node $v$ receives the message from the root node then node $v$ becomes the most central node and starts the merging process that connects the closest fragment via the minimum cost path. During the merging process, the state of each node along the connecting paths and the information about the multi-cast forest are updated. If all multi-cast nodes are already covered by the newly connected fragment then algorithm terminates. Otherwise $v$ becomes new root and starts the next phase of the algorithm. The round and message complexities of this algorithm are $O(Dt)$ and $O(m + tn)$ respectively, where $D$ is the unweighted diameter of the network.

Khan et al. [76] proposed a distributed algorithm in the CONGEST model for a more generalized version of the ST problem known as the generalized Steiner forest (GSF)[1], which is randomized in nature and based on a probabilistic tree embedding due to Fakcharoenphol, Rao, and Talwar [43] (FRT embedding). Considering that the nodes are ranked according to a random order, they gave an almost time-optimal distributed algorithm for computing *least element* (LE) lists, which is a distributed data structure. Then using the LE lists an FRT embedding is implicitly computed. Using this embedding, they obtained an $O(\log n)$-approximate distributed algorithm for the GSF problem whose round and message complexities are $O(Sk \log^2 n)$ and $O(Sn \log n)$ respectively, where $S$ is the *shortest path diameter* of the graph (the definition is deferred to Chapter 3, Section 3.1) and $k$ is the number of terminal components (disjoint subsets of terminals) in the input graph. Note here that if $k = 1$ then the GSF problem reduces to the ST problem. In this case the round complexity reduces to $O(S \log^2 n)$, whereas the message complexity remains the same as that of the GSF algorithm.

Based on the approximate distributed solutions to the (weighted) all pairs shortest paths (APSP) problem, Lenzen and Patt-Shamir [90] presented two distributed algorithms for the Steiner forest problem (a generalized version of the ST problem) in the CONGEST model: one is deterministic and the other one is randomized. The former one finds, a $(2 + o(1))$-approximate Steiner forest using $\tilde{O}(\sqrt{\min\{D, k\}}(D + k) + S + \sqrt{\min\{St, n\}})$ rounds. The latter one finds a $(2 + o(1))$-approximate Steiner forest using $\tilde{O}(\sqrt{\min\{D, k\}}(D + k) + \sqrt{n})$ rounds with high probability. Note that if $k = 1$ then the Steiner forest problem reduces to the ST problem. In this case the round complexities of the two algorithms in [90], in which one is deterministic and the other one is randomized reduce to $\tilde{O}(S + \sqrt{\min\{St, n\}})$ and $\tilde{O}(D + \sqrt{n})$ respectively.

Performances of some of the distributed algorithms for the ST problem are summarized in Table 2.4.

[1]The **GSF problem** is defined as follows: given a weighted graph $G = (V, E, w)$, and a collection of $k$ disjoint subsets (groups) of $V : V_1, V_2, ..., V_k$, the goal is to find a minimum weight subgraph in which each pair of nodes belonging to the same group $V_j$ is connected.

**Table 2.5:** *Summary of related work of the PCST problem in sequential setting. LP: linear programming, ILP: integer LP, MA: memetic algorithm, DP: dynamic programming, ES: empirical study, GW-clustering: Goemans and Williamson clustering [58], "*" indicates that the corresponding parameter is is not discussed in that related work. $\epsilon$ is any constant $> 0$. $c$ is any constant.*

| **Authors** | **Heuristics** | **Remarks** | **Time Complexity** | **Approx.** |
|---|---|---|---|---|
| Bienstock et al. [16] | LP | First Work | * | 3 |
| Goemans et al. [58] | LP | Rooted PCST | $O(n^2 \log n)$ | $2 - \frac{1}{n-1}$ |
| Johnson et al. [72] | LP and Strong pruning | Unrooted PCST | $O(n^2 \log n)$ | $2 - \frac{1}{n}$ |
| Cole et al. [29] | GW-clustering | Implementation | $O(c(n+m)\log^2 n)$ | $(2 + \frac{1}{n^c})$ |
| Caunto et al. [79] | Local Search, Perturbation | ES | * | * |
| Klau et al. [78] | MA and ILP | ES | * | * |
| Ljubic et al. [95] | MA and ILP | ES | * | * |
| Feofiloff et al. [44] | LP | Unrooted PCST | $O(n^2 \log n)$ | $2 - \frac{2}{n}$ |
| Miranda et al. [2] | DP | 2-tree, Interval Data | * | 1 |
| Bateni et al. [10] | Tree Decomposition | Planar Graphs | * | $1 + \epsilon$ |
| Aarcher et al. [4] | LP | Lagrangian-preserving | * | 1.9672 |
| Haouari et al. [63] | ILP | Quota PCST, ES | * | * |

## 2.3 Sequential PCST algorithms

The first sequential approximation algorithm for PCST was given by Bienstock et al. [16], although a related problem named prize-collecting travelling salesman problem (PCTSP) was introduced earlier by Balas [7]. Bienstock et al. achieved an approximation factor of 3 by using linear programming (LP) relaxation technique.

Two years later, based on the work of Agrawal, Klein and Ravi [1], Goemans and Williamson [58] proposed a primal-dual algorithm using the LP relaxation for the rooted variant of the PCST problem, which runs in $O(n^2 \log n)$ time. They presented the algorithm for the PCST as part of a study on a novel approximation technique that can be applied to many graph problems based on an Integer Linear Programming (ILP). This algorithm consists of two phases namely *growth* and *pruning* and yields a solution of approximation factor $(2 - \frac{1}{n-1})$ of the optimal. This algorithm is often denoted as *GW-algorithm.*

The empirical behavior of GW-algorithm was first studied by Johnson et al. [72]. They introduced an improved pruning rule termed as *strong pruning*, which typically provides better solutions than the GW-algorithm. On a selection of real-world instances whose underlying graphs are country street maps, the improvement on the approximation factor to that of the GW-algorithm is from 1.7% to 9.2%. Moreover they also presented a review of different PCST related problems. Specifically they studied the unrooted version of the PCST problem and improved the running time to $O(n^2 \log n)$ by maintaining an approximation factor $(2 - \frac{1}{n})$ of the optimal. Note that to get a similar guarantee for the unrooted version of the PCST problem using the original GW-algorithm one might have to run the algorithm for all $n$ potential roots, yielding a running-time bound of $O(n^3 \log n)$.

Cole et al. [29] gave an implementation of the clustering procedure proposed by Goemans and Williamson (GW-clustering) [58]. For graphs with $n$ nodes and $m$ edges, they achieved an $O(c(n+m)\log^2 n)$ time approximation algorithm for the PCST problem (as well as for generalized Steiner tree problem, prize-collecting traveling salesman problem, and 2-edge connected subgraph problem) which guarantees an approximation factor of $(2 + \frac{1}{n^c})$, for any constant $c$. This approximation factor has a slight additive degradation of $\frac{1}{n^c}$ in comparison to that of the original GW-algorithm, which is $(2 - \frac{1}{n-1})$.

Feofiloff et al. [44] introduced a new algorithm for the PCST problem based on the GW-algorithm having a different LP formulation. They achieved a solution of $(2 - \frac{2}{n})$ approximation factor for the unrooted version of the PCST whose running time is $O(n^2 \log n)$.

A *branch-and-cut* based algorithmic framework for the PCST problem, which is based on the integer linear programming (ILP) formulation, was developed by Ljubić et al. [95]. This framework solves many benchmark instances from the literature to optimality, including some of them for which the optimum was not known.

Bateni et al. [11] presented a PTAS for the PCST problem on *planar* graphs. Specifically they showed that an $\alpha$-approximation algorithm for the PCST problem on graphs of *bounded treewidth*[1] implies an $(\alpha + \epsilon)$-approximation algorithm for the PCST problem on *planar* graphs and also on *bounded-genus* graphs, for any constant $\epsilon > 0$. Bateni et

[1]The basic notion of *treewidth* was introduced by Robertson and Seymour [120]. To define treewidth, a graph is represented by a tree structure, called *tree decomposition*. A tree decomposition of a graph $G = (V, E)$ is a pair, $(T, \mathcal{B})$ in which $T(I, F)$ is a tree and $\mathcal{B} = \{B_i | i \in I\}$ is a family of subsets of $V(G)$ such that 1) $\cup_{i \in I} B_i = V$; 2) for each edge $e = (u, v) \in E$ , there exists an $i \in I$ such that both $u$ and $v$ belong to $B_i$; and 3) for every $v \in V$, the set of nodes $\{i \in I | v \in B_i\}$ forms a connected subtree of $T$. Here each $B_i$ is termed as a bag. The *width* of a tree decomposition is the maximum size of a bag in $\mathcal{B}$ minus 1. The treewidth of a graph $G$ denoted by $tw(G)$, is the minimum width over all possible tree decomposition of $G$.

al. also studied prize-collecting variants of other related problems, namely prize-collecting traveling salesman (PCTS), prize-collecting stroll (PCS), and prize-collecting Steiner forest (PCSF). They proved that there exists PTAS for each of the problems PCS and PCTS on planar graphs and bounded-genus graphs. In contrast, the PCSF is proved APX-hard to approximate on series-parallel graphs, which are planar graphs of treewidth at most 2.

Archer et al. [4] were the first to provide a $(2-\epsilon)$-approximation (for any constant $\epsilon > 0$) algorithm for the PCST problem. Specifically the approximation ratio of this algorithm is 1.9672. They achieved this by using the ST algorithm of Byrka et al. [20] as a black box in their algorithm.

The *quota* version of the PCST problem was studied by Haouari et al. [63]. Given a root node, edge costs, node prizes, as well as a preset quota, the quota version of the PCST finds a subtree that includes the root node and collects a total prize not smaller than the specified quota, while minimizing the sum of the total edge costs of the tree plus the penalties associated with the nodes that are not included in the subtree. Haouari et al. [63] presented two valid 0-1 programming formulations and use them to develop preprocessing procedures for reducing the graph size. By combining preprocessing procedures, effective heuristics, and tight 0-1 programming formulations they showed that the quota version of the PCST problem can be solved optimally for many benchmark instances from the literatures.

Miranda et al. [2] studied the PCST problem for a special graph called *2-tree*.[1] They also considered that the prizes (node weights) and edge weights belong to a given interval. They showed that there exists efficient algorithm for PCST on 2-trees. This result is based on the work of Wald and Colbourn [128] who proved that the ST problem is polynomial time solvable on 2-trees. Miranda et al. [3] also proposed an algorithm for the robust PCST problem.

Geunes et al. [54] showed that any LP-based $\alpha$-approximation algorithm for a *covering* problem can be leveraged to a $(\frac{1}{1-e^{-1/\alpha}})$-approximation algorithm for the corresponding prize-collecting problem, and Li et al. [92] extend this result for sub-modular penalties.

Performances of some of the sequential algorithms for the PCST problem are summarized in Table 2.5.

[1]In graph theory, a $k$-tree is an undirected graph formed by starting with a $(k + 1)$-vertex complete graph then repeatedly adding vertices in such a way that each added vertex $v$ has exactly $k$ neighbors $U$ and together the $k + 1$ vertices formed by $v$ and $U$ form a clique.

### 2.3.1 GW-algorithm

This algorithm is the basis for many other approximate PCST algorithms proposed in literature [44, 72]. Goemans and Williamson [58] proposed a framework based on the *primal-dual* method, which can be used to develop approximation algorithms for various network design problems. They presented an approximation algorithm (the GW-algorithm) for the rooted variant of the PCST problem as a part of their's novel approximation technique which works as follows. It consists of two phases namely *growth* and *pruning*. The growth phase maintains a forest which contains a set of candidate edges being selected for the construction of the PCST. Initially the forest is empty and is considered as a connected component containing a singleton node. The growth phase also maintains a set of components whose possible states can be either *active* or *inactive*. The active state of a component indicates that it is in a growing state, whereas the inactive state indicates that it stops growing tentatively. The growth phase proceeds in iterations. In each iteration, either two components are merged together or one component is deactivated. The algorithm deactivates a component when it finds that the best move is to pay the penalties of all the nodes in that component. In case of merging of two components in which one of the merging components contains the root, the resulting component becomes inactive. The growth phase terminates when there are no active components left. Because at each iteration the total number of active components decreases, the growth phase is guaranteed to terminate. In the pruning phase except the rooted tree, all other trees are pruned from the resulting forest. The algorithm achieves an approximation ratio of $\left(2 - \frac{1}{n-1}\right)$ and its running time is $O(n^2 \log n)$ for a graph of $n$ vertices.

There exist other approaches in literature to construct PCST in the sequential setting. We briefly discuss some of them in the following subsections.

### 2.3.2 Perturbation algorithms

Canuto et al. [22] gave a multi-start local search based algorithm for the PCST problem, which is termed as the *perturbation algorithm*. The overall behavior of an iteration of the algorithm is as follows. A local search is performed on the neighborhood of the result of the GW-algorithm computed on the input graph. If some requirements are met, the solution is kept on a list of elite solutions. When the local search produces a new result, the algorithm randomly selects an elite solution and performs a path-relinking step. The path-relinking step produces an improved solution by exploring trajectories which merges the elite solutions. The final step of the iteration is a perturbation algorithm that modifies the prize of some nodes. Subsequent iteration of the loop feeds the GW-algorithm with the graph perturbed as in the previous iteration. This algorithm successfully outperforms the direct use of the GW-algorithm and computes optimal solutions for multiple instances. Furthermore authors experimented with different variants of the algorithm and established a benchmark of solved instance with up to 1000 nodes and 25000 edges.

### 2.3.3 Memetic and ILP Based algorithms

Memetic algorithms can be seen as "evolutionary algorithms which intend to exploit all available knowledge of the underlying problem" [101] available in the form of greedy heuristics, approximation algorithms, local search, specialized recombination operators, or some other ways. Klau et al. [78] proposed an algorithmic framework to solve the PCST problem based on the combination of Memetic algorithm and ILP, and achieved results that compare positively to the ones of previous publications, showing a significant gain in the running time for medium and large instances. The algorithm framework given by Klau et al. is divided in three phases: 1) a preprocessing phase to reduce the size of the problem, 2) the core Memetic algorithm, and 3) the post-optimization that solves a relaxation of an ILP model constructed from a model for finding the minimum Steiner arborescence in a directed graph.

Ljubić et al. [94] proposed a Memetic algorithm for the PCST problem. This algorithm first creates a pool of initial candidate solutions and then it performs two operations, namely *recombination* and *mutation* starting from two parents candidates. Each iteration which consists of initial creation, recombination and mutation, creates a new solution. The population typically contains different solutions and a newly created one replaces the worst one from the population. Note that after each of the operations, i.e. initial creation, recombination, and mutation, a local improvement procedure is applied. The local improvement is an algorithm that computes the optimal solution in case the input graph is a tree. The three operations of the Mametic algorithm due to Ljubić et al. [94] are discussed bellow.

- **Initial creation.** The initial candidate solutions are created as follows: Some nodes with prizes greater than some threshold value are considered as the terminals for the ST, and the rest of the nodes are considered as Steiner nodes. Now a full connected subgraph inferred by the terminal set is computed. After that an MST of the full connected subgraph is computed. In this way an ST is computed, which connects all the terminal nodes. Since the ST computation is an MST heuristic, the approximation ratio of the resultant ST is 2 of the optimal [57].

- **Recombination.** The recombination operation is designed to provide the highest possible heritability. The recombination operation picks two candidate solutions and creates a new one by performing merging operation. If the two solutions share at least one vertex, then an spanning tree is constructed over the union of their edge sets. When the parent solutions are disjoint, a customer vertex from each of the solutions is randomly chosen and connects them by the minimum weighted path. Furthermore, in order to avoid premature convergence of the algorithm, the set of edges of the new candidate is selected by the construction of a random spanning tree among its set of vertices.

- **Mutation.** In the mutation operation small changes are made in the current solution. This is achieved by connecting one or more new customers to the current solution.

## 2.4 Distributed PCST algorithms

Regarding distributed PCST we were able to find only one manuscript in the literature, by Rossetti [122], where the following algorithms were proposed: (i) *Tree Algorithms* that solve PCST on trees to optimality, (ii) MST based Algorithm that fails to guarantee any constant approximation ratio, and (iii) Distributed GW-algorithm, which is in essence centralized and of very limited practical value.

**Tree Algorithms.** Rossetti proposed three variants of the Tree Algorithm, namely *Rooted Tree Algorithm* (RTA), *Simple Unrooted Tree Algorithm* (SUA), and *Unrooted Tree Algorithm* (UTA). Note that Tree Algorithms solve the Net Weight Maximization variant of the PCST problem on trees, which is defined as follows.

**Definition 2.4.1** (Net Weight Maximization PCST problem [72]). *Given a connected weighted graph $G = (V, E, p, w)$ where $V$ is the set of vertices, $E$ is the set of edges, $p : V \to \mathbb{R}^+$ is a vertex prize function and $w : E \to \mathbb{R}^+$ is an edge weight function, the goal is to find a subtree $T = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ that maximizes the* *following quantity:*

$$NW(T) = \sum_{v \in V'} p_v - \sum_{e \in E'} w_e$$

The RTA is a heuristic algorithm which is similar to the combined form of local search and post-optimization applied to candidate solutions [72, 94]. The RTA variant works for the rooted variant of the PCST problem. Specifically the RTA is equivalent to the *strong pruning* function introduced by Johnson et al [72]. The distributed RTA works as follows. The algorithm starts at the leaf nodes of the tree. Let $nw(v)$ denotes the sum of prizes of all nodes minus the sum of weights of all the edges in a subtree rooted at $v$. Initially $nw(v) = p_v$ for each node $v$ in the input tree. Each children $v$ sends its $nw(v)$ to its parent $u$. Upon receiving $nw(v)$, $u$ evaluates the condition $w_{(u,v)} \geq nw(v)$. If this condition is true then $u$ removes the edge $(u, v)$ and the subtree rooted at $v$ from the solution. Otherwise $u$ updates the value of $nw(u)$ to $nw(u) + nw(v) - w_{(u,v)}$. Once $u$ performs the above procedure for all of its children, it reports its $nw(u)$ to its parent. This principle is recursively followed by each node in the tree. Note that in the RTA every node agrees on a single root (r) before the start of the above procedure. The algorithm terminates once $r$ evaluates $nw(r)$. The algorithm has a message complexity of $O(m)$. Considering every message arrives in one unit of time, the total time taken by the algorithm is $2h$, where $h$ is the height of the tree.

To solve the unrooted variant of the PCST problem, Rossetti proposed SUA. In this algorithm it is assumed that every node has a unique integer identifier. The SUA proceeds in iteration. Each iteration selects a node as a root (starting with the lowest identifier) and then applies the RTA. At the end of each iteration, the current root knows the Net Weight value found by the RTA algorithm and every node knows its local solution (i.e. which of its edges are in the solution). The root node also broadcasts a message containing the Net Weight value over the tree on which the current RTA is performed. In this way each node keeps track of the best value and local solution found so far. In the subsequent iterations, the algorithm selects the node with the next lowest value as the root and proceeds with the RTA. The time and message complexities of the SUA are $O(nh)$ and $O(mn)$ respectively.

Note that for an input graph of $n$ nodes, the SUA iterates RTA $n$ times. Therefore SUA does not show a high degree of parallelism on the computation among the nodes. In the UTA variant of the Tree Algorithm, the RTA needs not be run for each node as a root. This increases the level of parallelism on the computation among the nodes. In order to achieve the parallelism, i.e. to perform all the iterations of the RTA algorithm at once, every node of the graph acts as three possible roles at the same time: as root and as both child and parent of every one of its neighbours. The time and message complexities of the UTA variant of the Tree Algorithm are $O(h)$ and $O(m)$ respectively.

**MST based algorithm.** This algorithm works for general graphs. It consists of two components: MST construction and Tree Algorithm. Initially an MST of the input graph is constructed by using one of the known distributed MST algorithms. On the computed MST, one of the Tree Algorithms is applied to compute the Net Weight and the corresponding PCST.

The time and message complexities of the MST based algorithm depend on the implementation of its two components. If the implementation uses the distributed algorithm due to [50] for MST construction and the UTA variant of the Tree Algorithm for PCST construction then its time and message complexities become $O(n \log n)$ and $O(m + n \log n)$ respectively. Note that this MST based algorithm computes a feasible solution to the Net Weight maximization variant of the PCST problem, however it can not guarantee a constant approximation factor.

**Distributed GW-algorithm.** Rossetti [122] adapted the sequential GW-algorithm [58] to the distributed setting. He showed that a PCST can be computed in distributed setting with a guaranteed constant approximation ratio, i.e. is $(2 - \frac{1}{n-1})$, which is same as that of the sequential GW-algorithm. This distributed implementation relies on a directed spanning tree and consists of two phases namely growth and pruning, similar to the sequential GW-algorithm. The growth phase proceeds in iterations that maintains a forest. Initially the forest consists of components and each component contains a singleton node. The growth phase also maintains a set of components whose possible states can be either *active* or *inactive*. Initially except the root, all other components are active. Initially the root component is inactive. Each iteration performs a convergecast operation, followed by a broadcast operation. At the end of each iteration, all nodes agree on the current candidate solution for the problem. The convergecast operation finds a global parameter $\epsilon$ among the one found locally by each node and eventually the root of the directed spanning tree collects it. The root uses $\epsilon$ value to decide whether two components can be merged or one of them can be deactivated. The broadcast operation modifies the forest of connected components by performing a merging or deactivation operation (depending on the value of $\epsilon$) and also updates local variables accordingly. During the convergecast operation the directed spanning tree is used by the nodes to send messages to their root. The directed spanning tree is also used by the root to send messages to all the nodes in the broadcast operation. In each of the iterations of convergecast-broadcast, the directed spanning tree is modified by the algorithm.

Whenever all the components in the network become inactive, the growth phase terminates. Now on the root component, one of the distributed Tree Algorithms discussed above is applied to perform the pruning phase. Note that Rossetti does not discuss the time and message complexities of his distributed GW-algorithm. However from the description of his algorithm it is apparent that its time and message complexities are $O(n^2)$ and $O(mn)$ respectively.

# 3
# Improved distributed approximation for Steiner tree in the CONGEST model

In this chapter we study the Steiner tree (ST) problem in the CONGEST model of distributed computing, which is already defined in Chapter 1. For the sake of completeness of this chapter, we restate the definition as follows.

**Definition 3.0.2** (ST problem). *Given a connected undirected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}^+$, and a set of vertices $Z \subseteq V$, known as the set of terminals, the goal of the ST problem is to find a tree $T' = (V', E')$ such that $\sum_{e \in E'} w_e$ is minimized subject to the conditions that $Z \subseteq V' \subseteq V$ and $E' \subseteq E$.*

The best deterministic round complexity known so far for solving the ST problem in the CONGEST model is $\tilde{O}(S + \sqrt{\min\{St, n\}})$ due to Lenzen and Patt-Shamir [90], where $S$, $n$, and $t$ are the *shortest path diameter* (definition is deferred to Section 3.1), number of nodes, and number of given terminals respectively in the input graph, and the result is optimal upto a factor of $2 + o(1)$. On the other hand, the minimum spanning tree (MST) problem is a special case of the ST problem and best known deterministic round complexity for MST construction in the CONGEST model of distributed computing is $O(D + \sqrt{n} \log^* n)$ due to Garay, Kutten, and Peleg [51, 87], where $D$ is the *unweighted diameter* of the graph.

The deterministic lower bounds on round and message complexities of the MST problem in the CONGEST model are $\Omega(D + \sqrt{n}/\log n)$ [115] and $\Omega(m)$ [86] respectively. Here $m$ is the number of edges in the input graph. Since the ST problem is a generalization of the MST problem, the lower bounds on round and message complexities of the MST problem also apply for the ST problem in the distributed setting. Therefore it is highly desirable to obtain a round or message optimal distributed algorithm which computes a near optimal ST.

In this chapter we present two deterministic distributed algorithms for the ST problem in the CONGEST model. The first one, which will be denoted as *DST algorithm*, achieves an approximation factor of $2(1-1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST. It has a round complexity of $O(S + \sqrt{n}\log^* n)$, which is better than the round complexity of the best distributed ST algorithm known so far [90]. This algorithm also significantly reduces the round complexity gap between solving the ST problem and the MST problem in the CONGEST model. The message complexity of the DST algorithm is $O(Sm + n^{3/2})$.

We also propose a deterministic distributed *shortest path forest* (see Section 3.2 for the definition) algorithm (SPF algorithm) in the CONGEST model that computes an SPF using $O(S)$ rounds and $O(Sm)$ messages. The SPF algorithm will be used as a subroutine in the DST algorithm.

The proposed DST algorithm is inspired by a couple of centralized algorithms. It consists of four steps (each step is a small distributed algorithm)– the first step is to build an SPF of the given input graph $G = (V, E, w)$ with a terminal set $Z$, which is essentially a partition of $G$ into disjoint trees: Each partition contains exactly one terminal and a subset of non-terminals. A non-terminal $v$ joins a partition containing the terminal $z \in Z$ if $\forall x \in Z \setminus \{z\}$, $d(z, v) \leq d(x, v)$. Note here that $d(u, v)$ denotes the (weighted) length of the shortest path between nodes $u$ and $v$. In second step, weights of the edges are suitably changed; in third step, the GKP algorithm (Garay, Kutten, and Peleg [51, 87]) is applied on the modified graph to build an MST; and finally some edges are pruned from the MST in such a way that in the remaining tree (which is the required ST) all leaves are terminals.

The overall performance of the DST algorithm is stated in the following theorem.

**Theorem 3.0.1.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes an ST using $O(S+\sqrt{n}\log^* n)$ rounds in the CONGEST model with an approximation factor of $2(1-1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST.*

The second algorithm, which will be denoted as *modified DST*, is a modification of the first one. It shows the round-message trade-off in computing ST in the CONGEST model of distributed computing. Specifically we replace the third step of the DST algorithm by the singularly-optimal MST algorithm proposed by Elkin [41] which helps us achieve the round and message complexities of $\tilde{O}(S + \sqrt{n})$ and $\tilde{O}(Sm)$ respectively and still achieve an approximation factor of $2(1-1/\ell)$. The polylogarithmic factors involved with the round and message complexities of the modified DST algorithm are $O(\log n)$ and $O(\log^2 n)$ respectively. The modified DST algorithm improves the message complexity of the DST algorithm by dropping the additive term of $O(n^{3/2})$ at the expense of a logarithmic multiplicative factor in the round complexity.

The overall performance of the modified DST algorithm is summarized in the following theorem.

**Theorem 3.0.2.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists a deterministic distributed algorithm that computes an ST using $\tilde{O}(S + \sqrt{n})$ rounds and $\tilde{O}(Sm)$ messages in the CONGEST model with an approximation factor of $2(1 - 1/\ell)$.*

As a by-product of the above theorem, for networks with constant or sufficiently small shortest path diameter ($S = O(\log n)$) the following corollary holds.

**Corollary 3.0.2.1.** *If $S = O(\log n)$ then a $2(1 - 1/\ell)$-approximate ST can be deterministically computed with the round and message complexities of $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively in the CONGEST model.*

Note that in case of $S = O(\log n)$, the round and message complexities of the modified DST algorithm are $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively, which almost coincide with the best known singular-optimality results of distributed MST construction in the CONGEST model due to [41, 62, 110] and the approximation factor of the resultant ST is at most $2(1 - 1/\ell)$.

**Organization.** The rest of the chapter is organized as follows. In Section 3.1 we define the system model and notations. Section 3.2 contains the description, an illustrating example, and the correctness of the SPF algorithm. Section 3.3 contains the description, an illustrating example, and the correctness of the DST algorithm. Section 3.4 describes the round-message trade-off in distributed ST construction. Finally Section 3.5 contains some concluding remarks.

## 3.1 Model and Notations

**System model**. We consider the CONGEST model as specified in [114]. A communication network is modelled as a weighted undirected graph $G = (V, E, w)$, where $V$ is the set of nodes, $E$ is the set of communication links, and $w : E \to \mathbb{R}^+$ is a weight function. We assume that each node has a unique identity (ID) which can be encoded in $O(\log n)$ bits. Each node knows the weight of each edge incident on it. We also assume that the weight of each edge in $G$ is either a non-negative integer or a non-negative real number which can be encoded in $O(\log n)$ bits and therefore polynomially many sums of weights can be encoded in $O(\log n)$ bits. Nodes communicate and coordinate their actions with their neighbors by passing messages (of size $O(\log n)$ bits) only. In general, a message contains a constant number of edge weights, node IDs, and arguments (each of them is polynomially bounded in $n$).

The algorithm proceeds in synchronous rounds as follows. At the beginning of each round, each node receives all the messages sent to it. After that each node performs some local computation based on the messages received in that round. Then each node sends (possibly different) messages on its incident links, which will be processed in the next round. The round complexity is measured by the number of rounds required until all the nodes terminate. The message complexity is measured by the number of messages sent until all the nodes terminate. It is assumed that nodes and links do not fail. Also, it is considered that nodes are computationally unbounded.

**Formulation of the ST problem in the CONGEST model**. The input graph $G = (V, E, w)$ coincides with the communication network. Each vertex of $G$ is uniquely mapped to a node (processor) and edges of $G$ are naturally mapped to the links between the corresponding nodes. Each node knows whether the vertex assigned to it is a terminal or not. All nodes run the same algorithm for the ST problem. Regarding output, whenever an algorithm for the ST problem terminates, each node knows whether the vertex assigned to it is in the solution or not and which of its incident edges are in the solution. For termination, each node terminates the algorithm for the ST problem in a finite time.

**Notation**. We use the following terms and notations.

- $w_e$ denotes the weight of an edge $e$.

- $\delta(v)$ denotes the set of edges incident on a node $v$. Similarly $\delta(C)$ denotes the set of edges having exactly one endpoint in a subgraph $C$.
- $h(v)$ denotes the height of a node $v$ with respect to a tree.
- $child(v)$ denotes the set of child nodes of a node $v$ with respect to a tree.
- $s(v)$ denotes the source node of a node $v$. Intuitively $s(v)$ denotes the root of a subtree in which $v$ is a descendant.
- $ts(v)$ denotes the *tentative* $s(v)$ of a node $v$.
- $d(u, v)$ denotes the weighted distance[1] between nodes $u$ and $v$.
- $d(v)$ denotes the weighted distance between $v$ and $s(v)$. Similarly $td(v)$ denotes the (weighted) tentative distance between $v$ and $ts(v)$.
- Let $e \in \delta(v)$. Then at node $v$, $tdn(e)$ denotes the *tentative distance of a neighbor node* incident on the other end of $e$. Similarly at node $v$, $idn(e)$ and $tsn(e)$ denote the *ID* and the *tentative source* respectively of a neighbor node incident on the other end of $e$.
- $\pi(v)$ and $t\pi(v)$ denote the *predecessor*[2] and the *tentative predecessor* respectively of a node $v$.
- $\langle M\rangle$ denotes the message $M(a_1, a_2, ...)$. Here $a_1, a_2, ...$ are the arguments of the message $M$. Note that unless it is necessary, arguments of $\langle M\rangle$ will not be shown in it.
- $\rho(u, v)$ denotes the number of edges in a weighted shortest path between $u$ and $v$. Note that there may be more than one weighted shortest path between $u$ to $v$. In this case $\rho(u, v)$ is the number of edges of the weighted shortest path having the least number of edges.
- $S(G, w)$ (or $S$ for short) denotes the *shortest path diameter* of a given undirected weighted graph $G$, which was first introduced by Khan and Pandurangan [77]. It is defined as follows.

  $$S = \max_{u,v \in V} \rho(u, v)$$

  Note here that $1 \le D \le S \le n - 1$.

[1]The distance between two vertices in a graph is the length of the shortest path connecting them.
[2]In this thesis we use predecessor and parent interchangeably.

## 3.2 SPF construction

**Definition 3.2.1** (SPF [27]). *Let $G = (V, E, w)$ be a connected undirected weighted graph, where $V$ is the vertex set, $E$ is the edge set, and $w : E \to \mathbb{R}^+$ is the non-negative weight function. Given a subset $Z \subseteq V$, an SPF is a sub-graph $G_F = (V, E_F, w)$ of $G$ consisting of disjoint trees $T_i = (V_i, E_i, w)$, $i = 1, 2, ..., |Z|$ such that*

- *For all $i$, $V_i$ contains exactly one node $z_i$ of $Z$.*
- *For all $v \in V_i$, $s(v) = z_i$, where $z_i \in Z$*
- *$V_1 \cup V_2 \cup ...V_{|Z|} = V$ and $V_i \cap V_j = \phi$ for all $i \neq j$.*
- *$E_1 \cup E_2 \cup ...E_{|Z|} = E_F \subseteq E$.*
- *The weighted shortest path between $v$ and $s(v) = z_i$ in $T_i$ is one of the weighted shortest paths between $v$ and $s(v)$ in $G$.*[1]

Regarding distributed SPF construction in the CONGEST model, Chen et al. [27] presented a deterministic distributed algorithm to construct an SPF for $Z \subseteq V$ of a graph $G = (V, E, w)$ with the round and message complexities of $O(n(n-t))$ and $O(m+n(n-t))$ respectively, where $n = |V|, t = |Z|$, and $m = |E|$. Lenzen and Peleg [91] studied a similar problem called *source detection.* Given an unweighted graph $G = (V, E)$, a subset $Z \subseteq V$ of source nodes, let $\mathcal{L}_v^{(\infty)}$ denote the (ascending) lexicographically ordered list of pairs $(g(s, v), s)$, where $s \in Z$ and $g(s, v)$ is the length of the (unweighted) shortest path from $s$ to $v$. The $(Z, g', x)$-detection problem requires that each node $v \in V$ learns the first $\min\{x, \lambda_v^{g'}\}$ entries of $\mathcal{L}_v^{(\infty)}$, where $\lambda_v^{g'}$ is the number of sources $s \in Z$ satisfying that $g(s, v) \leq g'$. Lenzen and Peleg [91] showed that $(Z, g', x)$-detection problem can be solved using $\min\{g', D\} + \min\{x, |Z|\}$ rounds in the CONGEST model.

### 3.2.1 Distributed SPF algorithm.

In this subsection we give a detailed description of our proposed SPF algorithm that constructs an SPF using $O(S)$ rounds and $O(Sm)$ messages in the CONGEST model. The algorithm will be used later as a subroutine in the DST algorithm. It is inspired by the well known Bellman-Ford algorithm and computes an SPF $G_F = (V, E_F, w)$ for a given graph

[1]Note that there may exist more than one weighted shortest path between $v$ and $s(v)$ in $G$.

$G = (V, E, w)$ and $Z \subseteq V$. We assume that at the beginning of the algorithm there exists a breadth first search (BFS) tree of $G$ rooted at a terminal $r$. Note that a BFS tree can be computed using $O(D)$ rounds and $O(m)$ messages in the CONGEST model [114].

We also assume that $r$, the root node, knows the height (denoted as $h$) of the BFS tree. Note that the height of a node is the number of edges on the longest path between that node and a leaf, and the height of a tree is the height of the root node. Intuitively $r$ can compute $h$ using a broadcast and convergecast procedure as follows. Initially $r$ sends a message called $\langle \mathit{compute\_height} \rangle$ to all of its child nodes in the BFS tree. Upon receiving $\langle \mathit{compute\_height} \rangle$, each node forwards it to all of its child nodes. Whenever a leaf node $v$ in the BFS tree receives $\langle \mathit{compute\_height} \rangle$, it sets its $h(v)$ to 0 and sends a message called $\langle \mathit{my\_height}(h(v)) \rangle$ to its parent. Upon receiving $\langle \mathit{my\_height}(h(v_i)) \rangle$ from each of its child nodes $v_i \in \mathit{child}(u)$, an internal node $u$ sets $h(u)$ to $\max\{h(v_i) \mid v_i \in \mathit{child}(u) \text{ and } 1 \leq i \leq |\mathit{child}(u)|\} + 1$ and sends the resulted $h(u)$ to its parent.[1] In this way eventually $r$ receives the heights of all of its child nodes. Then it computes $h(r) = \max\{h(v_i) \mid v_i \in \mathit{child}(r) \text{ and } 1 \leq i \leq |\mathit{child}(r)|\} + 1$, which is the required height of the BFS tree. It is obvious that $r$ can compute $h$ of the BFS tree using $O(h)$ rounds and $O(n)$ messages. Since the height of the BFS tree ($h(r)$) can be at most $D$, the round complexity of computing $h(r)$ in the worst-case is $O(D)$.

**Input.** Each node knows whether it is a terminal or not, and the set of edges incident on it. Let $\mathit{bfs\_}\pi(v)$ denotes the parent of a node $v$ in the BFS tree rooted at $r$. Except $r$, all other nodes in the BFS tree know their parents; for $r$ we assume that $\mathit{bfs\_}\pi(r) = r$. Initially, $t\pi(v) = \mathit{nill}$, $ts(v) = \mathit{nill}$, and $td(v) = \infty$ for each node $v \in V$. Note here that the notations $t\pi$, $ts$, and $td$ are used to describe the tentative SPF. Furthermore $r$ uses two additional local variables called *height* and *start_flag* which are initially set to $h$ and *false* respectively.

**Output.** Whenever the algorithm terminates, $d(v) = td(v)$, $\pi(v) = t\pi(v)$, and $s(v) = ts(v)$ for each node $v \in V$.

**Outline of the algorithm.** The special node $r$ initiates the algorithm by setting *start_flag* = *true*, $td(r) = 0$, $t\pi(r) = r$, $ts(r) = r$, and sending $\langle \mathit{update}(r, ts(r), td(r)) \rangle$ messages on all of its incident edges. After that if $r$ does not receive any message in a round, it sets *height* to $\mathit{height} - 1$. Whenever *height* beomes $-1$, $r$ sets *start_flag* to *false* and the algorithm terminates.

Let $U$ and $Y$ denote the set of $\langle \mathit{update} \rangle$ and the set of $\langle \mathit{echo} \rangle$ messages respectively

[1]Here $|\mathit{child}(u)|$ denotes the number of child nodes of $u$ in the BFS tree.

**SPF algorithm** at node $v$ upon receiving a set of messages or no message.

```
1: upon receiving no message
2: if $v = r$ and $v \in Z$ then                                ▷ we assume that $r \in Z$
3:     if $start\_flag = false$ then
4:         $start\_flag \leftarrow true$;                    ▷ spontaneous awaken of the root node
5:         $ts \leftarrow v$; $t\pi \leftarrow v$; $td \leftarrow 0$; $height \leftarrow h$; $update\_flag \leftarrow false$;   ▷ $update\_flag(v)$
   denotes a boolean variable at node $v$.
6:         for each $e \in \delta(v)$ do
7:             send $\langle update(v, ts, td)\rangle$ on $e$
8:         end for
9:     else
10:        $height \leftarrow height - 1$;
11:        if $height = -1$ then
12:            $start\_flag \leftarrow false$;                         ▷ Algorithm terminates
13:        end if
14:    end if
15: end if

16: upon receiving a set of $\langle update\rangle$ or $\langle echo\rangle$ messages        ▷ $U \neq \phi$ or $Y \neq \phi$
17: if $v \in Z$ and $td = \infty$ then              ▷ for the first time $v$ receives some messages
18:     $ts \leftarrow v$; $t\pi \leftarrow v$; $td \leftarrow 0$;
19:     if $update\_flag = false$ then
20:         $update\_flag \leftarrow true$;
21:     end if
22: else if $v \in V \setminus Z$ then
23:     for each $\langle update(idn(e), tsn(e), tdn(e))\rangle \in U$ such that $e \in \delta(v)$ do
24:         if $tdn(e) + w_e < td$ then
25:             $td \leftarrow tdn(e) + w_e$; $t\pi \leftarrow idn(e)$; $ts \leftarrow tsn(e)$;
26:             if $update\_flag = false$ then
27:                 $update\_flag \leftarrow true$;
28:             end if
29:         end if
30:     end for
31: end if
32: if $update\_flag = true$ then
33:     for each $e \in \delta(v)$ do
34:         send $\langle update(v, ts, td)\rangle$ on $e$
35:     end for
```

**SPF algorithm** (continued).

```
36:     send $\langle echo \rangle$ to $bfs\_\pi$;
37:     $update\_flag \leftarrow false$;
38: else
39:     if $v = r$ then
40:         $height \leftarrow h$;
41:     else if $Y \neq \phi$ then
42:         send $\langle echo \rangle$ to $bfs\_\pi$;
43:     end if
44: end if
```

received by a node in a round. Upon receiving a set of $\langle update \rangle$ or $\langle echo \rangle$ messages ($U \neq \phi$ or $Y \neq \phi$) a node $v$ acts as per the following rules.

R1. if $v = r$ then it resets *height* to $h$.

R2. if $v \in Z$ and $td(v) = \infty$ and for the first time it receives some messages then it sets $ts(v) = v$, $td(v) = 0$, and $t\pi(v) = v$.

R3. if $v \in V \setminus Z$ and $U \neq \phi$ then it computes $w_e + tdn(e)$ for each $\langle update(idn(e), tsn(e), tdn(e)) \rangle \in U$ and chooses the minimum one, say $w_{e'}+tdn(e')$ resulted by $\langle update(idn(e'), tsn(e'), tdn(e')) \rangle \in U$. If $td(v) > w_{e'} + tdn(e')$ then it updates $td(v) = w_{e'} + tdn(e')$, $ts(v) = tsn(e')$, and $t\pi(v) = idn(e')$. Otherwise, $td(v)$, $ts(v)$ and $t\pi(v)$ remain unchanged.

R4. if $td(v)$ is updated then

(a) sends $\langle update(v, ts(v), td(v)) \rangle$ on all of its incident edges.

(b) sends $\langle echo \rangle$ to $bfs\_\pi(v)$.

R5. if $td(v)$ is not updated and $Y \neq \phi$ then $v$ sends $\langle echo \rangle$ to $bfs\_\pi(v)$.

**Termination detection.** Termination of an algorithm is a state in which no message is in transit or sent by any node in the network. The root $r$ of the BFS tree detects the termination of the SPF algorithm. Whenever $r$ finds that the local variable *height* becomes $-1$, it terminates the algorithm. At the beginning of the algorithm, $r$ sets *height* to $h$. If $r$ does not receive any message in a round, it sets *height* to *height* $-$ *1*. During the execution of the algorithm a node $v$ sends an $\langle echo \rangle$ message to $bfs\_\pi(v)$ in any one of the following cases.

1. it updates its local state
2. receives $\langle echo \rangle$ message from at least one of its child nodes

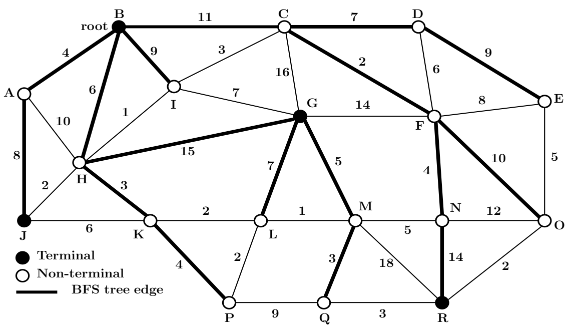
Above two cases guarantee that an $\langle echo \rangle$ message generated at any node in the network eventually reaches $r$. If $r$ receives some messages, it resets *height* to $h$. Since $h$ is the height of the BFS tree, it is guaranteed that from the time of any change of local state in a node (which generates an $\langle echo \rangle$ message), $r$ receives this information (by receiving an $\langle echo \rangle$ message) after at most $h$ rounds. This ensures that $r$ resets *height* to $h$ before it becomes $-1$ in at most $h$ rounds from the time of any changes occur in the network. In case $r$ does not receive any message for $h + 1$ consecutive rounds, this ensures that no changes have been occurred at any node in the network in last $h + 1$ rounds. In this case *height* becomes $-1$ and $r$ terminates the algorithm by setting *start_flag* to *false*.

### 3.2.2 An illustrative example of the SPF algorithm

Let us consider the application of the SPF algorithm in a graph $G = (V, E, w)$ as shown in Figure 3.1(a). The thick edges represent the BFS tree. The set of source nodes (terminals) is $Z = \{B, G, J, R\}$ and $B$ is the root of the BFS tree. The initial tentative source, tentative distance, and tentative predecessor of each node are shown in the table. The root node $B$ starts the algorithm by sending $\langle update \rangle$ messages to all of its neighbors. Figure 3.1(b) shows the states of all the nodes after the second round of the algorithm. Upon receiving $\langle update \rangle$ messages, nodes $A$, $C$, $H$ and $I$ update their local information. Arrows along the edges indicate the tentative predecessors ($t\pi$) of the nodes, except for those whose predecessors are yet undefined. Similarly, Figure 3.1(c) shows the state of the graph after the 6th round of execution of the algorithm. The final SPF $G_F = (V, E_F, w)$ (indicated by the thick edges) for $Z$ of $G$ and a table which contains the lengths of shortest paths of all the nodes to their respective sources are shown in Figure 3.1(d).

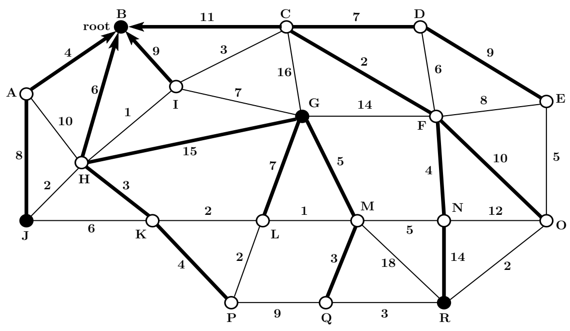
### 3.2.3 Correctness of the SPF algorithm

In this subsection we discuss some of the properties of the SPF algorithm.



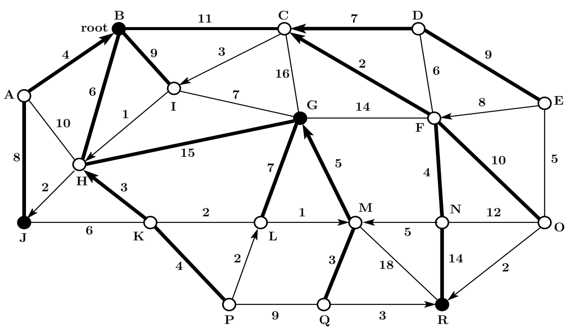
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| source | nil | B | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil |
| distance | $\infty$ | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| predecessor | nil | B | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil | nil |

(a)



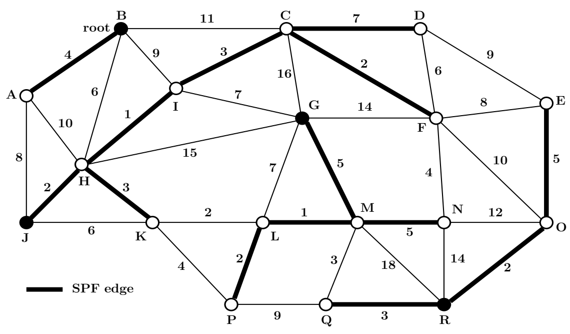
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| source | B | B | B | nil | nil | nil | nil | B | B | nil | nil | nil | nil | nil | nil | nil | nil | nil |
| distance | 4 | 0 | 11 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 6 | 9 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| predecessor | B | B | B | nil | nil | nil | nil | B | B | nil | nil | nil | nil | nil | nil | nil | nil | nil |

(b)



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| source | B | B | J | B | B | B | G | J | J | J | J | G | G | G | R | G | R | R |
| distance | 4 | 0 | 6 | 17 | 20 | 12 | 0 | 2 | 3 | 0 | 5 | 6 | 5 | 10 | 2 | 8 | 3 | 0 |
| predecessor | B | B | I | C | F | C | G | J | H | J | H | M | G | M | R | L | R | R |

(c)



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| source | B | B | J | J | R | J | G | J | J | J | J | G | G | G | R | G | R | R |
| distance | 4 | 0 | 6 | 13 | 7 | 8 | 0 | 2 | 3 | 0 | 5 | 6 | 5 | 10 | 2 | 8 | 3 | 0 |
| predecessor | B | B | I | C | O | C | G | J | H | J | H | M | G | M | R | L | R | R |

(d)

**Figure 3.1:** *(a) A weighted connected graph $G = (V, E, w)$ with the terminal set $Z = \{B, G, J, R\}$, and $B$ is the root of the BFS tree. The initial values are shown in the table. (b) After the second round all neighbors of $B$ receive $\langle update \rangle$ and update their local information. The updated values are shown in the table. (c) After the $6^{th}$ round of execution of the algorithm. (d) The final SPF $G_F = (V, E_F, w)$ for $Z$ of $G$ is indicated by the thick edges.*

**Lemma 3.2.1.** *The SPF algorithm terminates after at most $S + 2h + 1$ rounds.*

*Proof.* At the beginning of the algorithm, $r$ sends $\langle update \rangle$ message to all of its neighbors. Whenever a node $v$ receives $\langle update \rangle$ messages, if applicable, it updates its own states and then sends $\langle update \rangle$ messages including its updated parameters to all of its neighbors and an $\langle echo \rangle$ message to $bfs\_\pi(v)$. This guarantees that after $h$ rounds of execution every node in the network receives at least one $\langle update \rangle$ message.

After $h$ rounds of execution, all nodes proceed in parallel and in every subsequent round, upon receiving $\langle update \rangle$ messages, if applicable, each node updates its local state. Since $S$ is the shortest path diameter, any path in the SPF contains no more than $S$ edges. The SPF algorithm has a single initiator (the root node $r$ of the BFS tree). This implies that a leaf node $v$ may be delayed of updating its local information by $h$ rounds; such a node $v$ converges to its correct $d(v)$ value in at most $S$ additional rounds. Therefore, in the worst-case, after $h + S$ rounds of execution no local changes occur at any node in the network. Note that every time local changes occur at a node $v$, it sends $\langle echo \rangle$ message to $bfs\_\pi(v)$. Since after at most $h + S$ rounds no changes occur at any node in the network, after $h + S$ rounds an $\langle echo \rangle$ message can not be generated by any node in the network due to any local changes. However after $h + S$ rounds of execution of the algorithm, some $\langle echo \rangle$ messages can be still in transit in the network which were generated on or before $(h + S)^{th}$ round. In this case, upon receiving a set of $\langle echo \rangle$ messages, a node $v$ simply sends an $\langle echo \rangle$ message to $bfs\_\pi(v)$. Since $h$ is the height of the BFS tree, it is guaranteed that from the time of any change of local state in a node (which generates an $\langle echo \rangle$ message), $r$ receives this information (by receiving an $\langle echo \rangle$ message) after at most $h$ rounds. In case $r$ does not receive any message for $h$ consecutive rounds, it ensures that no changes have been occurred at any node in the network in the last $h$ rounds. In this case $r$ terminates the algorithm in the next subsequent round. Therefore after at most $S + 2h + 1$ rounds of execution, no further messages related to the SPF algorithm will be sent or in transit in the network. This concludes that the SPF algorithm terminates after at most $S + 2h + 1$ rounds. □

**Theorem 3.2.2.** *The round complexity of the SPF algorithm is $O(S)$.*

*Proof.* Lemma 3.2.1 ensures that the SPF algorithm terminates after at most $S + 2h + 1$ rounds. We know that $h$ is the height of the BFS tree. Since $h \leq D \leq S \leq n - 1$, the round complexity of the SPF algorithm is $O(S)$. □

**Theorem 3.2.3.** *The message complexity of the SPF algorithm is $O(Sm)$.*

*Proof.* During the execution of the SPF algorithm, a terminal node sends $\langle update \rangle$ messages to all of its neighbors only once. This happens only when a terminal node receives $\langle update \rangle$ message for the first time. After that a terminal node never sends any $\langle update \rangle$ messages. It is clear that until the termination of the algorithm, in the worst-case, at most $\Delta t$ $\langle update \rangle$ messages are generated due to the terminal set $Z$, where $t = |Z|$ and $\Delta$ is the maximum degree of a vertex in the input graph $G$. On the other hand every time $td(v)$ is updated at a non-terminal $v$, it sends $\langle update \rangle$ messages to all of its neighbors. Therefore, in the worst-case, in each round, the number of $\langle update \rangle$ messages generated is at most $\Delta(n-t)$ due to the non-terminal set $V \setminus Z$. Since Theorem 3.2.2 ensures that the SPF algorithm terminates in $O(S)$ rounds, in the worst-case, the number of $\langle update \rangle$ messages generated until the termination of the algorithm is $O(\Delta(n-t)S)$ due to the non-terminal set $V \setminus Z$. Therefore total $\langle update \rangle$ messages generated until the termination of the algorithm is $O(\Delta t + \Delta(n-t)S)$. We know that $\Delta t = O(\Delta n)$ and $\Delta(n-t) = O(\Delta n)$, and $\Delta n$ is upper bounded by $O(m)$. This implies that $O(\Delta t + \Delta(n-t)S) = O(Sm)$.

On the other hand during the execution of the algorithm a node $v$ sends an $\langle echo \rangle$ message to $bfs\_\pi(v)$ in the following cases: (i) it updates its local state, or (ii) receives $\langle echo \rangle$ message from at least one of its child nodes. This implies that in a round a node can generate at most one $\langle echo \rangle$ message. Since after $h$ rounds of execution (from the beginning of the algorithm) all nodes proceed in parallel, in the worst-case, in each round, all nodes can together generate at most $n$ $\langle echo \rangle$ messages. Theorem 3.2.2 ensures that the SPF algorithm terminates in $O(S)$ rounds. This implies that total $\langle echo \rangle$ messages generated until the termination of the algorithm is $O(Sn)$.

It is clear that total messages generated until the termination of the algorithm is $O(Sm+Sn)$. Since the input graph $G$ is connected, $n \leq m$. Therefore the overall message complexity of the SPF algorithm is $O(Sm)$.

□

**Lemma 3.2.4.** *Let $td_i(v)$ be the length of the tentative (weighted) shortest path from node $v$ to $ts(v)$ after $i$ rounds ($i \geq 0$). Suppose the SPF algorithm terminates after $X$ rounds ($X \leq S + 2h + 1$). Then for each node $v \in V$, $td_X(v) = d(v, s(v))$ and the corresponding shortest path contains at most $S$ edges.*

*Proof.* The root $r$ of the BFS tree initiates the algorithm by sending an $\langle update \rangle$ message to all of its neighbors. This indicates that a node which is $x$-hops away from $r$ receives $\langle update \rangle$ message for the first time after $x$ rounds of execution of the algorithm. A node remains at its initial state until it receives some $\langle update \rangle$ messages. If a node receives $\langle update \rangle$ messages for the first time, it always updates its local state.

Let $x_v$ denotes the round at which a node $v \in V$ receives $\langle update \rangle$ messages for the first time. This indicates that a node $v$ starts changing its local state from $x_v^{th}$ round onwards.

Therefore we assume that for a node $v$, Lemma 3.2.4 holds only after $x_v^{th}$ round onwards.

It is obvious that after $h$ rounds of execution Lemma 3.2.4 trivially holds for all terminal nodes. This is because during the execution of the algorithm, terminal nodes do not maintain distance information to any other node in the network; they keep the record of only their own local information. Whenever a node $v \in Z$ receives $\langle update \rangle$ message for the first time, it updates its $s(v) = ts(v) = v$, $d(v) = td(v) = 0$, and these values remain same throughout the execution of the algorithm. The information $s(v) = v$ and $td_h(v) = 0$ for a node $v \in Z$ together ensure that the corresponding path contains 0 edges. Therefore after $h$ rounds, Lemma 3.2.4 always holds $\forall v \in Z$.

Now we prove that after $x_v + k$ $(k \geq 0)$ rounds of execution Lemma 3.2.4 holds $\forall v \in V \setminus Z$. For the sake of simplicity we divide the proof into two parts. In the first part we show the correctness for the case of $x_v + k \leq S$ for $v \in V \setminus Z$. In the second part we show the correctness for the case of $1 + S \leq x_v + k \leq S + 2h + 1$ for $v \in V \setminus Z$.

**Case 1:** $x_v + k \leq S$**.** By induction on $k$, we show that after $x_v + k$ rounds of execution, $td_{x_v+k}(v)$ is the length of the tentative shortest path from $v$ to $ts(v)$ that contains at most $x_v + k$ edges.

**Base case.** For a node $v \in V \setminus Z$, the base case starts only after $x_v$ rounds. If $k = 0$, it is obvious that first changes occur at node $v$ in the $x_v^{th}$ round. Therefore after round $x_v^{th}$, $ts(v) \neq nill$, $td_{x_v}(v)$ is equal to the length of the tentative shortest path from $v$ to $ts(v)$, and the corresponding tentative shortest path contains at most $x_v$ edges.

**Inductive step.** Suppose, for $k > 0$, $td_{x_v+k-1}(v)$ is the length of the shortest path from $v$ to $ts(v)$ and the corresponding path contains at most $x_v + k - 1$ edges.

Let $P$ be the path from $v$ to $ts(v)$ which is formed after $x_v + k$ rounds and $u$ be the node through which $v$ joins $P$. Let $R$ be the path from $u$ to $ts(u)$. Since $v$ joins $P$ in which $u$ is the predecessor of $v$, $ts(v) = ts(u)$. By the induction hypothesis, the path from $u$ to $ts(u)$, which is $R$, contains at most $x_u + k - 1$ edges and $w(R) = td_{x_u+k-1}(u)$, where $w(R)$ denotes the length of $R$.

After $x_v + k$ rounds, $v$ updates $td_{x_v+k}(v) = \min\{td_{x_v+k-1}(v), td_{x_u+k-1}(u) + w_{(v,u)}\}$. Since $v$ joins $u$, $w(P) = td_{x_u+k-1}(u) + w_{(v,u)} = w(R) + w_{(v,u)}$; this implies that $td_{x_v+k}(v) = w(P) < td_{x_v+k-1}(v)$. Note that $v$ first updated $td(v)$ at $x_v^{th}$ round from the beginning of the algorithm and after that the algorithm executed $k$ subsequent rounds. Therefore it is obvious that $P$ contains at most $x_v + k$ edges.

Therefore, for every node $v \in V \setminus Z$, the claim holds for all $k \geq 0$ such that $x_v + k \leq S$.

**Case 2:** $1+S \leq x_v+k \leq S+2h+1$. The Lemma 3.2.1 shows that algorithm may proceed upto $S+2h+1$ rounds. This induces the possibility of further improvement of $td(v)$ for each node $v \in V \setminus Z$ even after $S$ rounds. However, by using the same argument of the induction step of **Case 1**, it can be shown that $td(v)$ correctly converges to $d(v, s(v))$. Since $S$ is the shortest path diameter, any path in the SPF contains no more than $S$ edges. This implies that after $x_v + k$ rounds, even if $x_v + k \geq S + 1$, $td_{x_v+k}(v)$ is the length of the shortest path from $v$ to $ts(v)$ that contains at most $S$ edges.

Therefore we conclude that if the SPF algorithm terminates after $X$ rounds, $td_X(v) = d(v, s(v))$ for each node $v \in V$ and each shortest path in the SPF contains at most $S$ edges. $\square$

**Deadlock freeness.** The SPF algorithm is free from deadlock. Deadlock occurs only if a set of nodes in the network enter into a circular wait state. In the SPF algorithm, a node sends $\langle$*update*$\rangle$ messages to its neighbors only to inform about its local changes or sends $\langle$*echo*$\rangle$ message to its parent node. It never sends resource request message to any other nodes. This ensures that nodes never create circular waiting (or waiting request) for any resource during the execution of the algorithm. Furthermore, since we assume that nodes and links do not fail, the SPF algorithm does not suffer from any deadlock or other problems like counting-to-infinity etc.

**Message size.** The SPF algorithm uses two types of messages: $\langle$*update*$\rangle$ and $\langle$*echo*$\rangle$. The $\langle$*update*$\rangle$ message contains three different arguments and each of them can be encoded using $O(\log n)$ bits, whereas $\langle$*echo*$\rangle$ message contains no arguments. Therefore, the message size in the SPF algorithm is $O(\log n)$ bits.

## 3.3 DST algorithm

### 3.3.1 Preliminaries

**Definition 3.3.1** (Complete distance graph [84, 130]). *A graph $K_Z$ is called a complete distance graph on the node set $Z \subseteq V$ of a connected undirected weighted graph $G = (V, E, w)$ if for each pair of nodes $u, v \in Z$, there is an edge $(u, v)$ in $K_Z$ and the weight of the edge $(u, v)$ is the length of the shortest path between $u$ and $v$ in $G$.*

Our DST algorithm is inspired by the principles of two centralized algorithms— one is by Kou et al. [84] (Algorithm H) and the other one is by Wu et al. [130] (Algorithm M).

For the sake of completeness, below we briefly describe the working principles of both the algorithms.

The worst-case approximation ratio of the DST algorithm follows from the correctness of the Algorithm H. For a given connected undirected weighted graph $G = (V, E, w)$ and a set of terminal nodes $Z \subseteq V$, Algorithm H computes an ST $T_Z$ as follows.

1. Construct a complete distance graph $K_Z$.
2. Find an MST $T_A$ of $K_Z$.
3. Construct a sub-graph $G_A$ of $T_A$ by replacing each edge of $T_A$ with the corresponding shortest path in $G$.
4. Find an MST $T_B$ of $G_A$.
5. Construct an ST $T_Z$ from $T_B$ by deleting edges of $T_B$ so that all leaves of $T_Z$ are terminal nodes.

The running time of the Algorithm H is $O(tn^2)$. Following the principles of both Prim's and Krushkal's algorithms, Wu at el. [130] proposed a faster algorithm (Algorithm M) which improves the time complexity to $O(m \log n)$, achieving the same approximation ratio as of the Algorithm H. The speed-up is achieved by computing a so-called *generalized* MST $T_Z$ for $Z$ of $G$ in one step as opposed to the multiple steps of the Algorithm H. The generalized MST is defined as follows.

**Definition 3.3.2** (Generalized MST [130]). *Let $G = (V, E, w)$ be an undirected weighted graph and $Z$ be a subset of $V$. Then a generalized MST $T_Z$ is a sub-graph of $G$ such that*

- *there exists an MST $T_A$ of the complete distance graph $K_Z$ such that for each edge $(u, v)$ in $T_A$, the length of the unique path between $u$ and $v$ in $T_Z$ is equal to the weight of the edge $(u, v)$ in $T_A$.*
- *all leaves of $T_Z$ are in $Z$.*

It is clear that $T_Z$ is an ST for $Z$ in $G$ and is the actual realization of $T_A$. In summary, the Algorithm M constructs a generalized MST $T_Z$ for $Z$ of $G$ as follows.

Initially, the set of nodes in $Z$ are treated as a forest of $|Z|$ separate trees and successively merge them until all of them are in a single tree $T_Z$. A priority queue $Q$ is used to store the frontier vertices of paths extended from the trees. Each tree gradually extends its branches into the node set $V \setminus Z$. When two branches belonging to two different trees meet at some node then they form a path through that node merging these two trees. The algorithm always guarantees to compute only such paths of minimum length for merging trees.

**GKP algorithm.** Since we use the GKP algorithm proposed by Garay, Kutten, and Peleg [51, 87] as a subroutine in the DST algorithm, here we give a high-level description of the same. Note that GKP algorithm has the best known deterministic round complexity ($O(D + \sqrt{n} \log^* n)$) for MST construction in the CONGEST model. It is a combination of two algorithms: the *GHS algorithm* due to Gallager, Humblet, and Spira [50] and *Pipeline* algorithm due to Peleg [113]. The key idea is to apply GHS algorithm on a given graph until the diameter of each fragment becomes $O(\sqrt{n})$. The GHS algorithm is applied up to $O(\log \sqrt{n})$ phases and at the end it ensures that there are at most $\sqrt{n}$ fragments. The running time of this part is $O(\sqrt{n} \log^* n)$. In the second part it uses the Pipeline algorithm to find at most $\sqrt{n} - 1$ MWOE (minimum weight outgoing edges) to connect all the remaining fragments ($\leq \sqrt{n}$). In the Pipeline algorithm initially a rooted BFS tree $T_r$ is built. Let $r$ be the root of $T_r$. Using the edges of $T_r$, the information about all the candidate edges which are *inter-fragment* tree edges are sent to the root $r$ using pipelining and each intermediate node $v$ of $T_r$ filters out all candidate edges that have largest weight in a cycle. Eventually the root $r$ of $T_r$ collects all inter-fragment tree edges with their weights, constructs an MST ($M'$) by considering each fragment as a super node. Then $r$ broadcasts the edges in $M'$ to all the respective nodes by using the edges of $T_r$. Since the height of $T_r$ is at most $D$ and each node sends at most $\sqrt{n}$ edges upward (in case of the collection of the inter-fragment tree edges by $r$) and at most $\sqrt{n}$ edges downward (in case of broadcasts), the running time of the Pipeline algorithm is $O(D + \sqrt{n})$. Therefore the overall round complexity of the GKP algorithm is $O(D + \sqrt{n} \log^* n)$. However the second part of the GKP algorithm is responsible for its large message complexity which is $O(m + n^{\frac{3}{2}})$.

### 3.3.2 Outline of the DST algorithm

In this subsection we present the outline of the DST algorithm. It has a round complexity of $O(S + \sqrt{n} \log^* n)$. There are four steps (small distributed algorithms) in the DST algorithm. The speed-up is achieved due to step 1 and step 3. Step 1 computes an SPF using $O(S)$ rounds. Step 3 computes an MST using $O(D + \sqrt{n} \log^* n)$ rounds due to the application of the GKP algorithm.

We assume that a BFS tree rooted at some terminal node ($r \in Z$) of $G$ is available at the start of the algorithm (any node $v \in V$ can be considered as the root of the BFS tree). The root node $r$ initiates the algorithm. Note that an ordered execution of steps is necessary for the correct working of the DST algorithm. We assume that the root $r$ ensures the ordered execution of steps (from step 1 to step 4) and initiates step $i+1$ after step $i$ is terminated. The outline of the DST algorithm is as follows.

Step 1. (SPF construction). Construct an SPF $G_F = (V, E_F, w)$ for $Z$ in $G$ by applying the SPF algorithm described in Subsection 3.2.1. This produces $|Z|$ disjoint *shortest path trees. Each subtree contains one terminal, which is the root of the subtree, and a subset of non-terminals. A non-terminal $v$ is included in a subtree rooted at $z \in Z$ if $\forall x \in Z \setminus \{z\}$, $d(v,z) \leq d(v,x)$.* Theorem 3.2.2 and Theorem 3.2.3 ensure that the round and message complexities of this step are $O(S)$ and $O(Sm)$ respectively.

The graph $G_F$ is the basis for the construction of the generalized MST for $Z \subseteq V$ of $G$ (denoted as $T_Z$). To construct $T_Z$, the following two steps (Edge Weight modification and MST construction) (in order) are necessary.

Step 2. (Edge Weight modification). With respect to the SPF ($G_F = (V, E_F, w)$), each edge $e \in E$ of the graph $G = (V, E, w)$ is classified as any one of the following three types.

(a) *tree* edge: if $e \in E_F$.

(b) *inter_tree* edge: if $e \notin E_F$ and end points are incident in two different trees of the $G_F$.

(c) *intra_tree* edge: if $e \notin E_F$ and end points are incident in the same tree of the $G_F$.

Now transform $G = (V, E, w)$ into $G_c = (V, E, w_c)$. Let $w_c(e)$ denotes the cost of an edge $e$ in $G_c$. For each edge $(u,v) \in E$, $w_c(u,v)$ is computed as follows.

(a) $w_c(u,v) = 0$ if $(u,v)$ is a *tree* edge.

(b) $w_c(u,v) = \infty$ if $(u,v)$ is an *intra_tree* edge.

(c) $w_c(u,v) = d(u) + w_{(u,v)} + d(v)$ if $(u,v)$ is an *inter_tree* edge. In this case $w_c(u,v)$ realizes the weight of a path from the source node $s(u)$ to the source node $s(v)$ in $G$ that contains the *inter_tree* edge $(u,v)$.

The classification of the edges of $G$ and the transformation to $G_c$ can be done as follows. Each node $v$ of $G$ sends a message called $\langle set\_category(v, s(v), d(v), \pi(v))\rangle)$ to all of its neighbors. Let a node $v$ receives $\langle set\_category(u, s(u), d(u), \pi(u))\rangle$ on an incident edge $(v, u)$. If $s(v) \neq s(u)$ then $v$ sets $(v, u)$ as an *inter_tree* edge and $w_c(v, u)$ to $d(v) + w_{(v,u)} + d(u)$. On the other hand if $s(v) = s(u)$ then $(v, u)$ can be either a *tree* edge or an *intra_tree* edge: if $v = \pi(u)$ or $\pi(v) = u$ then node $v$ sets $(v, u)$ as *tree* edge and $w_c(v, u)$ to 0. Otherwise, $v$ sets $(v, u)$ as *intra_tree* edge and $w_c(v, u)$ to $\infty$.

Assuming that the root $(r)$ of the BFS tree initiates this step, it is clear that step 2 can be performed using $O(D)$ rounds. Also on each edge of $G$, the message $\langle set\_category\rangle$ is sent exactly twice (once from each end). Therefore, the message complexity of step 2 is $O(m)$.

Step 3. (MST construction). Construct an MST $T_M$ of $G_c$. $T_M$ contains all *tree* edges and $t - 1$ *inter_tree* edges of $G_c$. Specifically, in this step, we apply the GKP algorithm to construct the MST $T_M$. The round and message complexities of the GKP algorithm are $O(D + \sqrt{n}\log^* n)$ and $O(m + n^{3/2})$ respectively.

Step 4. (Pruning). Construct a generalized MST $T_Z$ from $T_M$. This is accomplished by performing a *pruning* operation in the $T_M$. The pruning operation deletes edges from $T_M$ until all leaves are terminal nodes. Pruning starts at the non-terminal leaf nodes. A non-terminal leaf node $v$ prunes itself from $T_M$ and instructs its neighbor $u$ to prune the common edge $(v, u)$ from the $T_M$. This process is repeated until no further non-terminal leaf nodes remain and we get the final $T_Z$ which contains $t - 1$ *inter_tree* edges and all leave nodes are in $Z$. Now edge weights of the $T_Z$ are restored to $w$.

Since pruning can be started in parallel from all the non-terminal leaf nodes of the $T_M$ and any non-terminal leaf node $v$ can be at most $S$ hops away from its source $s(v)$ (which is a terminal node), step 4 can be performed using $O(S)$ rounds. Also, in pruning, at most one message is sent over each edge of the $T_M$. Since the $T_M$ has exactly $n - 1$ edges, the message complexity of step 4 is $O(n)$.

The DST algorithm terminates with step 4.

### 3.3.3 An illustrative example of the DST algorithm

Let us consider the application of the DST algorithm in a graph $G = (V, E, w)$ shown in Figure 3.2(a). The set of terminals (sources) is $Z = \{B, G, J, R\}$. An SPF $G_F = (V, E_F, w)$ for $Z$ is constructed which is shown in Figure 3.2(b). In $G_F$, each non-terminal node $v$ is connected to a terminal node $s(v) \in Z$ whose distance is minimum to $s(v)$ than any other terminal node in $G$ which is shown in the table of Figure 3.2(b). The construction of graph $G_c$ and labelling of the edge weights according to the definition of $G_c$ are shown in Figure 3.2(c). Figure 3.2(d) shows after the application of the GKP algorithm on $G_c$ which constructs an MST $T_M$ of $G_c$. The final ST $T_Z$ for $Z$ of $G$, which is a generalized MST for $Z$ of $G$ is constructed from $T_M$ by performing a pruning operation, which is shown in Figure 3.2(e).

### 3.3.4 Correctness of the DST algorithm

**Theorem 3.3.1.** *The round complexity of the DST algorithm is $O(S + \sqrt{n}\log^* n)$.*

*Proof.* Each of step 1 and step 4 of the DST algorithm takes $O(S)$ rounds. Step 2 and step 3 take $O(D)$ and $O(D + \sqrt{n}\log^* n)$ rounds respectively. Since $1 \le D \le S \le n - 1$, the overall round complexity of the DST algorithm is $O(S + \sqrt{n}\log^* n)$. □

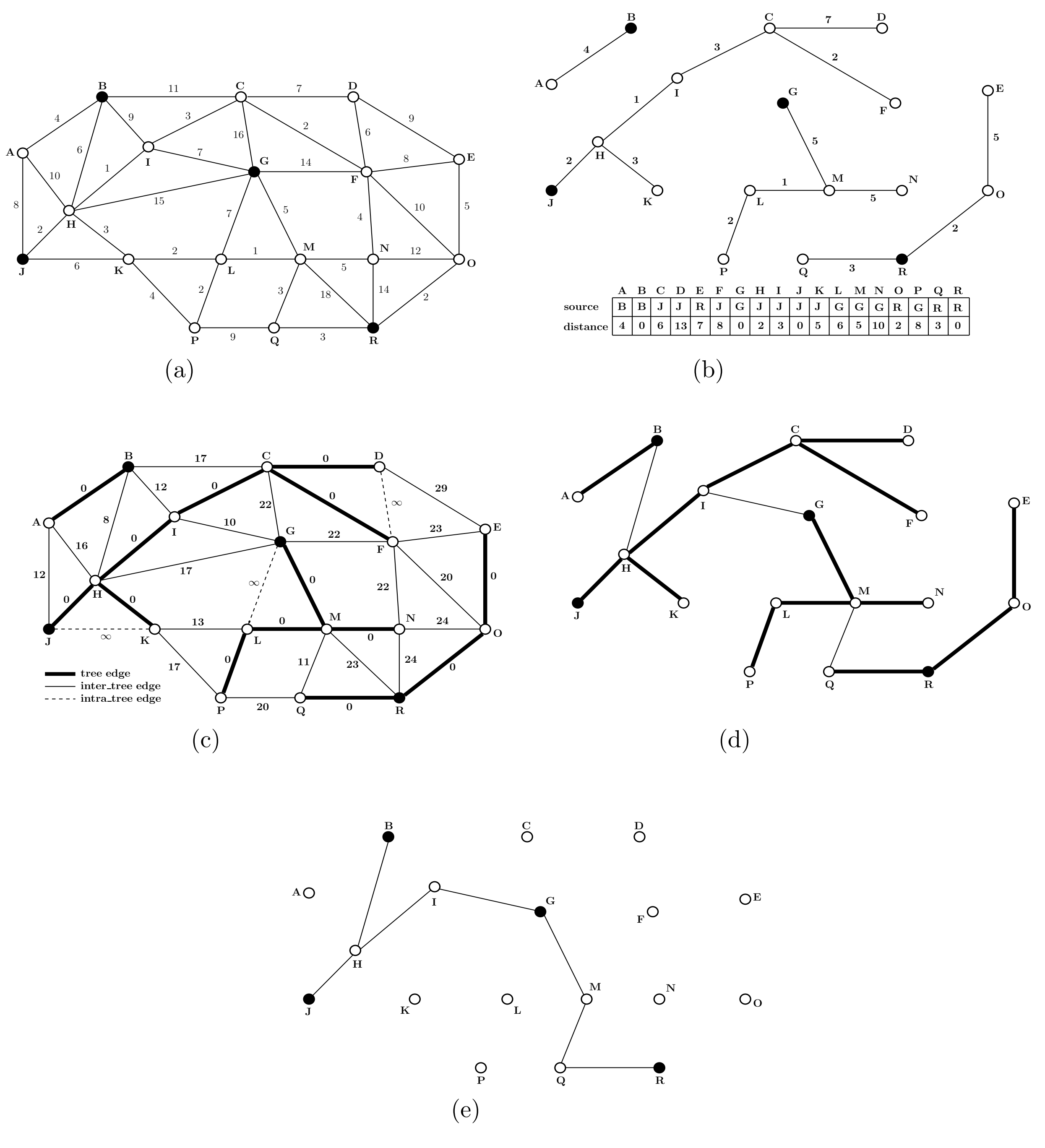
**Theorem 3.3.2.** *The message complexity of the DST algorithm is $O(Sm + n^{3/2})$.*

*Proof.* It is clear that the overall message complexity of the DST algorithm is dominated by step 1 and step 3. By Theorem 3.2.3, the message complexity of step 1 is $O(Sm)$. Also the message complexity of step 3 is $O(m + n^{3/2})$. Combining all steps (step 1 to step 4) we get that the message complexity of the DST algorithm is $O(Sm + n^{3/2})$. □

**Definition 3.3.3 (Straight path).** *Given that $G = (V, E, w)$ is a connected undirected weighted graph and $Z \subseteq V$. Let $u, v \in Z$. Then a path $P_{uv}$ (may contain only one edge) between $u$ and $v$ is called straight only if all the intermediate nodes in $P_{uv}$ are in $V \setminus Z$.*

**Definition 3.3.4.** *Let $X$ be a sub-graph of a graph $G = (V, E, w)$. Then $cost(X)$ denotes the sum of weights of all the edges in $X$.*

**Lemma 3.3.3.** *Given that $G = (V, E, w)$ is a connected undirected weighted graph and $Z \subseteq V$. Let $u, v \in Z$ and there exists a straight path $P_{uv}$ between $u$ and $v$ in $T_M$, where $T_M$*



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| source | B | B | J | J | R | J | G | J | J | J | J | G | G | G | R | G | R | R |
| distance | 4 | 0 | 6 | 13 | 7 | 8 | 0 | 2 | 3 | 0 | 5 | 6 | 5 | 10 | 2 | 8 | 3 | 0 |

**Figure 3.2:** *(a) A graph $G = (V, E, w)$ and a terminal set $Z = \{B, G, J, R\}$. (b) An SPF $G_F = (V, E_F, w)$ for $Z$ of $G$. The distances of nodes to their respective sources are shown in the table. (c) The graph $G_c = (V, E, w_c)$. (d) An MST $T_M$ of the graph $G_c$. (e) The final ST $T_Z$ (generalized MST) for $Z$ of $G$.*

*is a resultant MST constructed after the consecutive applications of step* 1*, step* 2*, and step* 3 *of the DST algorithm on the given graph $G$. Then $P_{uv}$ in $T_M$ is the shortest straight path between $u$ and $v$ in $G$.*

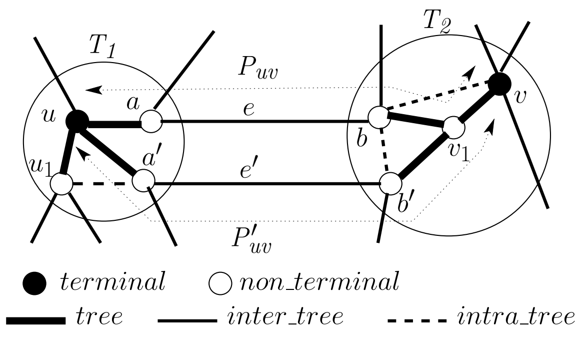


**Figure 3.3:** *A state before merging of two shortest path trees $T_1$ and $T_2$ along a shortest straight path $P_{uv}$. The type of the edges in $T_1$ and $T_2$ are shown according to the graph $G_c$ constructed in step* 2 *of the DST algorithm.*

*Proof.* By contradiction assume that $P_{uv}$ is not the shortest straight path between $u$ and $v$ in $T_M$. Let there exists a straight path $P'_{uv}$ between $u$ and $v$ such that $cost(P'_{uv}) < cost(P_{uv})$. We show that $cost(P'_{uv}) < cost(P_{uv})$ does not hold. Since $u$ and $v$ are two different terminals, they were in different shortest path trees in the SPF, say $u \in T_1$ and $v \in T_2$, before they are included in the $T_M$. Let $e = (a, b) \in \delta(T_1) \cap \delta(T_2)$, $e' = (a', b') \in \delta(T_1) \cap \delta(T_2)$ such that $P_{uv}$ and $P'_{uv}$ contain $e$ and $e'$ respectively as shown in Figure 3.3. Note that the correctness of the SPF algorithm described in Section 3.2 ensures that $P_{ua}$, $P_{ua'}$, $P_{vb}$, and $P_{vb'}$ are the shortest paths between the respective nodes in $G$ as shown in Figure 3.3. In step 2 of the DST algorithm, the weight of each of the edges in the SPF (which are *tree_edges* as specified in step 2 of the DST algorithm) is set to 0. This implies that after step 2 of the DST algorithm $cost(P_{ua}) = cost(P_{ua'}) = cost(P_{vb}) = cost(P_{vb'}) = 0$. Also step 2 of the DST algorithm ensures that $cost(P_{uv}) = w_c(e)$ and $cost(P'_{uv}) = w_c(e')$. During the execution of step 3 of the DST algorithm, which is the MST ($T_M$) construction, $a$ finds $e$ as its minimum weight outgoing edge (MWOE) and $a'$ finds $e'$ as its MWOE. Similarly $b$ and $b'$ find $e$ and $e'$ as their MWOEs respectively. Since during the construction of the $T_M$ the fragments $T_1$ and $T_2$ are merged through $e$, this indicates that $e$ is the MWOE of at least one of the fragments $T_1$ or $T_2$. In other words, $w_c(e') \geq w_c(e)$. We know that $cost(P_{uv}) = w_c(e)$

and $cost(P'_{uv}) = w_c(e')$. This implies that $cost(P'_{uv}) \geq cost(P_{uv})$, contradicting the fact $cost(P'_{uv}) < cost(P_{uv})$. Therefore $P_{uv}$ in $T_M$ is the shortest straight path between $u$ and $v$ in $G$. □

Now we show that the DST algorithm constructs an ST with an approximation factor of $2(1 - 1/\ell)$. Towards this we show the following.

- The computed $T_Z$ is a generalized MST for $Z$ of $G$.
- The cost of $T_Z$ is at most $2(1 - 1/\ell)$ times the cost of the optimal ST ($T_{opt}$), i.e., $cost(T_Z)/\ cost(T_{opt}) \leq 2(1 - 1/\ell)$.

**Lemma 3.3.4.** (Chen et al. [27]). *The length of any path containing an inter_tree edge* $(u, v)$ *between two terminal nodes in* $G$ *is greater than or equal to* $w_c(u, v)$.

The detailed proof of the above lemma can be found in [27].

Consider a graph $T^* = (Z, E^*, w^*)$ whose vertex set is $Z$ and whose edge set $E^*$ is defined from $T_M$ as follows. For each *inter_tree* edge $(u, v)$ of $T_M$, there is an edge $(s(u), s(v))$ in $E^*$ such that $s(u) = a, s(v) = b$, and $w^*(a, b) = w_c(u, v)$. Then we claim the following lemma.

**Lemma 3.3.5.** $T^*$ *is a spanning tree of a complete distance graph* $K_Z$ *for* $Z \subseteq V$ *of graph* $G = (V, E, w)$.

*Proof.* To prove $T^*$ is a spanning tree of $K_Z$, we need to show the followings.

1. $T^*$ is acyclic.
2. $T^*$ spans all the nodes of $K_Z$.

Since $T_M$ is a tree, each pair of nodes in $T_M$ is uniquely connected. This implies that each edge is a cut edge of $T_M$. A path between two distinct terminal nodes in $T_M$ contains at least one *inter_tree* edge of $G_c$. According to the definition of $T^*$, each edge of $E^*$ corresponds to a path which contains exactly one *inter_tree* edge of $T_M$. This implies that each edge of $T^*$ is a cut edge and there exists exactly one unique path for each pair of nodes in $T^*$. Therefore $T^*$ contains no cycle.

It is given that the end points of an edge of $T^*$ correspond to two terminal nodes of $T_M$. Since $T_M$ has exactly $t - 1$ number of *inter_tree* edges and all of them are considered to construct $T^*$, $T^*$ contains exactly $t - 1$ edges. The preceding fact plus the acyclic property of $T^*$ ensure that $T^*$ spans exactly $t$ nodes, i.e. all the nodes of $K_Z$. □

**Lemma 3.3.6.** *$T^*$ is an MST of $K_Z$.*

*Proof.* By contradiction let $T^*$ is not an MST of $K_Z$ and instead of that $T'$ is an MST of $K_Z$. Therefore $T'$ is different from $T^*$ by at least one edge. For simplicity we consider that $T'$ is different from $T^*$ by one edge. Suppose the distinct edges in $T^*$ and $T'$ are $(a^*, b^*)$ and $(a', b')$ respectively. Recall that $w^*(e)$ denotes the weight of an edge $e$ in $K_Z$. Since by assumption $cost(T') < cost(T^*)$ and $cost(T') - w^*(a', b') = cost(T^*) - w^*(a, b)$, this implies that $w^*(a, b) > w^*(a', b')$. We prove by contradiction that $w^*(a, b) > w^*(a', b')$ does not hold.

Since $(a, b)$ is an edge in $T^*$, according to the definition of $T^*$, $(a, b)$ corresponds to an *inter_tree* edge $(u, v)$ of $T_M$ such that $s(u) = a, s(v) = b$ and $w^*(a, b) = w_c(u, v)$. By Lemma 3.3.5 $T^*$ is a spanning tree of $K_Z$. Note that $(a, b)$ is an edge of $K_Z$. This ensures that $w^*(a, b)$ corresponds to the weight of the shortest path between $a$ and $b$ in $G$. Since the path between $a$ and $b$ in $G$ contains the *inter_tree* edge $(u, v)$, by using Lemma 3.3.4 we get

$$d(a, b) \geq w_c(u, v) = w^*(a, b) \tag{3.1}$$

Similarly assuming that the path between $a'$ and $b'$ in $G$ contains the *inter_tree* edge $(u', v')$, where $s(u') = a'$ and $s(v') = b'$, by using Lemma 3.3.4 we get,

$$d(a', b') \geq w_c(u', v') = w^*(a', b') \tag{3.2}$$

Note that $(u, v)$ is an edge of $T_M$. If we remove $(u, v)$ from $T_M$, $T_M$ will split into two subtrees $T_u$ and $T_v$ containing $u$ and $v$ respectively. Since $(u, v)$ is included in the $T_M$, the distributed MST algorithm (GKP algorithm, step 3 of the DST algorithm) must selected $(u, v)$ as an MWOE (minimum weight outgoing edge) to merge $T_u$ and $T_v$ instead of selecting the edge $(u', v')$ as an MWOE. This guarantees that $w_c(u, v) \leq w_c(u', v')$. Now using (3.1) and (3.2) we get $w^*(a, b) \leq w^*(a', b')$, a contradiction to the fact that $w^*(a, b) > w^*(a', b')$. Therefore $T^*$ is an MST of $K_Z$.

Now by contradiction assume that $T'$ is different from $T^*$ by multiple edges. Considering each such edge one by one and using the same logic described above we can show that the assumption does not hold. This concludes that $T^*$ is an MST of $K_Z$.

□

Note that since $T_Z$ is built from $T_M$ by pruning all the non-terminal leaf nodes, all the leaf nodes of $T_Z$ are in $Z$. Furthermore, by Lemma 3.3.6 there exists an MST $T^*$ of $K_Z$ in which each edge $(a^*, b^*)$ of $T^*$ corresponds to a shortest path between the nodes $a^*$ and $b^*$ in $G$. Therefore we claim the following theorem.

**Theorem 3.3.7.** *The tree $T_Z$ computed by the DST algorithm is a generalized MST for $Z \subseteq V$ of $G$.*

Since $T_Z$ is the actual realization of $T^*$ for $Z$ of $G$, $cost(T_Z) = cost(T^*)$. Now if we substitute $T_H$ of the Theorem 1 in Kou et al. [84] by $T_Z$ then we get the following Theorem.

**Theorem 3.3.8.** $cost(T_Z)/cost(T_{opt}) \leq 2(1 - 1/\ell) \leq 2(1 - 1/|Z|)$.

The correctness of the above theorem essentially follows from the correctness of the Theorem 1 of Kou et al. [84]. For the sake of completeness here we give the outline of the correctness. Let $T_{opt}$ consists of $q \geq 1$ edges. Then there exists a loop $L = v_0, v_1, v_2, \cdots, v_{2q}(= v_0)$ in $T_{opt}$ in such a way that

- every edge in $T_{opt}$ appears exactly twice in $L$. This implies that $cost(L) = 2 \times cost(T_{opt})$.
- every leaf of $T_{opt}$ appears exactly once in $L$. If $v_i$ and $v_j$ are two consecutive leaves in $L$ then the sub-path connecting $v_i$ and $v_j$ is a simple path.

Note that the loop $L$ can be decomposed into $\ell$ simple sub-paths (recall that $\ell$ is the number of leaf nodes in the $T_{opt}$), each of them connects two consecutive leaf nodes in $L$. By deleting the longest such simple sub-path from $L$ the remaining path (say $P$) in $L$ satisfies the followings.

- every edge in $T_{opt}$ appears at least once in $P$.
- $cost(P) \leq (1 - 1/\ell) \times cost(L) = (1 - 1/\ell) \times 2 \times cost(T_{opt})$.

Now assume that $T_Z$ is a generalized MST for $Z$ of $G$. We know that $T_Z$ realizes the MST $T^*$ of the complete distance graph $K_Z$ for $Z$ of $G$. In other words $cost(T_Z) = cost(T^*)$. Note that each edge $(u, v)$ in $T^*$, where $u, v \in Z$, is a *shortest straight path* between $u$ and $v$ in $G$. This ensures that the weight of an edge $(u, v)$ in $T^*$ is at most the length of the path (say $P_{uv}$) between nodes $u$ and $v$ in $P$. In other words, for each edge $(u, v)$ in $T^*$, where $u, v \in Z$, $w^*((u, v)) \leq cost(P_{uv})$. If we consider all the edges of $T_A$ then $cost(T_A) \leq cost(P)$. This concludes that $cost(T_Z) = cost(T_A) \leq cost(P) \leq 2 \times (1 - 1/\ell) \times cost(T_{opt})$.

**Deadlock freeness.** The DST algorithm is free from deadlock. We have already shown earlier that step 1 (i.e., the SPF construction) of the DST algorithm is free from deadlock. In step 2, each node independently sends a message (containing its own state) to all of its neighbors only once. Upon receiving messages from all of its neighbors, a node performs some local computation and then terminates itself. Therefore, in step 2 nodes are free from any possible circular waiting. The correctness of the deadlock freeness of step 3 essentially follows from the work of [51, 87]. In step 4, the pruning operation is performed on a tree structure ($T_M$) and a node never requests and waits for resources holding by any other node in the network. This implies that during the pruning operation, nodes are free from any possible circular waiting. Therefore, the deadlock freeness of all the four steps together ensure the deadlock freeness of the DST algorithm.

## 3.4 Round-message trade-off in Distributed ST construction

There exists a few *singularly-optimal* distributed algorithms for the MST problem [41,62,110] which address the long standing time-message trade-off. To the best of our knowledge, till date, no such algorithm exists for the ST problem. In this section we investigate the round-message trade-off of ST construction in the CONGEST model. Towards this we show that a $2(1-1/\ell)$-approximate ST can be deterministically computed using $\tilde{O}(S+\sqrt{n})$ rounds and $\tilde{O}(Sm)$ messages.

Regarding the singularly-optimal distributed algorithms, Pandurangan et al. [110] proposed a randomized singularly-optimal distributed algorithm for the MST problem with the round and message complexities of $\tilde{O}(D+\sqrt{n})$ and $\tilde{O}(m)$ respectively. Note that both round and message complexities of the algorithm proposed by Pandurangan et al. are optimal upto a polylogarithmic factors in $n$ and the factors are $O(\log^3 n)$ and $O(\log^2 n)$ respectively.[1] Elkin [41] proposed a simple deterministic distributed algorithm for the MST problem with the near-optimal round and message complexities of $O((D+\sqrt{n})\log n)$ and $O(m\log n + n\log n \cdot \log^* n)$ respectively. Recently Hauepler et al. [62] proposed a singularly-optimal distributed algorithm for the MST problem which is deterministic in nature but they do not explicitly specify the polylogarithmic factors involved with the round and message complexities. However, it is apparent from their analysis that the rounds and messages

[1] This analysis can be found in [41].

incurred by their algorithm are $O((D + \sqrt{n}) \log^2 n)$ and $O(m \log^2 n)$ respectively.

**Elkin's algorithm [41].** We use Elkin's algorithm as a subroutine in the modified DST algorithm (see Subsection 3.4.1). Note that it is the best singularly-optimal deterministic distributed MST algorithm till date. The algorithm starts with the construction of an auxiliary BFS tree $T_r$ rooted at some root node $r$. It consists of two parts. The first part constructs an $(O(n/D), O(D))$-MST forest[1] which is termed as the *base* MST forest. In the second part, the algorithm in [110] is applied on top of the base MST forest to construct the final MST. Specifically in the second part, the Boruvka's algorithm is applied in such a way that merging of any two fragments requires $O(D)$ rounds.

The first part of the algorithm runs for $\lceil \log D \rceil$ phases. At the beginning of a phase $i$, ($i$ varies from 0 to $\lceil \log D \rceil - 1$), the algorithm computes $(n/2^{i-1}, O(2^i))$-MST forest ($\mathcal{F}_i$). Each node updates its neighbors with the identity of its fragment. This requires $O(1)$ rounds and $O(m)$ messages. Every fragment $F \in \mathcal{F}_i$ of diameter $O(2^i)$ computes the MWOE $e_F$. For each $e_F = (u, v)$, $u \in V(F)$, $v \in V \setminus V(F)$, a message is sent over $e_F$ by $u$ and the receiver $v$ notes down $u$ as a *foreign-fragment* child of itself. In case the edge $(u, v)$ is the MWOE for both the neighboring fragments $F_u$ and $F_v$, the endpoint with the higher identity fragment becomes the parent of the other endpoint. This defines a *candidate fragment graph* $\mathcal{G}'_i = (\mathcal{F}'_i, \mathcal{E}_i)$, where $\mathcal{F}'_i$ is the set of fragments whose diameter is $O(2^i)$ and each such fragment is considered as a vertex of $\mathcal{G}'_i$ and $\mathcal{E}_i$ is the set of MWOE edges of all the fragments in $\mathcal{F}'_i$. The computation of the *candidate fragment graph* $\mathcal{G}'_i$ requires $O(2^i)$ rounds and $O(n)$ messages. Then a *maximal matching* of $\mathcal{G}'_i$ is built by using the Cole-Vishkin's 3-vertex coloring algorithm [30]. The computation of a maximal matching of $\mathcal{G}'_i$ requires $O(2^i \cdot \log^* n)$ rounds and $O(n \cdot \log^* n)$ messages. The phase $i$ ends at the computation of a maximal matching of the candidate fragment graph $\mathcal{G}'_i$. Hence the running time of a phase $i$ is $O(2^i \cdot \log^* n)$ which requires $O(m + n \cdot \log^* n)$ messages. Since the first part runs for $\lceil \log D \rceil$ phases, the overall time and message complexities to construct a base MST forest are $O(D \cdot \log^* n)$ and $O(m \log D + n \log D \cdot \log^* n)$ respectively.

In the second part, the algorithm in [110] is applied on top of the $(O(n/D), O(D))$-MST forest ($\mathcal{F}$). This part maintains two forests: one is the base MST forest $\mathcal{F}$ which is already computed in the first part of the algorithm and the other one is the MST forest $\hat{\mathcal{F}}$ obtained by merging some of the base fragments into the fragments of $\hat{\mathcal{F}}$ via Boruvka's algorithm.

[1]For a pair of parameters $\alpha, \beta$, an $(\alpha, \beta)$-MST forest is an MST forest with at most $\alpha$ fragments, each of diameter at most $\beta$.

The second part of the algorithm requires $O(\log n)$ phases. In each phase $j+1$, it does the following. In every base fragment $F \in \mathfrak{F}$, the MWOE $e = (u,v)$ is computed (in parallel in all base fragments) that crosses between $u \in V(F)$ and $v \in V \setminus V(\hat{F})$, where $\hat{F} \in \mathfrak{F}_j$ is the larger fragment that contains $F$. This step requires $O(D)$ rounds and $O(n)$ messages. Once all $O(n/D)$ MWOEs are computed, all of these information are sent to $r$ of $T_r$. This is done using pipelined convergecast procedure over $T_r$, in which each vertex $u$ in $T_r$ forwards to its parent in $T_r$ the lightest edge for each fragment $\hat{F} \in \mathfrak{F}_j$. This step requires $O(D + |\mathfrak{F}_j|)$ rounds and $O(D \cdot |\mathfrak{F}_j|)$ messages. Then the root node $r$ does the following: (i) computes MWOE $e_{\hat{F}}$ for every $\hat{F} \in \mathfrak{F}_j$, (ii) computes a graph whose vertices are fragments of $\mathfrak{F}_j$ and edges are the MWOEs, and (iii) computes the MST forest $\mathfrak{F}_{j+1}$. After this $r$ sends $|\mathfrak{F}|$ messages over $T_r$ using pipelined broadcast where each message is of the form $(F, \hat{F}')$, and $F \in \mathfrak{F}$ and $\hat{F}' \in \mathfrak{F}_{j+1}$. The root $r_F$ of the base fragment $F$ (each base fragment has a root node) receives this information $(F, \hat{F}')$ and broadcasts the identity of $\hat{F}'$ as the new fragment identity to all the vertices of $F$. This takes $O(D)$ rounds and $O(n)$ messages. Finally each vertex $v$ updates its neighbors in $G$ with the new fragment identity. This requires $O(1)$ rounds and $O(m)$ messages. Combining both the parts, the overall round and message complexities of Elkin's algorithm are $O((D + \sqrt{n}) \log n)$ and $O(m \log n + n \log n \cdot \log^* n)$ respectively.

### 3.4.1 Modified DST algorithm

Observe that the asymptotic term $O(n^{3/2})$ involved with the message complexity of the DST algorithm presented in Subsection 3.3.2 is the bottleneck. To get rid of the asymptotic term $O(n^{3/2})$ from the message complexity we modify the DST algorithm. Similar to the DST algorithm there are four steps (small distributed algorithms) in the modified DST algorithm namely— step 1 (SPF construction), step 2 (Edge weight modification), step 3 (MST construction), and step 4 (pruning). Specifically we replace step 3 of the DST algorithm by the singularly-optimal MST algorithm proposed by Elkin [41]. The other steps of the modified DST algorithm remain the same as that of the original DST algorithm.

The correctness of the modified DST algorithm and its approximation factor, which is $2(1-1/\ell)$, directly follows from the correctness of the algorithm proposed by Kou et al. [84].

**Round complexity.** By Lemma 3.2.2, step 1 of the modified DST algorithm takes $O(S)$

rounds. Step 2 and step 3 of the modified DST algorithm take $O(D)$ and $O((D+\sqrt{n})\log n)$ rounds respectively. Since step 4 also takes $O(S)$ rounds, the overall round complexity of the modified DST algorithm is $O(S+(D+\sqrt{n})\log n)$. We know that $1 \leq D \leq S \leq n-1$. Therefore $O(S+(D+\sqrt{n})\log n) = \tilde{O}(S+\sqrt{n})$. Note that the polylogarithmic factor involved with the round complexity of the modified DST algorithm is at most $\log n$.

**Message complexity.** By Lemma 3.2.3, the message complexity of step 1 of the modified DST algorithm is $O(Sm)$. The message complexities of step 2, step 3, and step 4 are $O(m)$, $O(m\log n + n\log n \cdot \log^* n)$, and $O(n)$ respectively. It is clear that the overall message complexity of the modified DST algorithm is dominated by step 1 and step 3. Combining these two steps we get that the message complexity of the modified DST algorithm is $O(Sm + m\log n + n\log n \cdot \log^* n)$. Note that $\log^* n \leq \log n$, and since the graph $G$ is connected, $m \geq n-1$. Therefore we can write $O(Sm + m\log n + n\log n \cdot \log^* n) = \tilde{O}(Sm)$. The polylogarithmic factor involved with the message complexity is at most $\log^2 n$.

Observe that in case of $S = O(\log n)$, the round and message complexities incurred by the modified DST algorithm are $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m)$ respectively. This implies that for sufficiently small values of the shortest path diameter (where $S = O(\log n)$) the round and message complexities of the modified DST algorithm coincide with the results of the singularly-optimal MST algorithms proposed in [41, 62, 110] and the cost of the resultant ST is at most $2(1-1/\ell)$ of the optimal.

## 3.5 Summary

In this chapter we have proposed DST, a deterministic distributed algorithm for the ST problem in the CONGEST model that computes a $2(1-1/\ell)$-approximate ST with the round and message complexities of $O(S+\sqrt{n}\log^* n)$ and $O(Sm+n^{3/2})$ respectively. The round complexity of the DST algorithm is better than the best known round complexity of the ST construction known so far [90]. We have also investigated the round-message tradeoff in distributed ST construction. Specifically we have proposed a modified DST algorithm which computes a $2(1-1/\ell)$-approximate ST and its round and message complexities are $\tilde{O}(S+\sqrt{n})$ and $\tilde{O}(Sm)$ respectively. The polylogarithmic factors involved with the round and message complexities of the modified DST algorithm are $O(\log n)$ and $O(\log^2 n)$ respectively.

The modified DST algorithm improves the message complexity of the DST algorithm by dropping the additive term of $O(n^{3/2})$ at the expense of a logarithmic multiplicative factor in the round complexity.

# 4 Distributed approximation algorithms for Steiner tree in the CONGESTED CLIQUE

In this chapter we study the Steiner tree (ST) problem in the CONGESTED CLIQUE model (CCM) of distributed computing that was first introduced by Lotker et al. [96]. To the best of our knowledge, this is the first work to study the ST problem in the CCM. The CCM is a special case of the CONGEST model. Note that CONGEST model equally considers both the congestion (by bounding the transmitted message size) and the locality. In distributed computing the issue of locality is that nodes (processors) are restricted in collecting data from others which are at a distance of $x$ hops in $x$ time units. However, CCM takes locality out of the picture (by restricting the hop diameter of the underlying communication network to one) and *solely focuses on congestion*. Specifically, in CCM, nodes can directly communicate with each other via an underlying communication network, which is a *clique*. Communication happens in synchronous rounds and a pair of nodes can exchange $b$ bits in each round. Therefore, in each round, all nodes in the CCM can together exchange $O(bn^2)$ bits, where $n$ is the number of nodes in the communication network. Following the convention we assume that $b = O(\log n)$.

Consider a scenario in an *overlay network* where a number of nodes spread over different parts of the network are running the same distributed algorithm (protocol). These nodes are not necessarily adjacent in the base network but are instead connected via an overlay in which there exist point-to-point communications between each pair of nodes. Such overlay networks can be modelled as CCM. Furthermore, models of processing large-scale graphs namely, the $k$-machine [80] and the MapReduce [32, 66] are also related to the CCM.

There has been a lot of progress in solving various problems in the CCM including minimum spanning tree (MST) [56, 67, 73, 96, 107], facility location [15, 53], shortest paths and distances [24,68,104], sub-graph detection [36], triangle finding [34,36], sorting [89,112], routing [89], and ruling sets [15, 67]. However to the best of our knowledge, till date, the ST problem has not been studied in the CCM. Therefore an intriguing question is:

*"What is the best round complexity that can be achieved in solving the ST problem in the CCM while maintaining an approximation factor of at most 2?"*

In CCM, one can trivially compute a 2-approximate ST using $O(n)$ rounds as follows. One can collect the entire topology at a special node using $O(n)$ rounds. Then one can apply one of the best known centralized ST algorithms [20, 84, 130] (locally) whose approximation factor is at most 2. Finally one can distribute the resultant ST among the nodes using $O(1)$ rounds by applying the deterministic routing scheme of Lenzen [89].

Regarding *lower bounds*, till date, no result is known for the ST problem in the CCM. However, there exist lower bound results for other problems in the CCM, namely *sub-graph detection, MST verification, graph connectivity, connected components* etc. Drucker et al. [37] studied the lower bounds on the round complexity of the sub-graph detection problem in the CCM. They proposed the following lower bound results in the BCCM($b$):[1] (i) for every fixed $p \geq 4$, $K_p$*-sub-graph detection* requires $\Omega(n/b)$ rounds, where $K_p$ is a clique of size $p$, (ii) for every fixed $p \geq 4$, $C_p$*-sub-graph detection* requires $\Omega(ex(n, C_p)/nb)$ rounds, where $C_p$ is a cycle of size $p$ and $ex(n, C_p)$ denotes the *Turán number*, (iii) for any $p, q \geq 1$, $K_{p,q}$*-sub-graph detection* requires $\Omega(\sqrt{n}/b)$ rounds, where $K_{p,q}$ is a complete bipartite graph, and (iv) *triangle detection* requires $\Omega(n/(be^{O(\sqrt{n})}))$ rounds. Patt-Shamir and Perry [111] showed that the deterministic lower bound round complexity for the MST verification problem in the BCCM(1) is $\Omega(\log n)$. Recently Pai and Pemmaraju [108] studied

[1]In literature the CCM is mainly classified into two types namely BROADCAST CONGESTED CLIQUE model (BCCM) and UNICAST CONGESTED CLIQUE model (UCCM) [37]. In BCCM($b$) each node can only broadcast a single $b$-bit message over each of its incident links in each round. On the other hand, the UCCM($b$) allows each node to send possibly a different $b$-bit message over each of its incident links in each round.

the *graph connectivity* lower bounds in the CCM. Specifically they showed that the lower bound round complexity for solving the graph connectivity problem in the $KT_0$ version[1] of the BCCM(1) is $\Omega(\log n)$; this bound holds for both deterministic as well as constant-error randomized Monte Carlo algorithms. This lower bound is also extended to the $KT_1$ version of the BCCM(1) and it is proved that the deterministic lower bound round complexity for solving the *graph connectivity* problem in the $KT_1$ version of the BCCM(1) is $\Omega(\log n)$. Furthermore, it is also shown that $\Omega(\log n)$ is the lower bound round complexity for the *connected component* problem in the $KT_1$ version of the BCCM(1) and this result holds for both deterministic as well as constant-error randomized Monte Carlo algorithms. Note that a lower bound of $\Omega(x)$ in BCCM(1) immediately translates to a lower bound of $\Omega(x/b)$ in BCCM($b$).

In this chapter we propose two non-trivial deterministic distributed approximation algorithms for the ST problem in the CCM: STCCM-A and STCCM-B. They are better than the trivial one in terms of the round complexity. Both the algorithms achieve an approximation factor of $2(1 - 1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST. The STCCM-A algorithm computes a $2(1 - 1/\ell)$-approximate ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. We also propose a deterministic distributed *shortest path forest* (SPF) (it is defined in Section 3.2, Chapter 3) algorithm in the CCM (henceforth it will be denoted by SPF-A algorithm) that computes a SPF using $O(n^{1/3} \log n)$ rounds and $O(n^{7/3} \log n)$ messages. The SPF-A algorithm will be used as a subroutine in the STCCM-A algorithm. The overall performance of the STCCM-A algorithm is summarized in the following theorem.

**Theorem 4.0.1.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists an algorithm that computes an ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages in the CCM with an approximation factor of $2(1 - 1/\ell)$, where $\ell$ is the number of leaf nodes in the optimal ST.*

The STCCM-B algorithm computes a $2(1 - 1/\ell)$-approximate ST using $O(S + \log \log n)$ rounds and $O(Sm + n^2)$ messages. Furthermore, we propose another SPF algorithm in the CCM (henceforth it will be denoted as SPF-B algorithm) that computes a SPF using $O(S)$

[1]$KT_0$ (**K**nowledge **T**ill radius **0**) model. Initially each node knows only its own identity and weights of all the edges incident on it. Nodes do not have initial knowledge of the identifiers of their neighbors and other nodes.
$KT_1$ (**K**nowledge **T**ill radius **1**) model. Each node has initial knowledge of itself and the identifiers of its neighbors. Only the knowledge of the identifiers of neighbors is assumed, not other information such as the incident edges of the neighbors.

rounds and $O(Sm)$ messages. The SPF-B algorithm will be used as a subroutine in the STCCM-B algorithm. We summarize the overall performance of the STCCM-B algorithm in the following theorem.

**Theorem 4.0.2.** *Given a connected undirected weighted graph $G = (V, E, w)$ and a terminal set $Z \subseteq V$, there exists an algorithm that computes an ST using $O(S + \log\log n)$ rounds and $O(Sm + n^2)$ messages in the CCM with an approximation factor of $2(1 - 1/\ell)$, where $S$ is the shortest path diameter of $G$, and $\ell$ is the number of leaf nodes in the optimal ST.*

As a by-product of the above theorem, for networks with constant or sufficiently small shortest path diameter (where $S = O(\log\log n)$) the following corollary holds.

**Corollary 4.0.2.1.** *If $S = O(\log\log n)$ then a $2(1 - 1/\ell)$-approximate ST can be deterministically computed using $O(\log\log n)$ rounds and $\tilde{O}(n^2)$ messages in the CCM.*

Note that in case of $S = O(\log\log n)$, the round and message complexities of the STCCM-B algorithm are $O(\log\log n)$ and $\tilde{O}(n^2)$ respectively, which almost coincide with the best known deterministic results for MST construction in the CCM due to Lotker et al. [96] and the approximation factor of the resultant ST is at most $2(1-1/\ell)$ of the optimal.

**The STCCM-A algorithm vs. the STCCM-B algorithm.** For graphs with $S = \omega(n^{1/3}\log n)$, the STCCM-A algorithm performs better than the STCCM-B algorithm in terms of the round complexity. On the other hand, for graphs with $S = \tilde{o}(n^{1/3})$, the STCCM-B algorithm outperforms the STCCM-A algorithm in terms of the round complexity.

**Organization.** The rest of the chapter is organized as follows. In Section 4.1 we define the system model and notations. The description of the SPF-A algorithm is given in Section 4.2. The description of the STCCM-A algorithm, an illustrative example, and the proof of the STCCM-A algorithm are given in Section 4.3. Section 4.4 contains the description as well as the proof of the STCCM-B algorithm. Section 4.5 contains some concluding remarks.

## 4.1 Model and notations

**System model.** We consider the CCM as specified in [96]. This model consists of a complete network (which is a clique) of $n$ nodes. Each node represents an independent computing entity (processor). Nodes are connected through a point-to-point network of $\binom{n}{2}$

bidirectional communication links. The bandwidth of each of the communication links is bounded by $O(\log n)$ bits.

We assume that each vertex has a unique identity (ID) which can be encoded in $O(\log n)$ bits. We also assume that weight of each edge of $G$ can be encoded in $O(\log n)$ bits. Nodes communicate and coordinate their actions with other nodes by passing messages (of size $O(\log n)$ bits) only. In general, a message contains a constant number of edge weights, node IDs, and other arguments (each of them is polynomially bounded in $n$). Note here that each of the arguments in a message is polynomially bounded in $n$ and therefore polynomially many sums of arguments can be encoded in $O(\log n)$ bits. We assume that nodes and links do not fail. Also, nodes are considered computationally unbounded.

An execution of the system advances in synchronous rounds. In each round, nodes receive messages that were sent to them in the previous round, perform finite local computations, and then sends (possibly different) messages. The time complexity is measured by the number of rounds required until all nodes terminate. The message complexity is measured by the number of messages sent until all nodes terminate.

**Formulation of the ST problem in the CCM**. The input graph $G = (V, E, w)$ is distributed among the nodes in the network via a *vertex partition* [56, 67, 73, 96]. In this partition each vertex of the input graph $G$ is uniquely mapped to a node and the edges of $G$ are naturally mapped to the links between the corresponding nodes. If an edge does not exist in $G$ then the weight of the corresponding edge in the CCM is assumed to be $\infty$. Each node runs the same algorithm for the ST problem. Also each node knows whether the vertex assigned to it is a terminal or not. Regarding output, whenever an algorithm for the ST problem terminates, each node knows whether the vertex assigned to it is in the solution or not and which of its incident edges are in the solution. For termination, each node terminates the algorithm for the ST problem in a finite time.

**Notations.** In this chapter we use some of the terms and notations that are already defined in Chapter 3, Section 3.1. In addition to that, in this chapter, we also use the following notations.

- $tf(v)$ denotes the *terminal flag* of a node $v$.
- $ES(e)$ denotes the state of an edge $e$.

- Let $M_{n\times n}$ denotes a matrix of size $n \times n$, where $n$ is a positive integer. Then $M_{ij}$ denotes the value of an entry located at the $i^{th}$ row and $j^{th}$ column in $M_{n\times n}$.

## 4.2 SPF construction

In this section first we give the outline of an all pairs shortest paths (APSP) algorithm in the CCM due to Censor-Hillel et al. [24]. The resulted APSP will be used as a base in the proposed SPF-A algorithm to compute an SPF in the CCM. In this section we also give a brief description of the SPF-A algorithm. Note that the definition of the SPF is already given in Chapter 3, Section 3.2, Definition 3.2.1.

### 4.2.1 Censor-Hillel et al.'s APSP algorithm in CCM

Censor-Hillel et al. [24] showed that given a weighted graph $G = (V, E, w)$, an APSP of $G$ can be computed via iterated squaring of the weight matrix over the min-plus semiring [46,103]. Let $W$ be the weight matrix of size $n \times n$ for a given graph $G$, where $n = |V|$. Then the distance product which is also known as the min-plus product is defined as follows.

$(W \star W)_{uv} = W^2_{uv} = \min_x (W_{ux} + W_{xv})$

where $u, v, x \in V$. The $n^{th}$ distance product denoted by $W^n$ produces the actual distances in $G$, i.e., $d(u, v) = W^n_{uv}$ for each pair of vertices $u, v \in V$. The distance product $W^n$ can be computed by iteratively squaring $W$ for $\lceil \log n \rceil$ times. The distributed squaring of a weight matrix over the min-plus semiring is similar to the parallel 3D matrix multiplication. The squaring of $W$ over the min-plus semiring involves $n^3$ element wise additions of the form $W_{ux} + W_{xv}$, where $u, v, x \in V$. This can be viewed as a cube of size $n \times n \times n$ where each point in the cube represents an addition operation. If we partition the cube of size $n \times n \times n$ into $n$ sub-cubes of equal sizes then each of them gets $n^{2/3} \times n^{2/3} \times n^{2/3}$ points of the cube. This indicates that each node $v \in V$ needs to perform $n^{2/3} \times n^{2/3} \times n^{2/3}$ addition operations. Note that the weight matrix $W$ has $n^2$ entries corresponding to the edge weights of the input graph.[1] This $n^2$ entries are distributed uniformly among $n$ nodes in such a way that each node $v \in V$ gets two sub-matrices of sizes $n^{2/3} \times n^{2/3}$. This ensures that each node needs to compute the min-plus product of matrix size $n^{2/3} \times n^{2/3}$ only.

Now we briefly describe the APSP algorithm due to Censor-Hillel et al. [24]. Initially

[1]If an edge $(u, v)$ is not in the input graph $G$, then $w_{(u,v)}$ is considered equal to $\infty$.

each node $u \in V$ knows the edge weights $w_{(u,v)}$ for each $v \in V$. Here it is assumed that $n^{1/3}$ is a positive integer, where $n = |V|$. Each node $v \in V$ partitions its input into $n^{1/3}$ blocks, each one has $n^{2/3}$ entries. Now each node $v \in V$ sends each such block to $2n^{1/3}$ nodes in such a way that each node $v \in V$ receives two sub-matrices, each one of size $n^{2/3} \times n^{2/3}$. Since the size of each block is $n^{2/3}$ and there are $2n^{2/3}$ recipients, a total of $2n^{4/3}$ messages are sent by each node. This distribution is performed using $O(n^{1/3})$ rounds and $O(n^{7/3})$ messages by using the deterministic routing scheme of Lenzen [89]. Then each node computes the min-plus product from the two known sub-matrices. After that all the resulting min-plus products are re-distributed among $n$ nodes in such a way that each node $v \in V$ receives $n^{4/3}$ values. The distribution of the min-plus products is also performed using $O(n^{1/3})$ rounds and $O(n^{7/3})$ messages by using the deterministic routing scheme of Lenzen [89]. From the received $n^{4/3}$ entries each node $v \in V$ locally computes the row $v$ of the resulting min-plus product $W^2 = W \star W$. With this the first iteration ends. Considering that each of the node IDs and edge weights can be encoded in $O(\log n)$ bits, the round and message complexities of the this iteration are $O(n^{1/3})$ and $O(n^{7/3})$ respectively. Repetition of the above procedure for $\lceil \log n \rceil$ times guarantees the final output $W^n$ which is the required APSP distances for a given graph $G$. The overall round and message complexities of this algorithm are $O(n^{1/3} \log n)$ and $O(n^{7/3} \log n)$ respectively.

**Routing table construction**. The task of routing table construction concerns computing local tables at all the nodes of a network in which each node $u$, when given a destination node $v$, knows the next hop through which $u$ is connected to $v$ by the distance in the network. Specifically the routing table entry $R[u, v] = x \in V$ is a node such that $(u, x) \in E$ and $x$ lies on a shortest path from $u$ to $v$. Censor-Hillel et al. [24] showed that iterated squaring of the weight matrix over the min-plus semiring can also be used to construct the routing table for each node $v \in V$. The min-plus product $W \star W$ provides a *witness matrix* $Q$ such that if $Q_{uv} = x$, then $(W \star W)_{uv} = W_{ux} + W_{xv}$. Whenever the iterated squaring algorithm computes the min-plus product $W^{2i} = W^i \star W^i$, then using the witness matrix $Q$ the routing table $R$ is updated to $R[u, v] = R[u, Q_{uv}]$ for each $u, v \in V$ with $W^{2i}_{uv} < W^i_{uv}$.

### 4.2.2 SPF-A algorithm

The SPF-A algorithm is used as a subroutine in the proposed STCCM-A algorithm. It consists of two parts: the first part constructs an APSP of the input graph $G$ and the second part constructs the required SPF from the graph formed by the APSP. Specifically, in the first part we apply the APSP algorithm due to Censor-Hillel et al. [24] as described in Subsection 4.2.1. Censor-Hillel et al. showed that in the CCM, the iterated squaring of the weight matrix over the *min-plus semiring* [46, 103] computes an *exact* APSP using $O(n^{1/3} \log n)$ rounds and $O(n^{7/3} \log n)$ messages.[1] One of the fundamental applications of the APSP is the construction of the routing tables in a network. Censor-Hillel et al. also showed that in the CCM the iterated squaring algorithm can be used to construct the routing tables of a network as well.

Now we describe the second part of the SPF-A algorithm and show that it requires $O(1)$ rounds and $O(n-t)$ messages, where $t = |Z|$. After the application of the first part each node in $V$ knows the distances to all other nodes in $V$ and its routing table entries $R$. Therefore, by using the distance information each node $v \in V \setminus Z$ can locally choose the closest terminal as its $s(v)$. Note that there may be more than one terminal with equal distances for a given non-terminal. In this case, the non-terminal chooses the one with the smallest ID among all such terminal nodes. Once a non-terminal $v$ chooses the closest terminal as its source node $s(v)$ by using its own routing table $R$, it can also choose its predecessor (parent) node $\pi(v)$ with respect to $s(v)$. Whenever a non-terminal $v$ sets its $\pi(v)$, it also informs $\pi(v)$ that it has chosen $\pi(v)$ as its parent. It is obvious that to establish the parent-child relationship between a pair of nodes $(\pi(v), v)$, where $s(v) = s(\pi(v))$, it requires $O(1)$ rounds and $O(1)$ messages. In this way each node $v$ in $V \setminus Z$ is connected to exactly one tree rooted at some source node $s(v)$. Here each source node is a terminal node. Therefore exactly $|Z|$ number of shortest path trees are constructed by the above procedure which together form the required SPF. Since the procedure of choosing predecessors can be started in parallel by all nodes in $V \setminus Z$, for all such pair of nodes it requires $O(1)$ rounds and $O(n-t)$ messages. It is clear that the overall round and message complexities of the SPF-A algorithm are dominated by the first part of the algorithm. Therefore the following theorem holds.

**Theorem 4.2.1.** *An SPF can be deterministically computed using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages in the CCM.*

[1]Recently Censor-Hillel et al. [23] showed that in CCM, a $(2+\epsilon)$-approximate APSP can be deterministically computed using $O(\log^2 /\epsilon)$ rounds on unweighted undirected graphs, where $\epsilon > 0$. This also applies for weighted graphs with a further additive $(1+\epsilon)w_{uv}$ error in the approximation for any distance value $d(u, v)$, where $w_{uv}$ is the heaviest edge on the shortest $u - v$ path. This result holds for sparse graphs only. However, here we use the exact APSP algorithm due to Censor-Hillel et al. [24], which has a round complexity of $\tilde{O}(n^{1/3})$ and this result also holds for dense graphs.

The correctness of the SPF-A algorithm directly follows from the correctness of the algorithm proposed by Censor-Hillel et al. [24].

## 4.3 STCCM-A algorithm

The proposed STCCM-A algorithm computes a generalized MST $T_Z$ for a given terminal set $Z \subseteq V$ of the input graph $G = (V, E, w)$ in the CCM, which is essentially the required ST. The definition of the generalized MST can be found in Chapter 3, Subsection 3.3.1. In particular the $T_Z$ realizes an MST ($T_A$) of a complete distance graph $K_Z$ for $Z \subseteq V$ of the input graph $G$. The STCCM-A algorithm is inspired by an algorithm (DST algorithm) proposed in Section 3.3, Chapter 3 of this thesis. The DST algorithm, which consists of four small distributed algorithms, was proposed for the CONGEST model, whereas the STCCM-A algorithm proposed here is for the CCM. The STCCM-A algorithm also consists of four small distributed algorithms (step 1 to step 4). However, except step 2, all other steps in the STCCM-A algorithm are different from the DST algorithm. Step 2 of the STCCM-A algorithm is an adaptation of step 2 of the DST algorithm tailored to work in CCM.

The approximation factor of the proposed STCCM-A algorithm, which is $2(1 - 1/\ell)$, directly follows from the correctness of a centralized algorithm due to Kou et al. [84]

**Lotker et al's MST algorithm in the CCM.** We use the MST algorithm due to Lotker et al. [96] as a subroutine in the proposed STCCM-A algorithm. Here we give a brief description of it. It runs in phases. Each phase takes $O(1)$ rounds. In each phase $k \geq 0$, it maintains a set of clusters $\mathcal{F}^k = \{F_1^k, F_2^k, .....F_p^k\}$, $\bigcup_i F_i^k = V$, where $V$ is the vertex set of the input graph $G$. For the sake of simplicity it is assumed that at the beginning of phase 1, the end of the imaginary phase 0, with the cluster set $\mathcal{F}^0 = \{F_1^0, F_2^0, .....F_n^0\}$ is known, where $F_i^0 = \{v_i\}$ for every $1 \leq i \leq n$. For each cluster $F \in \mathcal{F}^k$, the algorithm selects a spanning subtree $T(F)$. At the beginning of phase $k > 0$, the cluster set $\mathcal{F}^{k-1}$ and the corresponding subtree collection $\mathcal{T}^{k-1} = \{T(F) | F \in \mathcal{F}^{k-1}\}$, including the weights of the edges in those subtrees, are known to every vertex in the graph. Whenever the algorithm terminates each node in $V$ knows all the $n - 1$ edges in the MST of $G$.

The outline of a phase $k > 0$ is as follows. Initially each cluster $C \in \mathcal{F}^{k-1}$ is contracted to a vertex $v_C$. All these contracted vertices together form a smaller logical graph denoted by $\hat{G}$. The operation of each vertex $v_C$ is carried out by a special node called the leader of

$C$ denoted by $l(C)$. Let $N$ be the minimum size cluster in $\mathcal{F}^{k-1}$. At the beginning of phase $k > 0$, $N$ (or more) members of each cluster $C \in \mathcal{F}^{k-1}$ collect $N$ lightest edges connecting their cluster $C$ to other clusters in $\mathcal{F}^{k-1} \setminus \{C\}$. Then each cluster in $\mathcal{F}^{k-1}$ sends these $N$ (or more) lightest edges to a special node $v_0$ (node with the lowest ID) of the graph by appropriately sharing the workload among the nodes of the cluster. Now the node $v_0$ has a partial picture of the logical graph $\hat{G}$, consisting of all the contracted vertices $v_C$ but only some of the edges connecting them, specifically $N$ lightest edges emanating from each node of $\hat{G}$ to $N$ different nodes. On the basis of these received information $v_0$ performs (locally) fragment merging operations. For that the known edges are sorted in a non decreasing order of their weights. Then from the non decreasing ordered edge set, add edges to $\hat{G}$ to merge fragments as long as merging is perfectly safe. The *safety rule* is as follows: It is perfectly safe to continue merging a fragment $F$ in the logical graph $\hat{G}$ as long as the $N$ lightest edges of each vertex $v_C$ in $F$ are not inspected. It is shown that the safety rule allows to grow each of the fragments to contain at least $N + 1$ vertices of $\hat{G}$. This ensures that the size of each of the clusters in the next phase will be at least $N^2$. This is a *quadratic* growth of the clusters. Finally $v_0$ sends out the locally known identities of all the edges that are newly chosen (the number of such edges can be at most $n - 1$) by sending each edge to a different intermediate node, which will broadcast that edge to all other nodes. This requires $O(1)$ rounds.

It is clear that at the end of phase $k > 0$, the size of a cluster is at least $2^{2^{k-1}}$. Whenever the algorithm terminates, there exists only one cluster in the cluster set which is the required MST of $G$, i.e., $|\mathcal{F}^k| = 1$. Therefore $2^{2^{k-1}} = n \Rightarrow k = \log\log n + 1$. This ensures that the algorithm terminates in $O(\log\log n)$ rounds, whereas the message complexity is $O(n^2)$.

### 4.3.1 Outline of the STCCM-A algorithm

**Input.** We assume that there is a special node $r \in V$ available at the start of the algorithm. For correctness we assume that $r$ is the node with the smallest ID in the system. Initially each node $v \in V$ knows its unique ID, whether it is a terminal or not, and the weight $w_e$ of each edge $e \in \delta(v)$. Each node in $V$ maintains a boolean variable named *steiner_flag* whose values can be either *true* or *false*. Initially *steiner_flag*$(v)$ is set to *false* for each $v \in V \setminus Z$, whereas throughout the execution of the algorithm the value of *steiner_flag*$(v)$ is set to *true* for each $v \in Z$. Also each node $v \in V$ initially sets its local variable $ES(e)$ to *basic* for each $e \in \delta(v)$. Recall that $ES(e)$ denotes the state of an edge $e$, which can be either *basic*, *branch*, *tree*, *inter_tree*, or *intra_tree*. Except step 2 of the STCCM-A algorithm, in all other steps the state of an edge is either *basic* or *branch*. In step 2 the state of an edge can be either *tree*, *inter_tree*, or *intra_tree*.

**Output.** Whenever the algorithm terminates, each node outputs the pair $(steiner\_flag, B)$. Here $B \subseteq \delta(v)$. *steiner_flag*$(v)$ = *false* ensures that $v$ does not belong to the final ST; in this case $B = \emptyset$. On the other hand, *steiner_flag*$(v)$ = *true* ensures that $v$ is a part of the final ST; in this case $B \neq \emptyset$ and for each $e \in B$, $ES(e)$ is set to *branch*.

The special node $r$ initiates the algorithm. An ordered execution of the steps (step 1 to step 4) is necessary for the correct working of the STCCM-A algorithm. We assume that $r$ ensures the ordered execution of the steps and initiates step $i+1$ after step $i$ is terminated. The outline of the STCCM-A algorithm is as follows.

Step 1. **SPF construction**. Construct an SPF $G_F = (V, E_F, w)$ for $Z$ in $G$ by applying the SPF-A algorithm described in Subsection 4.2.2. Theorem 4.2.1 ensures that the round and message complexities of this step are $\tilde{O}(n^{1/3})$ and $\tilde{O}(n^{7/3})$ respectively.

Step 2. **Edge Weight modification**. With respect to the SPF $G_F$, each edge $e \in E$ of $G = (V, E, w)$ is classified as any one of the following three types.

(a) *tree* edge: if $e \in E_F$.

(b) *inter_tree* edge: if $e \notin E_F$ and end points are incident in two different trees of $G_F$.

(c) *intra_tree* edge: if $e \notin E_F$ and end points are incident in the same tree of $G_F$.

Now transform $G = (V, E, w)$ into $G_c = (V, E, w_c)$. The cost of each edge $(u, v) \in E$ in $G_c$ (denoted as $w_c(u, v)$) is computed as follows.

(a) $w_c(u, v) = 0$ if $(u, v)$ is a *tree* edge.

(b) $w_c(u, v) = \infty$ if $(u, v)$ is an *intra_tree* edge.

(c) $w_c(u, v) = d(u, s(u)) + w_{(u,v)} + d(v, s(v))$ if $(u, v)$ is an *inter_tree* edge. In this case $w_c(u, v)$ realizes the weight of a path from the source node $s(u)$ to the source node $s(v)$ in $G$ that contains the *inter_tree* edge $(u, v)$. Recall that $d(u, v)$ denotes the (weighted) distance between nodes $u$ and $v$ in $G$.

The classification of the edges of $G$ and the transformation to $G_c$ can be done as follows. Each node $v$ of $G$ sends a message (say $\langle set\_category(v, s(v), d(v, s(v)), \pi(v))\rangle$) on all of its incident edges with respect to the input graph $G$. Let a node $v$ receives $\langle set\_category(u, s(u), d(u, s(u)), \pi(u))\rangle$ on an incident edge $(v, u)$. If $s(v) \neq s(u)$ then $v$ sets $(v, u)$ as an *inter_tree* edge and $w_c(v, u)$ to $d(v, s(v)) + w_{(v,u)} + d(u, s(u))$. On the other hand if $s(v) = s(u)$ then $(v, u)$ can be either a *tree* edge or an *intra_tree* edge: if $v = \pi(u)$ or $\pi(v) = u$ then $v$ sets $(v, u)$ as *tree* edge and $w_c(v, u)$ to 0. Otherwise, $v$ sets $(v, u)$ as *intra_tree* edge and $w_c(v, u)$ to $\infty$.

It is clear that step 2 can be performed using $O(1)$ rounds. Also on each edge of $G$, the message $\langle set\_category\rangle$ is sent exactly twice (once from each end). Therefore, the message complexity of step 2 is $O(m)$.

Step 3. **MST construction**. Construct an MST $T_M$ of $G_c$. This step guarantees that each node $v \in V$ knows which of the edges in $\delta(v)$ are in $T_M$ and for each such edge $e$, it sets $ES(e)$ to *branch*. On the other hand for each $e \in \delta(v)$ which is not in $T_M$, $v$ sets $ES(e)$ to *basic*.

Specifically, for MST construction, we apply the deterministic MST algorithm due to Lotker et al. [96]. To the best of our knowledge, till date, it is the only known deterministic MST algorithm proposed in the CCM. All other existing MST algorithms in the CCM [56, 65, 73, 116] are randomized in nature. Note that the round and message complexities of the algorithm proposed by Lotker et al. are $O(\log \log n)$ and $O(n^2)$ respectively.

Step 4. **Pruning**. Construct a generalized MST $T_Z$ by performing a pruning operation on the MST $T_M$.

For correctness we assume that a node in $Z$ with the smallest ID is the root $r_t$ of the $T_M$. The pruning operation deletes edges from $T_M$ until all leaves are terminal nodes. It is performed as follows. Each $v \in V$ sends its parent information (with respect to the $T_M$ rooted at $r_t$) to all other nodes. This requires $O(1)$ rounds and $O(n^2)$ messages. Now each $v \in V \setminus Z$ locally computes the $T_M$ rooted at $r_t$ from these received parent information. Then each node in $V \setminus Z$ can locally decide whether it should prune itself or not from the $T_M$. Specifically, from the locally known $T_M$ each $v \in V \setminus Z$ finds whether it is an intermediate node in between two or more terminals in the $T_M$ or not. If yes, it does not prune itself from the $T_M$ and sets its *steiner_flag* to *true*. Otherwise, it prunes itself. Whenever a node $v$ prunes itself from the $T_M$, it sets its *steiner_flag* to *false* and for each $e \in \delta(v)$ such that $ES(e) \neq basic$, it sets its $ES(e)$ to *basic*. On each such pruned edge $e$, $v$ asks the other end of $e$ to prune the common edge $e$. Now the edge weights of the resultant ST $T_Z$ are restored to the original edge function $w$. Since each node in $V \setminus Z$ can start the pruning operation in parallel and in the worst-case the number of pruned edges in the network can be at most $n-1$, this step requires $O(1)$ rounds and $O(n)$ messages.

The overall round and message complexities of the pruning step are $O(1)$ and $O(n^2)$ respectively.

The STCCM-A algorithm terminates with the step 4.

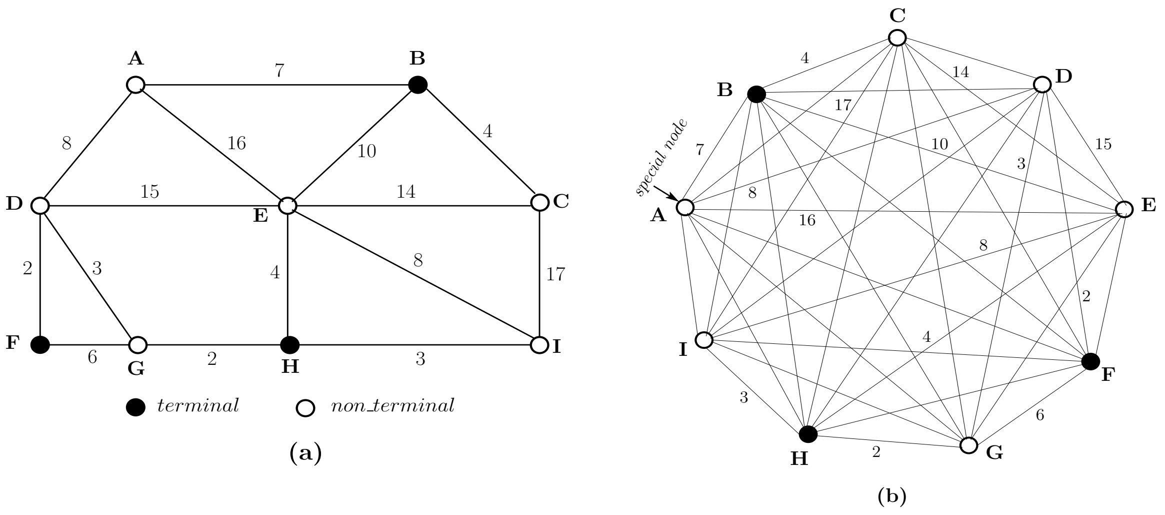
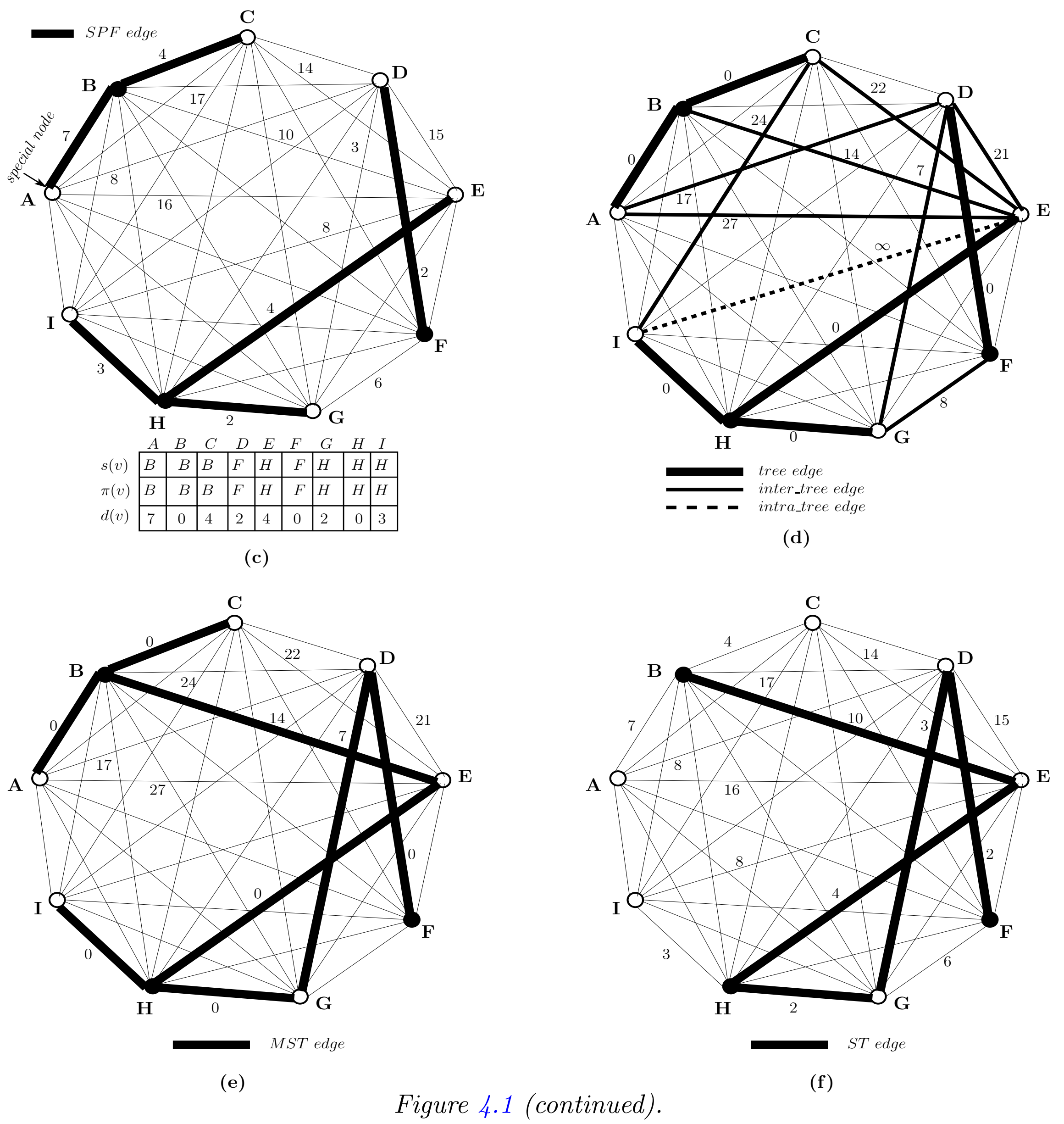


**Figure 4.1:** *(a) An arbitrary graph $G = (V, E, w)$ and a terminal set $Z = \{B, F, H\}$. (b) The input graph $G$ is distributed among the nodes of a complete network $K_9$ via a vertex partition. (c) An SPF $G_F = (V, E_F, w)$ for $Z$ of $G$. The distances of nodes to their respective sources are shown in the table. (d) The modified graph $G_c = (V, E, w_c)$. (e) An MST $T_M$ of $G_c$. (f) The final Steiner tree $T_Z$ (generalized MST) for $Z$ of $G$.*

### 4.3.2 An illustrative example of the STCCM-A algorithm

Consider the application of the STCCM-A algorithm in an arbitrary graph $G = (V, E, w)$ and a terminal set $Z = \{B, F, H\}$ shown in Figure 4.1(a). The input graph $G$ is distributed



|  | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $I$ |
|---|---|---|---|---|---|---|---|---|---|
| $s(v)$ | $B$ | $B$ | $B$ | $F$ | $H$ | $F$ | $H$ | $H$ | $H$ |
| $\pi(v)$ | $B$ | $B$ | $B$ | $F$ | $H$ | $F$ | $H$ | $H$ | $H$ |
| $d(v)$ | 7 | 0 | 4 | 2 | 4 | 0 | 2 | 0 | 3 |

*Figure 4.1 (continued).*

among the nodes of a complete network $K_9$ via vertex partition which is shown in Figure 4.1(b). Specifically each vertex and its incident edges (with weights) of $G$ are assigned to a distinct node in the $K_9$. The weight of an edge in $K_9$ which is not in $G$ is considered equal to $\infty$. An SPF $G_F = (V, E_F, w)$ for $Z$ is constructed which is shown in Figure 4.1(c).

In $G_F$, each non-terminal $v$ is connected to a terminal $s(v) \in Z$ whose distance is minimum to $s(v)$ than any other terminal in $G$ which is shown in the table of Figure 4.1(c). The modified graph $G_c$ and labelling of the edge weights according to the definition of $G_c$ are shown in Figure 4.1(d). Figure 4.1(e) shows after the application of the Lotker et al.'s MST algorithm on $G_c$ which constructs an MST $T_M$ of $G_c$. The final ST $T_Z$ for $Z$ of $G$, which is a generalized MST for $Z$ of $G$ is constructed from the $T_M$ by applying the pruning step of the STCCM-A algorithm, which is shown in Figure 4.1(f).

## 4.3.3 Proof of the STCCM-A algorithm

**Theorem 4.3.1.** *The round complexity of the STCCM-A algorithm is $\tilde{O}(n^{1/3})$.*

*Proof.* It is clear that the overall round complexity of the STCCM-A algorithm is dominated by step 1, which is $\tilde{O}(n^{1/3})$. The polylogarthmic factor involved with this round complexity is $\log n$.

□

**Theorem 4.3.2.** *The message complexity of the STCCM-A algorithm is $\tilde{O}(n^{7/3})$.*

*Proof.* By Theorem 4.2.1 the message complexity of step 1 of the STCCM-A algorithm is $\tilde{O}(n^{7/3})$. Each of step 3 and step 4 requires $O(n^2)$ messages. Step 2 requires $O(m)$ messages. We know that $m \leq n^2$. All of these ensures that the overall message complexity of the STCCM-A algorithm is dominated by step 1. Therefore the message complexity of the STCCM-A algorithm is $\tilde{O}(n^{7/3})$. The polylogarthmic factor involved with this message complexity is $\log n$. □

Note that a path $P_{uv}$ (may contain only one edge) between $u$ and $v$ is called straight if all the intermediate nodes in $P_{uv}$ are in $V \setminus Z$ (the definition can be found in Subsection 3.3.4, Chapter 3). Consider a graph $T^* = (Z, E^*, w^*)$ whose vertex set $Z$ and edge set $E^*$ are defined from $T_M$ as follows. For each *straight path* $P_{uv}$ of $T_M$, let $(a, b)$ be an edge of $E^*$. Then the following lemma holds.

**Lemma 4.3.3.** *$T^*$ is an MST of $K_Z$ for $Z \subseteq V$ of graph $G = (V, E, w)$.*

The correctness of the above lemma directly follows from the correctness of the Lemma 3.3.6, Chapter 3.

It is obvious that if we unfold the tree $T^* = (Z, E^*, w^*)$ then it transforms to a resultant graph $T_Z$, which satisfies the following properties.

- For each straight path between $u$ and $v$ in $T_Z$, there exists an edge $e = (u, v) \in E^*$, where $u, v \in Z$ and the length of the straight path between $u$ and $v$ in $T_Z$ is the shortest one between $u$ and $v$ in $G$.
- All leaves of $T_Z$ are in $Z$.

Therefore the following theorem holds.

**Theorem 4.3.4.** *The tree $T_Z$ computed by the STCCM-A algorithm is a generalized MST for $Z \subseteq V$ of $G = (V, E, w)$.*

Let $T_{opt}$ denotes the optimal ST. Then the following theorem holds.

**Theorem 4.3.5.** $cost(T_Z)/cost(T_{opt}) \leq 2(1 - 1/\ell)$.

The correctness of the above theorem essentially follows from the correctness of the Theorem 1 of Kou et al. [84].

**Deadlock issue.** The STCCM-A algorithm is free from deadlock. Deadlock occurs only if a set of nodes in the system enter into a circular wait state. In step 1 (the SPF construction) of the STCCM-A algorithm, a node uniformly distributes its input (a brief description of the input distribution is given in 4.2.1), which consists of its incident edge weights, among a subset of nodes and never sends resource request message to any other nodes. This ensures that nodes never create circular waiting (or waiting request) for any resource during the execution of the algorithm. In step 2, each node independently sends a message (containing its own state) to all of its neighbors (w.r.t. the input graph $G$) only once. Upon receiving messages from all of its neighbors, a node performs some finite local computations and then terminates itself. Therefore, in step 2 nodes are free from any possible circular waiting. The correctness of the deadlock freeness of step 3 essentially follows from the work of Lotker et al. [96]. In step 4, the pruning operation is performed over a tree structure ($T_M$) and a node never requests and waits for resources holding by any other node in the system. This implies that during the pruning operation, nodes are free from any possible circular waiting. Therefore, the deadlock freeness of all four steps together ensure deadlock freeness of the STCCM-A algorithm.

## 4.4 STCCM-B algorithm

The STCCM-B algorithm is a modified version of the STCCM-A algorithm that computes an ST using $O(S + \log \log n)$ rounds and $O(Sm + n^2)$ messages in the CCM maintaining the same approximation factor as that of the STCCM-A algorithm. Similar to the STCCM-A algorithm, the STCCM-B algorithm also has four steps (small distributed algorithms). Except step 1, all other steps in STCCM-B algorithm are same as that of the STCCM-A algorithm. Recall that step 1 of the STCCM-A algorithm constructs an SPF of a given input graph $G = (V, E, w)$ for a terminal set $Z \subseteq V$. Step 1 of the STCCM-B algorithm, which also constructs an SPF, is an adaptation of the SPF algorithm proposed in Section 3.2, Chapter 3, tailored to work in the CCM. Henceforth the new SPF algorithm will be denoted as SPF-B algorithm.

**The SPF-B algorithm vs. the SPF algorithm.** The SPF-B algorithm is designed for the CCM, whereas the SPF algorithm was proposed for the CONGEST model. The SPF-B algorithm terminates in $S + 4$ rounds, whereas the SPF algorithm terminates in $S + 2h + 1$ rounds, where $h$ is the height of a breadth first search tree of the given input graph $G$. Note here that $h \leq D \leq S$. Both algorithms have the same asymptotic round complexity, which is $O(S)$. The message complexities of the SPF-B algorithm and the SPF algorithm are $O(S(n - t)\Delta + \Delta t + n)$ and $O(Sm)$ respectively, where $\Delta$ is the maximum degree of a vertex in the input graph $G$. Asymptotically both the message complexities are $O(Sm)$.

### 4.4.1 SPF-B algorithm

**Input specification.**

- If $v \in Z$ then $td(v) = 0$, $t\pi(v) = v$, $ts(v) = v$, and $tf(v) = true$.
- If $v \in V \setminus Z$ then $td(v) = \infty$, $t\pi(v) = nill$, $ts(v) = nill$, and $tf(v) = false$.
- Each $v \in V$ sets $ES(e)$ to *basic* for each $e \in \delta(v)$.

Note here that $tf(v)$ denotes the terminal flag of a node $v$. For each $v \in Z$, $tf(v)$ is set to *true*. On the other hand, for each $v \in V \setminus Z$, $tf(v)$ is set to *false*. Recall that $td(v)$, $t\pi(v)$, and $ts(v)$ denotes the tentative distance, tentative predecessor, and tentative source of a node $v$ respectively (these definitions can be found in Section 3.1, Chapter 3). In the SPF-B algorithm, the state of an edge, which is denoted by $ES(e)$, can be either *basic* or *blocked*.

**Output specification.** Whenever the algorithm terminates, $d(v) = td(v)$, $\pi(v) = t\pi(v)$, and $s(v) = ts(v)$ for each node $v \in V$.

**Outline of the SPF-B algorithm.** This algorithm is initiated by a special node $r$ by setting its local variable *start_flag* to *true* and sending a $\langle wakeup \rangle$ message to all the nodes in $V \setminus \{r\}$ and an $\langle echo \rangle$ message to itself. We assume that a node can send messages to itself. Note that upon receiving $\langle echo \rangle$ messages $r$ does nothing. Upon receiving $\langle wakeup \rangle$ message a node awakes itself by setting its *start_flag* to *true*, initializes its local variables as shown in the input specification, and sends an $\langle echo \rangle$ message to $r$. Also each node $v \in Z$ sends $\langle update \rangle$ message on each $e$ in $\delta(v)$ and sets $ES(e)$ to *blocked* for each such $e$. After that the nodes in $Z$ send or receive no $\langle update \rangle$ messages, whereas the nodes in $V \setminus Z$ may send or receive $\langle update \rangle$ messages or send $\langle echo \rangle$ messages to $r$ in each subsequent round until the termination of the algorithm. Upon receiving a set of $\langle update \rangle$ messages (denoted by $U$) a node $v \in V$ acts as per the following rules.

R1. if an $\langle update \rangle \in U$ is received through an incident edge $e = (v, u)$ such that $u \in Z$ then it sets $ES(e)$ to *blocked*.

R2. if $v \in V \setminus Z$ then it computes $w_e + tdn(e)$ for each $\langle update(idn(e), tsn(e), tdn(e), tfn(e)) \rangle \in U$ and chooses the minimum one, say $w_{e'} + tdn(e')$ resulted by $\langle update(idn(e'), tsn(e'), tdn(e'), tfn(e')) \rangle \in U$. If $td(v) > w_{e'} + tdn(e')$ then it updates $td(v) = w_{e'} + tdn(e')$, $ts(v) = tsn(e')$, and $t\pi(v) = idn(e')$. Otherwise, $td(v)$, $ts(v)$ and $t\pi(v)$ remain unchanged.

R3. if the value of $td(v)$ is updated then it sends $\langle update(v, ts(v), td(v), tf(v)) \rangle$ messages on all of its incident *basic* edges and an $\langle echo \rangle$ message to $r$.

*Termination detection.* For this, we introduce two additional messages: $\langle echo \rangle$ and $\langle stop \rangle$. After the first round of the algorithm in which all nodes in the network become awake (after receiving $\langle wakeup \rangle$ messages), the root node $r$ receives at least one $\langle echo \rangle$ message in every subsequent round. Upon receiving $\langle echo \rangle$ messages $r$ does nothing. In the second round of the algorithm $r$ receives an $\langle echo \rangle$ message that is sent by itself. After the second round of

**SPF-B algorithm** at node $v$ upon receiving a set of messages or no message.

```
1: upon receiving no message
2: if $v = r$ then
3:     if $start\_flag = false$ then                              ▷ special node $r$ is sleeping
4:         $start\_flag \leftarrow true$;                         ▷ spontaneous awaken of $r$
5:         send $\langle echo \rangle$ to $r$                     ▷ $r$ sends $\langle echo \rangle$ message to itself
6:         for each $u \in V \setminus \{r\}$ do
7:             send $\langle wakeup \rangle$ to $u$
8:         end for
9:     else                  ▷ $r$ is awake but receives no $\langle echo \rangle$ message, termination state
10:        for each $u \in V$ do
11:            send $\langle stop \rangle$ to $u$
12:        end for
13:    end if
14: end if

15: upon receiving $\langle wakeup \rangle$
16: $start\_flag \leftarrow true$;                                    ▷ awaken of node $v$
17: if $v \in Z$ then
18:     $tf \leftarrow true; ts \leftarrow v; t\pi \leftarrow v; td \leftarrow 0$;
19:     for each $e \in \delta(v)$ do      ▷ $\delta(v)$ is the set of incident edges at $v$ w.r.t. the input
    graph $G$
20:         send $\langle update(v, ts, td, tf) \rangle$ on $e$
21:         $ES(e) \leftarrow blocked$;
22:     end for
23: else
24:     $tf \leftarrow false; ts \leftarrow nill; t\pi \leftarrow nill; td \leftarrow \infty$;
25:     for each $e \in \delta(v)$ do
26:         $ES(e) \leftarrow basic$;
27:     end for
28: end if
29: send $\langle echo \rangle$ to $r$                          ▷ $v$ sends $\langle echo \rangle$ message to $r$

30: upon receiving a set of $\langle update \rangle$ messages     ▷ let $U \neq \phi$ be the set of $\langle update \rangle$
    messages
31: for each $\langle update(idn(e), tsn(e), tdn(e), tfn(e)) \rangle \in U$ such that $e \in \delta(v)$ do
32:     if $tfn(e) = true$ then
33:         $ES(e) \leftarrow blocked$;
34:     end if
35:     if $tdn(e) + w_e < td$ then
36:         $update\_flag \leftarrow true$;              ▷ $update\_flag$ is a temporary boolean variable
37:         $td \leftarrow tdn(e) + w_e; t\pi \leftarrow idn(e); ts \leftarrow tsn(e)$;
38:     end if
39: end for
```

**SPF-B algorithm** (continued).

```
40: if $update\_flag = true$ then
41:     for each $e \in \delta(v)$ such that $ES(e) \neq blocked$ do
42:         send $\langle update(v, ts, td, tf)\rangle$ on $e$
43:     end for
44:     send $\langle echo\rangle$ to $r$
45:     $update\_flag \leftarrow false$;
46: end if

47: upon receiving $\langle echo\rangle$                    ▷ only $r$ receives $\langle echo\rangle$ message
48: do nothing;

49: upon receiving $\langle stop\rangle$
50: $start\_flag \leftarrow false$;                        ▷ $v$ terminates the algorithm
```

the algorithm, in every subsequent round at least one node in the network updates its local state and this guarantees the generation of at least one $\langle$*echo*$\rangle$ message in the network. Note that a node sends its generated $\langle$*echo*$\rangle$ message to $r$. In case no $\langle$*echo*$\rangle$ message is generated, it ensures that no changes have been occurred in the network and as a result $r$ does not receive any $\langle$*echo*$\rangle$ messages. In this case $r$ sends termination instruction to all nodes in the network by sending $\langle$*stop*$\rangle$ messages. Upon receiving $\langle$*stop*$\rangle$ message, each node terminates the algorithm by setting its local variable *start_flag* to *false*. Whenever each node in the network sets its local variable *start_flag* to *false*, the algorithm terminates.

**Lemma 4.4.1.** *The SPF-B algorithm terminates after at most* $S + 4$ *rounds.*

*Proof.* The special node $r$ initiates the SPF-B algorithm by sending $\langle$*wakeup*$\rangle$ messages to all the nodes in $V \setminus \{r\}$ and an $\langle$*echo*$\rangle$ message to itself which takes exactly one round (we assume that a node can send messages to itself). Upon receiving $\langle$*wakeup*$\rangle$ message, each node in $v \in Z$, in parallel, sends $\langle$*update*$\rangle$ messages to each node in $\delta(v)$ and an $\langle$*echo*$\rangle$ message to $r$. This takes one round only. After that all nodes in $V \setminus Z$ proceed in parallel, and if applicable they update their local states. In each subsequent round at least one node $v \in V \setminus Z$ must update its local state which generates $\langle$*update*$\rangle$ and $\langle$*echo*$\rangle$ messages ($v$ sends an $\langle$*echo*$\rangle$ message to $r$ and an $\langle$*update*$\rangle$ message on each of its *basic* edges in $\delta(v)$); otherwise algorithm must have reached the state in which no message is generated in the network. Note that $S$ is the shortest path diameter of the network and therefore any path in the SPF contains no more than $S$ edges. Since all nodes in $V \setminus Z$ update their local states in parallel, each such node $v$ converges to its correct $d(v)$ value in at most $S$ additional rounds. After at most $S+2$ rounds of execution no local changes occur at any node in the network. Therefore after at most $S+2$ rounds $r$ receives no $\langle echo \rangle$ messages. In this state $r$ initiates the termination of the algorithm by sending $\langle stop \rangle$ messages to all the nodes in the network. In the next round, which is $(S+4)^{th}$ round in the worst-case, upon receiving $\langle stop \rangle$ message, each node terminates the algorithm by setting its local variable *start_flag* to *false*. Therefore the SPF-B algorithm terminates after at most $S+4$ rounds. □

**Theorem 4.4.2.** *The round complexity of the SPF-B algorithm is* $O(S)$.

*Proof.* By Lemma 4.4.1 the SPF-B algorithm terminates after at most $S+4$ rounds. Therefore the overall round complexity of the SPF-B algorithm is $O(S)$. □

**Theorem 4.4.3.** *The message complexity of the SPF-B algorithm is* $O(Sm)$.

*Proof.* The special node $r$ initiates the SPF-B algorithm by sending $\langle wakeup \rangle$ messages to all the nodes in $V \setminus \{r\}$ and an $\langle echo \rangle$ message to itself. This generates exactly $n$ messages. Upon receiving $\langle wakeup \rangle$ message each node $v$ in $Z$, in parallel, sends $\langle update \rangle$ messages on each of its edges in $\delta(v)$ and an $\langle echo \rangle$ message to $r$. In the worst-case, this step generates $t(\Delta+1)$ messages, where $\Delta$ is the maximum degree of a node in $G$ and $t = |Z|$. After that no further messages are generated due to the node set $Z$. On the other hand in each subsequent round each node $v$ in $V \setminus Z$ may update its local variables and sends $\langle update \rangle$ message on each of its *basic* edges in $\delta(v)$ and an $\langle echo \rangle$ message to $r$. In the worst-case, each such round can generate $(n-t)(\Delta+1)$ messages. By Lemma 4.4.1, after (at most) $S+2$ rounds no $\langle echo \rangle$ or $\langle update \rangle$ messages are sent or in transit in the network. Therefore, in the worst-case $r$ receives no $\langle echo \rangle$ message in $(S+3)^{th}$ round which guarantees that in this round no message is generated in the network. After that $r$ sends $\langle stop \rangle$ messages to all the nodes in the network. This generates $n$ messages. Since Lemma 4.4.1 guarantees that the algorithm terminates after at most $S+4$ rounds, combining all the rounds, the total number of messages generated, in the worst-case, is $n+t(\Delta+1)+S(n-t)(\Delta+1)+0+n = O(S(n-t)\Delta+\Delta t+n)$. Since $G = (V, E, w)$ is connected, each of the terms $(n-t)\Delta$ and $\Delta t$ is upper bounded by $O(m)$ and $n \leq m$, where $m = |E|$ and we assume that $n > t$. Therefore the overall message complexity of the SPF-B algorithm is $O(Sm)$. □

**Lemma 4.4.4.** *Let $td_i(v)$ be the length of the tentative shortest path from node $v$ to $ts(v)$ after $i$ rounds ($i \geq 0$). Suppose SPF-B algorithm terminates after $X$ rounds ($X \leq S + 4$). Then for each node $v \in V$, $td_X(v) = d(v, s(v))$ and the corresponding shortest path contains at most $S$ edges.*

Note here that $S$ is the shortest path diameter of the network. Therefore any path between a node $v$ and its source $s(v)$ in the SPF contains no more than $S$ edges. The correctness of the Lemma 4.4.4 directly follows from the correctness of the Lemma 3.2.4, Chapter 3.

### 4.4.2 Complexity of the STCCM-B algorithm

Similar to the STCCM-A algorithm, the STCCM-B algorithm also computes a generalized MST for $Z$ of $G$. Therefore the approximation factor of the ST computed by the STCCM-B algorithm is same as that of the STCCM-A algorithm, which is $2(1 - 1/\ell)$.

**Theorem 4.4.5.** *The round complexity of the STCCM-B algorithm is $O(S + \log\log n)$.*

*Proof.* We know that the STCCM-B algorithm consists of four steps. By Theorem 4.4.2, the round complexity of step 1 is $O(S)$. The round complexities of step 2, step 3, and step 4 are $O(1)$, $O(\log\log n)$, and $O(1)$ respectively. It is clear that the overall round complexity of the STCCM-B algorithm is dominated by step 1 and step 3, which is $O(S + \log\log n)$. □

**Theorem 4.4.6.** *The message complexity of the STCCM-B algorithm is $O(Sm + n^2)$.*

*Proof.* By Theorem 4.4.3 the message complexity of step 1 of the STCCM-B algorithm is $O(Sm)$. The message complexities of step 2, step 3, and step 4 of the STCCM-B algorithm are $O(m)$, $O(n^2)$, and $O(n^2)$ respectively. We know that $m \leq n^2$. Therefore the overall message complexity of the STCCM-B algorithm is $O(Sm + n^2)$. □

## 4.5 Summary

In this chapter we have presented two deterministic distributed approximation algorithms for the ST problem in the CCM. The first one computes a $2(1 - 1/\ell)$-approximate ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. The polylogarthmic factor involved with each of the round and message complexities is $\log n$. The second one computes a $2(1-1/\ell)$-approximate

ST using $O(S + \log\log n)$ rounds and $O(Sm + n^2)$ messages. Note that if a graph has the property $S = \omega(n^{1/3} \log n)$, then the first algorithm exhibits a better performance in terms of the round complexity than the second one. On the other hand, for graphs with $S = \tilde{o}(n^{1/3})$, the second algorithm outperforms the first one in terms of the round complexity. To the best of our knowledge, till date, this is the first work to study the ST problem in the CCM.

# 5 Primal-dual based distributed approximation algorithm for prize-collecting Steiner tree

The prize-collecting Steiner tree (PCST) problem is a generalized version of the Steiner tree (ST) problem. The unrooted version of the PCST problem is defined as follows.

**Definition 5.0.1** (PCST problem)**.** *Given a connected weighted graph $G = (V, E, p, w)$ where $V$ is the set of vertices, $E$ is the set of edges, $p : V \to \mathbb{R}^+$ is a vertex prize function and $w : E \to \mathbb{R}^+$ is an edge weight function, the goal is to find a tree $T = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ that minimizes the following quantity:*

$$W(T) = \sum_{e \in E'} w_e + \sum_{v \in V \setminus V'} p_v$$

A feasible solution to the PCST problem partitions the node set $V$ into two parts, namely *Steiner* and *Penalty.* A node is in the Steiner part if it is covered by $T$ (i.e. it belongs to $V'$), otherwise it is in the Penalty part. Note that the ST problem is in fact a special case of PCST, where we set the prize of terminals to $\infty$ and the prize of all other nodes to 0; therefore PCST is NP-hard and at least as hard to approximate as ST.

In this chapter we study the rooted variant of the PCST problem in which a given root node $r \in V$ must be included in the resultant PCST. For correctness we assume that $p_r$ is equal to $\infty$. We present two deterministic distributed algorithms for the PCST problem in the CONGEST model. The first one, which will be denoted as D-PCST algorithm, computes a PCST using $O(n^2)$ rounds and $O(mn)$ messages, where $n = |V|$ and $m = |E|$. The second one, which is a modification of the first one, computes a PCST using $O(Dn)$ rounds and $O(mn)$ messages, where $D$ is the *unweighted diameter* of $G$. Both the algorithms achieve an approximation factor of $\left(2 - \frac{1}{n-1}\right)$.

The D-PCST algorithm is inspired by a sequential algorithm proposed by Goemans and Williamson (GW-algorithm [58]), which is one of the most elegant algorithms for the PCST problem, providing a constant approximation ratio. We make use of the primal-dual technique, appropriately tailored for the distributed setting, in order to construct a PCST with a cost at most $\left(2 - \frac{1}{n-1}\right)$ times the optimal.[1] The D-PCST algorithm uses the idea of preserving dual constraints in a distributed way in order to achieve the desired approximation ratio. The main challenge here is to satisfy the dual constraints using local information instead of global information. To this end we use a careful merging and deactivation of connected components so that each component always satisfies its dual constraints. The detailed description of the distributed preservation of the dual constraints is deferred to Subsection 5.5.1. We believe that our technique is equally applicable in other variants of the ST problems like *Steiner forest*, *directed ST* etc.

Like the D-PCST algorithm, the modified D-PCST algorithm also based on the use of the primal-dual technique. The main difference is that the modified D-PCST algorithm leverages on a breadth first search (BFS) tree of the input graph, whereas the D-PCST algorithm does not leverage on any such precomputed structure. Furthermore, for networks with $D = o(n)$, the modified D-PCST algorithm performs better than the original D-PCST algorithm in terms of the round complexity. We summarize the performance of the modified D-PCST algorithm in the following theorem.

**Theorem 5.0.1.** *Given a connected weighted graph $G = (V, E, p, w)$ where $V$ is the set of vertices, $E$ is the set of edges, $p : V \to \mathbb{R}^+$ is a vertex prize function and $w : E \to \mathbb{R}^+$ is an edge weight function, a $(2 - \frac{1}{n-1})$-approximate PCST can be deterministically computed using $O(Dn)$ rounds and $O(mn)$ messages in the CONGEST model of distributed computing,*

[1]Note here that Geunes et al. [54] showed that any LP-based $\alpha$-approximation algorithm for a *covering* problem can be leveraged to a $(\frac{1}{1-e^{-1/\alpha}})$-approximation algorithm for the corresponding prize-collecting problem, and Li et al. [92] extend this result for sub-modular penalties. However the frameworks used in [4, 54, 92], which are sequential in nature, may not be equally applicable in the distributed setting.

*where $n = |V|$, $m = |E|$, and $D$ is the unweighted diameter of $G$.*

Moreover, both the algorithms have a memory requirement of $O(\Delta \log n)$ bits for each node, where $\Delta$ is the maximum degree of a node in $G$. In contrast, none of the earlier primal-dual based distributed approximation algorithms [42, 59, 60, 102, 109] discuss about the memory efficiency of their proposed solutions. Indeed, one can design a trivial, essentially centralized, distributed algorithm for the PCST problem by 'black-box' use of the GW-algorithm. In such a case there should be a specified node (leader) to gather the entire graph information and compute the PCST solution in a centralized manner. Assuming polynomially bounded edge weights and node prizes, the leader node would need to store $O(m \log n)$ bits in its memory. In contrast, in our approach only local information is stored in each node.

**Lower bound results.** Regarding lower bounds no immediate result is known for the PCST problem. Recently Bacrach et al. [6] came with a lower bound round complexity for exact ST computation in the CONGEST model, which is $\Omega(n^2/log^2 n)$. Since the PCST problem is a generalized version of the ST problem, the exact lower bound round complexity for ST problem, which is $\Omega(n^2/log^2 n)$, also holds for exact PCST computation. However this does not apply for approximate results. Elkin [39] showed that approximating MST within any constant factor on graphs of small unweighted diameter ($D = O(\sqrt{n})$) requires $\Omega(\sqrt{n/B})$ rounds (assuming $B$ bits can be sent through each edge in each round). Das Sarma et al. [31] achieved an unconditional lower bound on round complexity of the MST problem and showed that approximating MST within any constant factor requires $\Omega(D + \sqrt{n/(B \log n)})$ rounds. Note that PCST is a more generalized version of the the MST problem.[1] Therefore, in the approximate sense the above lower bound round complexity of the distributed MST construction also applies to the distributed PCST construction. Kutten et al. [86] established that $\Omega(m)$ is the message lower bound for leader election in the $KT_0$ model (i.e. **K**nowledge **T**ill radius **0**) which holds for both the deterministic as well as randomized (Monte Carlo) algorithms even if the network parameters $D$, $n$, and $m$ are known, and all nodes wake up simultaneously. Since a distributed MST algorithm can be used to elect a leader, the above message lower bound in the $KT_0$ model also applies to the distributed MST construction. Since the PCST is a more generalized version of the

[1]If the prize value of each node in the network is equal to $\infty$ then the PCST problem simply becomes the MST problem.

MST problem the lower bound message complexity for distributed MST construction also holds for the distributed PCST construction in the CONGEST model.

**Prior art in primal-dual based distributed approximation.** Moscibroda and Wattenhofer [102] proposed a primal-dual based distributed algorithm in the CONGEST model that solves the facility location problem using $O(x)$ communication rounds for every constant $x$ with the approximation factor of $O\big(\sqrt{x}(fc)^{1/\sqrt{x}}\log(f+g)\big)$, where $f$ and $g$ are the number of facilities and clients respectively and $c$ is a coefficient that depends on the cost values of the instance. Pandit and Pemmaraju [109] showed that using primal-dual method matric facility location problem can be solved using $O(x)$ rounds with the approximation factor of $O(f^{2/\sqrt{x}} \cdot g^{3/\sqrt{x}})$ of the optimal in the CONGEST model of distributed computing. Grandoni et al. [59, 60] showed that primal-dual technique can be used to compute a $(2+\epsilon)$-approximate capacitated vertex cover using $O(\log(nR)/\epsilon)$ rounds in the CONGEST model, where $\epsilon$ is any constant $> 0$, $n$ is the number of nodes in the network, and $R$ is the ratio of largest to smallest weight. Recently Even et al. [42] presented a primal-dual based deterministic distributed algorithm for minimum set cover problem that computes a $f'(1+\epsilon)$-approximate set cover using $O(\log(f'\Gamma)/\log\log(f'\Gamma))$ rounds, where $f'$ denotes the maximum element frequency and $\Gamma$ denotes the cardinality of the largest set in the given input instance. Furthermore, distributed primal-dual method is also applied in machine learning optimization problems [98, 131]. By employing the distributed average consensus algorithms Yuan et al. [131] proposed a distributed primal-dual subgradient method to provide approximate saddle points of the Lagrangian function. Ma et al. [98] generalized the communication-efficient primal-dual framework for distributed optimization.

**Organization.** The rest of the chapter is organized as follows. In Section 5.1 we define the system model. In Section 5.2 we introduce the formulation of the PCST problem using integer programming (IP) and linear programming (LP). A brief description of the centralized PCST algorithm proposed by Goemans and Williamson [58] and the key challenge to make it distributed are given in Section 5.3. The high level description of the proposed D-PCST algorithm is given in Section 5.4. Section 5.5 contains the proof of properties of the D-PCST algorithm. The modified D-PCST algorithm and its properties are discussed in Section 5.6. Section 5.7 contains the concluding remarks. An illustrating example and the pseudo-code of the D-PCST algorithm are deferred to Appendix 5.A and Appendix 5.B respectively.

## 5.1 Model and notations

**System model.** We consider the CONGEST model as specified in [114]. This model is briefly described as follows. A communication network is modelled as a weighted undirected graph $G = (V, E, p, w)$, where $V$ is the set of nodes, $E$ is the set of links, $p : V \rightarrow \mathbb{R}^+$ is a positive prize function, and $w : E \rightarrow \mathbb{R}^+$ is a positive weight function. We assume that the weight of each edge $e$ (denoted as $w_e$) is polynomially bounded in $n$ and therefore polynomially many sums of weights can be encoded in $O(\log n)$ bits. Similarly the prize of each vertex (denoted as $p_v$) is polynomially bounded in $n$. Each node knows its unique ID (can be represented using $O(\log n)$ bits), and the weight of each link incident to it. Each node executes the same algorithm. The nodes communicate and coordinate their actions with their neighbors by passing messages (of size $O(\log n)$ bits) only. In general a message contains constant number of edge weights, node IDs, and arguments (each polynomially bounded in $n$). We consider that the links deliver messages in FIFO order. A special node of the network designated as *root* $(r)$ initiates the algorithm. We consider that $p_r$ is equal to $\infty$. We assume that nodes and links do not fail.

The algorithm proceeds in synchronous rounds as follows. At the beginning of each round, each node receives all the messages sent to it. After that each node performs some local computation. Then each node sends (possibly different) messages on its incident links, which will be processed in the next round. The *time complexity* is measured by the number of rounds required until all the nodes terminate. The *message complexity* is measured by the number of messages generated until all the nodes terminate. The *memory complexity* (aka *space complexity*) is measured as the amount of working memory (or Turing machine tape space) required by a node (computing entity) in order to store messages, sent or received, and all information needed to perform local computations. We do not count the amount of memory required to store the input data (node's ID and prize value, and weights of incident edges).

**Notations.** Most of the terms and notions we use in this chapter are already defined in Section 3.1, Chapter 3 and Section 4.1, Chapter 4. In addition to that, in this chapter, we also use the following notations.

- A set of nodes $C \subseteq V$ connected by a set of edges is termed as *component*.
- $L(C)$ denotes the leader node of a component $C$.

- A component $C'$ is called a *neighboring* component of a component $C$ if $\delta(C) \cap \delta(C') \neq \emptyset$.
- $CS(C)$ denotes the state of a component $C$.
- $W(C)$ denotes the weight of a component $C$.
- $TP(C)$ denotes the total prize of a component $C$.
- $EPM(e)$ denotes *edge for prune message* which is a boolean variable for an edge $e$. At node $v \in V$ if $EPM(e)$ is set to *true* for an edge $e \in \delta(v)$ then $v$ chooses edge $e$ to send a message called *prune* on it; on the other hand, if at node $v$, $EPM(e)$ is set to *false* for an edge $e \in \delta(v)$ then $v$ never chooses $e$ to send a prune message on it.

## 5.2 Problem formulation

Following [58] the rooted PCST problem can be formulated as the following integer program (IP).

$$\text{Min} \quad \sum_{e \in E} w_e x_e + \sum_{U \subset V; r \notin U} z_U \Big( \sum_{v \in U} p_v \Big)$$

Subject to :

$$\begin{aligned} x(\delta(C)) + \sum_{U \supseteq C} z_U \geq 1 & \qquad C \subset V; r \notin C \\ x_e \in \{0,1\} & \qquad e \in E \\ z_U \in \{0,1\} & \qquad U \subset V; r \notin U \end{aligned}$$

For each edge $e \in E$ there is a variable $x_e$ that takes a value in $\{0,1\}$. Here $x(\delta(C)) = \sum_{e \in \delta(C)} x_e$. For each $U \subset V : r \notin U$, there is a variable $z_U$ that takes values from $\{0,1\}$. A tree $T = (V', E')$ rooted at the root node $r$ corresponds to the following integral solution of the IP: $x_e = 1$ for each $e \in E'$, $z_{V \setminus V'} = 1$ and all other variables are zero. The integral constraint says that a subset of nodes $C \subset V$ ($r \notin C$) is connected to $T$ if there exists at least one $e \in \delta(C)$ such that $x_e = 1$ or it is not connected to $T$ if $C \subseteq U \subset V$ ($r \notin U$), $x_e = 0$ for all $e \in \delta(C)$ and $z_U = 1$. Note that we can set $p_r = \infty$ since every feasible tree is required to include the root node $r$.

The LP-relaxation of the above IP can be created by replacing $x_e \in \{0,1\}$ and $z_U \in \{0,1\}$ by $x_e \geq 0$ and $z_U \geq 0$ respectively.

The dual of the LP-relaxation is as follows:

$$\text{Max} \sum_{C \subseteq V-\{r\}} y_C$$

Subject to :

$$\sum_{C: e \in \delta(C)} y_C \leq w_e \qquad e \in E$$

$$\sum_{C \subseteq U} y_C \leq \sum_{v \in U} p_v \qquad U \subset V; r \notin U$$

$$y_C \geq 0 \qquad C \subset V; r \notin C$$

Here the variable $y_C$ corresponds to the primal constraint $\sum_{e \in \delta(C)} x_e + \sum_{U \supseteq C} z_U \geq 1$. The dual objective function indicates that for each $C \subseteq V \setminus \{r\}$, the variable $y_C$ can be increased as much as possible without violating the two dual constraints $\sum_{C: e \in \delta(C)} y_C \leq w_e$ and $\sum_{C \subseteq U} y_C \leq \sum_{v \in U} p_v$. The constraint $\sum_{C: e \in \delta(C)} y_C \leq w_e$ is known as *edge packing* constraint which is corresponding to the primal variable $x_e$. It says that for each $C \subseteq V \setminus \{r\}$ such that $e \in \delta(C)$, $y_C$ can be increased as much as possible until the *edge packing* constraint becomes tight, i.e. $\sum_{C: e \in \delta(C)} y_C = w_e$. This equality implies the case where the primal variable $x_e = 1$ for the corresponding edge $e$, and $e$ is added to the forest being constructed. The value $w_e$ contributes to the primal objective value of the PCST. The dual constraint $\sum_{C \subseteq U} y_C \leq \sum_{v \in U} p_v$ is known as *penalty packing* constraints which is corresponding to the primal variable $z_U$ of the LP relaxation. For each $C \subseteq U$ such that $r \notin U$, $y_C$ can be increased as much as possible until the penalty packing constraint becomes tight i.e. $\sum_{C \subseteq U} y_C = \sum_{v \in U} p_v$. Any positive value of $y_C$ can be considered feasible provided it does not lead to the violation of any of the two dual packing constraints. If we set $y_C = 0$ for each $C \subseteq V \setminus r$ then it gives a trivial feasible solution to the dual LP since it satisfies both packing constraints. The dual LP is feasible at its origin ($y_C = 0$ for each $C \subseteq V \setminus r$), whereas primal LP is not feasible at its origin ($x_e = 0$ for each $e \in E$ and $z_U = 0$ for each $U \subseteq V \setminus r$).

## 5.3 A brief description of the GW-algorithm

The D-PCST algorithm is inspired by the sequential PCST algorithm due to Goemans and Williamson (GW-algorithm) [58]. Note that apart from the GW-algorithm there exist other sequential algorithms for PCST [16, 44, 72] with slightly better approximation factor than that of the GW-algorithm. Specifically, the sequential PCST algorithm proposed in [4] achieves an approximation ratio of 1.9672, which is the best known approximation ratio for PCST till date. However we still choose the GW-algorithm as the starting point to adapt it to the distributed setting because it is a widely studied fundamental algorithm for PCST in the sequential setting as well as the basis for all other algorithms proposed in [4, 16, 44, 72].

Now we briefly describe the GW-algorithm. It consists of two phases namely *growth* and *pruning*. The growth phase maintains a forest $F$ which contains a set of candidate edges being selected for the construction of the PCST. Initially $F$ is empty, each node is unmarked and is considered as a connected component containing a singleton node. The growth phase also maintains a set of components whose possible states can be either *active* or *inactive*. The active state of a component indicates that it is in a growing phase, whereas the inactive state indicates that it stops growing tentatively. If a component $C$ is active then the current state of $C$ is set to 1, i.e., $CS(C) = 1$, otherwise $CS(C) = 0$. The state of the component containing the root node $r$ is always inactive. Initially, except the root component, all other components are in active state. Associated with each component $C$, there is a dual variable $y_C$, initialized to 0. The algorithm also maintains a deficit value $d_v$ for each vertex $v \in V$ and a weight $W(C)$ for each component $C$. Note that $W(C) = \sum_{A \subseteq C} y_A$ and $d_u = \sum_{A:u\in A} y_A$ for each vertex $u$. Initially $d_u = 0$ for each vertex $u \in V$ and $W(C) = 0$ for each component $C$ in the system. In each iteration the algorithm finds an edge $e = (u, v)$ with $u \in C_p$, $v \in C_q$, $C_p \neq C_q$, that minimizes $\epsilon_1 = \frac{w_e - d_v - d_u}{CS(C_p)+CS(C_q)}$ and a component $C$ such that $CS(C) = 1$ which minimizes $\epsilon_2 = \sum_{v\in C} p_v - W(C)$. And then it finds the global minimum $\epsilon = min(\epsilon_1, \epsilon_2)$. If $\epsilon = \epsilon_1$ then it merges two distinct components $C_p$ and $C_q$ using the edge $e$ (that gave the min $\epsilon$) and adds $e$ to $F$. On the other hand if $\epsilon = \epsilon_2$ then the corresponding component $C$ is deactivated. Note that for every decided value of $\epsilon$, for each active component $C$, $W(C)$ (as well as that of the implicit $y_C$) and each $d_v$, where $v \in C$ are increased by the value of $\epsilon$. In case of merging, if the resulting component contains $r$ then it becomes inactive; otherwise it remains active. In the other case i.e. deactivation of component $C$, the algorithm marks each $v \in C$ with the name of the component $C$. Since in each iteration total number of components or the number of active components decreases therefore after at most $2n - 1$ iterations all components become inactive. In pruning phase the algorithm removes as many edges as possible from $F$ without violating the two properties: (i) all unmarked vertices must be connected to the root, as these vertices never appeared in any deactivated components (ii) if a vertex with mark $C$ is connected to the root then every vertex marked with $C' \supseteq C$

should be connected to the root. The algorithm achieves an approximation ratio of $(2 - \frac{1}{n-1})$ and its running time is $O(n^2 \log n)$ for a graph of $n$ vertices.

**The key challenge to make GW-algorithm distributed.** The GW-algorithm is sequential in nature and proceeds in iterations. In each iteration a global minimum value of a special parameter (denoted as $\epsilon$) is computed and it is used to simultaneously raise the dual variables of all the active components. However, in the distributed setting a node (or a component) has limited knowledge of the network structure and also about the components residing in different parts of the network; as a result simultaneous raising of dual variables of all the active components is a challenging task. To overcome this, in this chapter we present an approach which does not depend on any global knowledge of the network and instead nodes (or components) preserve their dual constraints and update their other state variables by using only the local information. The local preservation of dual constraints at each node of the network eventually guarantees the global preservation of dual constraints. The detailed discussion on the distributed preservation of dual constraints is deferred to Subsection 5.5.1.

## 5.4 D-PCST algorithm

This section contains a description of the D-PCST algorithm whose pseudo-code can be found in Appendix 5.B.

**Preliminaries.** D-PCST algorithm maintains a set of components and each component $C$ has a state (denoted by $CS(C)$) which can be *sleeping*, *active* or *inactive*. Similarly each edge $e$ has a state (denoted by $ES(e)$) which can be *basic*, *branch*, *rejected* or *refind*. It is possible for the edge states at the two end points of the edge to be temporarily different. At node $v$ the state of an edge $e \in \delta(v)$ is *branch* if $e$ is selected as a candidate branch edge. Any edge inside a component (between two nodes $u, v \in C$) whose state is not *branch* is termed as *rejected*. If a node $v$ receives a message *refind_epsilon* on some $e \in \delta(v)$ then $ES(e) =$ *refind*. This message aims at recomputing a special value of the component where $v$ belongs. An edge $e$ which is neither *branch* nor *rejected* nor *refind* has the state named *basic*. Each component has a leader node which coordinates all the activities inside the component. Note that the set of branch edges inside a component $C$ form a tree rooted at the leader. Each node $v \in C$ locally knows the current state as well as the weight of its component $C$. Initially $W(C)$ is equal to 0 for each component $C$. Each node $v$ has a *deficit* value $d_v$, initially equal to 0, and is updated by the algorithm following some rules given in the description of the algorithm. For each node $v \in V$ it can be shown by induction that $d_v = \Sigma_{C:v \in C} y_C$ for $y_C \geq 0$ and this invariant holds throughout the execution of the algorithm. In addition, the following symbols and terms are used in the description of our algorithm.

- $\epsilon_e \in (-\infty, +\infty]$ denotes the *estimated remaining cost* of an edge $e$ (in fact, apart from the cases $\epsilon_e = -\infty$ and $\epsilon_e = +\infty$, the value of $|\epsilon_e|$ is polynomially bounded in $n$). The following cases are possible.
  - (i) $\epsilon_e \in [0, \infty)$. It indicates that the estimated cost of edge $e$ is feasible.
  - (ii) $\epsilon_e \in (-\infty, 0)$. It indicates that the remaining cost of edge $e$ is overestimated by an amount of $|\epsilon_e|$.
  - (iii) $\epsilon_e = \infty$. It indicates that the estimated cost of edge $e$ is not feasible.
- $\epsilon_1(v) = \min\limits_{e \in \delta(v) \cap \delta(C)} \{\epsilon_e\}$. If $\delta(v) \cap \delta(C) = \emptyset$ then $\epsilon_1(v) = \infty$.
- $\epsilon_1(C) = \min\limits_{v \in C} \{\epsilon_1(v)\}$
- $\epsilon_2(C) = \sum_{v \in C} p_v - W(C)$.
- *MEOE*$(v)$ denotes the *minimum epsilon outgoing edge* incident on node $v$ that gives $\epsilon_1(v)$. Similarly *MEOE*$(C)$ denotes the edge $e \in \delta(C)$ that gives $\epsilon_1(C)$.

**Input specification.** Initially each node $v \in V$ knows its own prize value $p_v$, unique identity, and weight $w_e$ of each edge $e \in \delta(v)$. Before the start of the algorithm, *prize_flag* is set to *true* for all $v \neq r$. If $v = r$ then *prize_flag* is set to *false*. Also each node $v \in V$ initially sets its local boolean variable *labelled_flag* to *false*, and *EPM*$(e)$ to *false* and *ES*$(e)$ to *basic* for each $e \in \delta(v)$.

**Output specification.** Whenever the algorithm terminates, each node $v$ outputs the pair (*prize_flag*, $Y$). Here $Y \subseteq \delta(v)$. If *prize_flag* is equal to *true* then $Y = \emptyset$; this ensures that $v$ belongs to the Penalty part. Otherwise $v$ belongs to the Steiner part and for each $e \in Y$, *ES*$(e)$ is set to *branch*.

### 5.4.1 Outline of the D-PCST algorithm

It consists of two phases namely *growth* and *pruning*. At the beginning of the algorithm each component consists of a single node. Initially each component except the root component (containing the special node $r$, the root) is in *sleeping* state. The initial state of the root component is *inactive*. The root node $r$ initiates the algorithm. At any point of the time algorithm maintains a set of components.

At a time only one component grows. The leader of a component $C$ (denoted as $L(C)$) initiates an operation called *proc_initiate*(). The *proc_initiate*() computes $\epsilon_1(C)$ by using a broadcast and convergecast procedure. If $C$ is an active component then the $L(C)$) also calculates $\epsilon_2(C)$. Now $L(C)$) computes $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C))$. After that any one of the following operations is performed.

(i) *Merging* if $C$ is active and $\epsilon(C) = \epsilon_1(C)$. In this case $C$ merges with a neighboring component $C'$ through an edge $e$, where $\epsilon_e = \epsilon_1(C)$ and $e \in \delta(C) \cap \delta(C')$.

(ii) *Deactivation* if $\epsilon(C) = \epsilon_2(C)$. In this case $C$ decides to deactivate itself.

(iii) *Proceed* if $C$ is inactive and $\epsilon(C) = \epsilon_1(C)$. In this case a $\langle$*proceed*$\rangle$ message is sent to a neighboring component $C'$ through an edge $e$, where $\epsilon_e = \epsilon_1(C)$ and $e \in \delta(C) \cap \delta(C')$.

(iv) *Back* if $C$ is inactive and all of its neighboring components become inactive. In this case the leader of $C$ sends a $\langle$*back*$\rangle$ message to an inactive neighboring component $C'$ from which it received the earliest $\langle$*proceed*$\rangle$ message.

Whenever all components in the network, in which one is the root component, become inactive, the growth phase terminates and the pruning phase begins. The pruning phase does the following.

(i) prunes all the non-root components.

(ii) keeps on pruning subcomponents from the root component until no more improvement of the PCST is possible.

Whenever the algorithm terminates, the root component is the required Steiner part whereas the set of nodes which are not included in the root component contributes to the Penalty part of the resultant PCST.

### 5.4.2 Phases of D-PCST

**Growth phase**. At any point of time of this phase only one component $C$ calculates its $\epsilon(C)$. The leader $L(C)$ computes $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C))$ using message passing. Depending on the value of $\epsilon(C)$, $L(C)$ proceeds with any one of the following actions.

(i) If $CS(C) =$ *active* then it may decide to merge with one of its neighboring component $C'$ or it may decide to become inactive.

(ii) If $CS(C) =$ *inactive* then it asks one of its neighboring components, say $C'$, to proceed further. The choice of $C'$ depends on the value $\epsilon_1(C)$ computed at $C$. Note that an inactive component $C$ never computes $\epsilon_2(C)$ and its $\epsilon(C)$ is equal to $\epsilon_1(C)$.

To compute $\epsilon_1(C)$, the leader $L(C)$ broadcasts $\langle$*initiate*$\rangle$ over the tree rooted at $L(C)$ asking each *frontier* node $v \in C$ to find its $\epsilon_1(v)$. A node $v \in C$ is called a *frontier* node if it has at least one edge $e \in \delta(v) \cap \delta(C)$. Upon receiving $\langle$*initiate*$\rangle$, each frontier node $v \in C$ calculates $\epsilon_e$ for each edge $e \in \delta(v) \cap \delta(C)$. Note that if an edge $e$ satisfies the condition $e \in \delta(v) \cap \delta(C)$ then the state of the edge $e$ at node $v$ is either *basic* or *refind*. Let $C'$ be a neighboring component of $C$ such that $e \in \delta(u)$, $e \in \delta(v)$, $u \in C'$, and $v \in C$. Now $v$ calculates $\epsilon_e$ as follows.

(i) $CS(C) =$ *active* and $CS(C') =$ *active* : in this case $\epsilon_e = \frac{w_e - d_v - d_u}{2}$.

(ii) $CS(C) =$ *active* and $CS(C') =$ *inactive* : in this case $\epsilon_e = w_e - d_v - d_u$.

(iii) $CS(C) =$ *active* and $CS(C') =$ *sleeping* : in this case $\epsilon_e = \frac{w_e - d_v - d_u}{2}$. Here the state of the component $C'$ is *sleeping* and therefore the deficit value $d_u$ of the node $u \in C'$ is considered to be equal to $d_v$.

(iv) $CS(C) =$ *inactive* and $CS(C') =$ *sleeping* : in this case $\epsilon_e = w_e - d_v - d_u$. Similar to the previous case the deficit value $d_u$ is equal to $d_v$.

(v) $CS(C) =$ *inactive* and $CS(C') =$ *inactive* : in this case the value of $\epsilon_e$ for an edge $e \in \delta(v)$ calculated by $v$ depends on the state of the edge $e$. If $ES(e) =$ *refind* then $\epsilon_e = w_e - d_v - d_u$. Otherwise, $\epsilon_e = \infty$.

The algorithm also maintains the following two properties:

*Property 1.* Whenever an inactive component $C$ is in the state of computing its $\epsilon_1(C)$ then there can not exist any component $C'$ in the neighborhood of $C$ such that $CS(C') = active$.

*Property 2.* A component $C$ never computes its $\epsilon_1(C)$ (or $\epsilon_2(C)$) while it is in the *sleeping* state.

Following these conditions each frontier node $v \in C$ ($C$ is essentially a tree formed by the branch edges rooted at the leader node) locally computes the value of $\epsilon_e$ for each edge $e$ of its incident *basic* or *refind* edges and among all of them the minimum one is selected as its $\epsilon_1(v)$. Note that if none of the edges of $\delta(v)$ is neither *basic* nor *refind* then $\epsilon_1(v)$ is equal to $\infty$. If $v$ is a leaf node then it reports its $\epsilon_1(v)$ to its parent using $\langle report \rangle$. On the other hand if $v$ is an intermediate node then it waits to arrive $\epsilon_1(u)$ from each of its children $u$. Among all of these values (its own computed value $\epsilon_1(v)$ and all of the received values from its children) an intermediate node $v$ locally selects the minimum one and reports it to its parent using $\langle report \rangle$. During this convergecast process overall $\epsilon_1(C)$ survives and eventually reaches $L(C)$. Also during the convergecast process each node $v \in C$ reports the total prize value of all the nodes in the subtree rooted at $v$. Therefore, eventually the total prize of the component $C$ (denoted as $TP(C)$) is also known to $L(C)$. If $CS(C) = active$ then $L(C)$ also calculates $\epsilon_2(C) = \sum_{v \in C} p_v - W(C) = TP(C) - W(C)$.

The $L(C)$ now computes $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C))$. If $\epsilon(C) = \epsilon_2(C)$ then $C$ decides to deactivate itself. This indicates that the dual penalty packing constraint $\sum_{A \subseteq C} y_A \leq \sum_{v \in C} p_v$ becomes tight for the component $C$. On the other hand, if $\epsilon(C) = \epsilon_1(C)$ then it indicates that the dual edge packing constraint $\sum_{A: e \in \delta(A)} y_A \leq w_e$ becomes tight for the MEOE $e$ such that $e \in \delta(C) \cap \delta(C')$, where $C'$ is a neighboring component of $C$. In this case $L(C)$ sends a $\langle merge(\epsilon(C)) \rangle$ to a frontier node $v \in C$ which resulted the $\epsilon(C) = \epsilon_1(C) = \epsilon_1(v)$. Upon receiving $\langle merge(\epsilon(C)) \rangle$ the node $v$ sends $\langle connect(v, W(C), d_v) \rangle$ over the MEOE to $C'$ to merge with it. Whenever a node $u \in C'$ receives $\langle connect(v, W(C), d_v) \rangle$ on an edge $e \in \delta(u)$ then depending on the state of $C'$ following actions are taken.

(i) $CS(C') = inactive$ : in this case the node $u \in C'$ sends $\langle accept \rangle$ to $v \in C$. This confirms the merging of two components $C'$ and $C$.

(ii) $CS(C') = sleeping$ : in this case it is obvious that $C'$ is a single node component $\{u\}$. The state of $C'$ becomes *active* and each of its local variables $d_u$ and $W(C')$ is initialized to $d_v$ (received in the $\langle connect \rangle$). After that the leader of the component

$C'$ ($u$ itself) computes $\epsilon_e$ for edge $e$ and $\epsilon_2(C')$. If $\epsilon_e < \epsilon_2(C')$ then the component $C'$ sends ⟨*accept*⟩ to the component $C$ which confirms the merging of two components $C'$ and $C$. On the other hand if $\epsilon_2(C') \leq \epsilon_e$ then $C'$ decides to deactivate itself and sends $\langle refind\_epsilon \rangle$ to the component $C$.

Whenever a node $v \in C$ receives ⟨*refind_epsilon*⟩ in response to a ⟨*connect*⟩ on an edge $e \in \delta(v)$ then the state of its local variable *ES*$(e)$ becomes *refind*. The node $v$ also reports the ⟨*refind_epsilon*⟩ to the leader of $C$. Upon receiving ⟨*refind_epsilon*⟩ the leader node of $C$ proceeds to calculate its $\epsilon(C)$ once again.

Whenever a component $C$ decides to merge or deactivate (only an active component can decide to deactivate itself) then each node $v \in C$ updates its $d_v$ and $W(C)$ to $d_v + \epsilon(C)$ and $W(C) + \epsilon(C)$ respectively. Note that for each component $C$ there is an implicit dual variable $y_C$ which we want to maximize subject to dual constraints. Whenever the local variables of a component $C$ are updated by $\epsilon(C)$, $y_C$ is also implicitly updated. Also note that at any point of the time of execution of the algorithm $y_C \geq 0$ for any component $C$ in the graph.

If two components $C$ and $C'$ decide to merge through an edge $e = (v, u)$ such that $e \in \delta(C) \cap \delta(C'))$ then the dual *edge packing* constraint $\sum_{A:e\in\delta(A)} y_A \leq w_e$ becomes tight for the edge $e$. Both nodes $v$ and $u$ set their local variables *ES*$(e)$ to *branch*. The weight of the resulting component $C \cup C'$ is the sum of the weights of $C$ and $C'$, i.e. $W(C \cup C') = W(C) + W(C')$. If $C \cup C'$ contains the root node $r$ then it becomes *inactive* (root component is always *inactive*) and $r$ remains the leader of the new component $C \cup C'$. In addition, whenever a component $C'$ merges with the root component then each $v \in C'$ sets its local variable *prize_flag* to *false* and there exists at least one edge $e \in \delta(v)$ such that *ES*$(e)$ is set to *branch*. On the other hand if none of the merging components $C$ or $C'$ is the root component then the resulting component $C \cup C'$ becomes *active*. In this case the node with the higher ID between the two adjacent nodes of the merging edge becomes the new leader of $C \cup C'$ and for each node $v \in C \cup C'$ the boolean variable *prize_flag* remains *true*.

In case of deactivation of a component $C$, each node $v \in C$ sets its *labelled_flag* to *true*. Whenever an active component $C$ becomes inactive and there exists no active component in its neighborhood then the leader of $C$ decides to send either ⟨*proceed*⟩ or ⟨*back*⟩ to one of its neighboring component $C'$. For this, first of all the leader of $C$ computes its $\epsilon(C)$ (recall that an inactive component $C$ never computes its $\epsilon_2(C)$ and its $\epsilon(C)$ is equal to $\epsilon_1(C)$). The value of $\epsilon(C)$ may be some finite real number or $\infty$. If $C$ has at least one neighboring component $C'$ such that $CS(C') = \mathit{sleeping}$; or $CS(C') = \mathit{inactive}$ for each neighboring component $C'$ of $C$ and $\exists e : e \in \delta(C) \wedge ES(e) = \mathit{refind}$ then the value of $\epsilon(C)$ is guaranteed to be a finite real number. Otherwise, the value of $\epsilon(C)$ is equal to $\infty$. If $\epsilon(C)$ is a finite real number then the leader $L(C) \in C$ sends $\langle \mathit{proceed} \rangle$ on $(L(C), u)$ (to node $u$ from which it receives $\epsilon(C)$). Whenever a node $v$ sends a $\langle \mathit{proceed} \rangle$ message on $e \in \delta(v)$ then it sets its local variable $EPM(e)$ to *true*. The $\langle \mathit{proceed} \rangle$ message propagate along the path in such a way that finally it reaches the node $v \in C$, where $\epsilon(C) = \epsilon_1(v)$. Then $v$ sends $\langle \mathit{proceed} \rangle$ over the MEOE $e \in \delta(C) \cap \delta(C')$. Upon receiving $\langle \mathit{proceed} \rangle$, the component $C'$ starts computing its $\epsilon(C')$ for taking further actions. If the value of $\epsilon(C)$ is equal to $\infty$ then the leader of the component $C$ sends $\langle \mathit{back} \rangle$ to a neighboring component $C''$ from which it received a $\langle \mathit{proceed} \rangle$ in some early stages of the algorithm. Note that as component grows by merging with different components it is possible to receive more than one $\langle \mathit{proceed} \rangle$ message over time by a component $C$; however, a node $v$ in a component $C$ receives at most one $\langle \mathit{proceed} \rangle$ message at a time. In this case the leader of $C$ sends $\langle \mathit{back} \rangle$ message to the *frontier* node which received the earliest $\langle \mathit{proceed} \rangle$ message in $C$. Eventually whenever the leader of the root component $C_r$ finds that the value of $\epsilon(C_r)$ is equal to $\infty$ then it indicates that all components in the network become *inactive*. This ensures the termination of the *growth* phase. After the termination of the *growth* phase, the root node initiates the *pruning* phase.

**Pruning phase**. In this phase following operations are performed.

- Each node $v \in C$, where $C$ is a non-root component, sets its $ES(e)$ to *basic* for each edge $e \in \delta(v)$ if $ES(e) \neq \mathit{basic}$.

- In the root component $C_r$, pruning starts in parallel at each leaf node of the tree rooted at the root node $r$ and is repeatedly applied at every leaf node $v \in C_r$ at any stage as long as the following two conditions hold.

  – $\mathit{labelled\_flag} = \mathit{true}$ at node $v$.

  – There exists exactly one edge $e \in \delta(v)$ such that $ES(e) = \mathit{branch}$.

  Each pruned node $v \in C_r$ sets its local variables *prize_flag* to *true*, *labelled_flag* to *false*, and $ES(e)$ to *basic* for each edge $e \in \delta(v)$ if $ES(e) \neq \mathit{basic}$. Note that once a node $v$ sets its $ES(e)$ to *basic* for an edge $e = (v, u) \in \delta(v)$ then the node $u$ also does the same. Finally for each of the non-pruned nodes $u \in C_r$, *prize_flag* $=$ *false* and there exists at least one edge $e \in \delta(u)$ such that $ES(e)$ is set to *branch*.

## 5.5 Proof of Correctness

The pseudo-code of the D-PCST algorithm is deferred to Appendix 5.B. It has many procedures. One of the procedures is *proc_initiate*(). A *round of proc_initiate*() in a component $C$ means the time from the beginning of the execution of *proc_initiate*() till the completion of finding $\epsilon(C)$. By *action* of an event $A$ we mean the triggering of event $A$.

**Lemma 5.5.1.** *If the leader of an inactive component $C$ finds $\epsilon_1(C) = \infty$ then for each neighboring component $C_k$ of $C$, $CS(C_k) =$ inactive.*

*Proof.* Suppose by contradiction the leader of $C$ finds $\epsilon_1(C) = \infty$ and there exists a neighboring component $C_k$ of $C$ such that $CS(C_k) \neq$ *inactive*. Therefore either $CS(C_k) =$ *sleeping* or $CS(C_k) =$ *active*. Since it is given that the leader of the component $C$ is in a state of finding its $\epsilon_1(C)$, and $CS(C) =$ *inactive* therefore *Property 1* ensures that $CS(C_k) \neq$ *active* for each neighboring component $C_k$ of $C$.

Consider the case of $CS(C_k) =$ *sleeping*. To find $\epsilon_1(C)$ the leader starts the procedure *proc_initiate*() which in turn sends ⟨*initiate*⟩ on each of its *branch* edges in the component $C$. Upon receiving ⟨*initiate*⟩ each node $v \in C$ forwards it on its outbound *branch* edges and if $v$ is a *frontier* node then it also sends ⟨*test*⟩ on each edge $e$ if state of $e$ is neither *branch* nor *rejected*. In response to each ⟨*test*⟩, $v$ receives either ⟨*status*$(CS(C_k), d_u)$⟩ on an edge $e$ from a node $u \in C_k \neq C$ if $C_k$ is a neighboring component of $C$ or ⟨*reject*⟩ if $e = (v, u)$ such that $v, u \in C$. The ⟨*reject*⟩ is simply discarded by $v$. In case of ⟨*status*$(CS(C_k), d_u)$⟩, $v$ calculates the $\epsilon_e$ for edge $e$. Since $CS(C_k) =$ *sleeping* therefore the *frontier* node $v$ computes $\epsilon_e = w_e - d_v - d_u$. Note that the state of $C_k$ is *sleeping* and the value of $d_u$ is equal to the value of $d_v$. In this case the computed value $\epsilon_e$ is a finite real number, since $w_e, d_v$ and $d_u$ are finite real numbers. Each *frontier* node $v \in C$ sends its computed $\epsilon_1(v)$ to its parent in the rooted tree of the component $C$ and eventually the leader of $C$ finds the value of $\epsilon_1(C)$ to be a finite real number, a contradiction to the fact that $\epsilon_1(C) = \infty$. This completes the proof. □

**Lemma 5.5.2.** *A round of proc_initiate() generates at most $6|V| + 2|E| - 4$ messages.*

*Proof.* In D-PCST algorithm in each round of *proc_initiate*(), the following messages are possibly generated: $\langle initiate \rangle$, $\langle test \rangle$, $\langle status \rangle$, $\langle reject \rangle$, $\langle report \rangle$, $\langle merge \rangle$, $\langle connect \rangle$, $\langle update\_info \rangle$, $\langle back \rangle$, $\langle proceed \rangle$, $\langle accept \rangle$, and $\langle refind\_epsilon \rangle$. Since the maximum number of *branch* edges in a component is at most $|V|-1$, at most $|V|-1$ number of messages are exchanged for each kind of $\langle initiate \rangle$, $\langle report \rangle$, $\langle merge \rangle$, and $\langle update\_info \rangle$ in each round of *proc_initiate*(). Similarly in each round of *proc_initiate*(), at most $|E|$ number of $\langle test \rangle$ messages are sent and in response at most $|E|$ number of $\langle status \rangle$ or $\langle reject \rangle$ messages are generated. The $\langle proceed \rangle$ and $\langle back \rangle$ are exchanged between the leaders of two different components. Therefore, for each kind of $\langle proceed \rangle$ and $\langle back \rangle$ in the worst-case at most $|V|-1$ number of messages are generated in each round of *proc_initiate*(). For each $\langle connect \rangle$, either an $\langle accept \rangle$ or a $\langle refind\_epsilon \rangle$ is generated. Therefore in each round of *proc_initiate*() either the pair $(\langle connect \rangle, \langle accept \rangle)$ or $(\langle connect \rangle, \langle refind\_epsilon \rangle)$ is generated. Therefore the total number of messages exchanged in each round of *proc_initiate*() is at most $4(|V|-1) + 2|E| + 2(|V|-1) + 2 = 6|V| + 2|E| - 4$. □

**Claim 5.5.1.** *D-PCST algorithm generates the action of* $\langle proceed \rangle$ *at most* $|V|-1$ *times.*

*Proof.* During the execution of the D-PCST algorithm an *inactive* component may send $\langle proceed \rangle$ more than once to different components, however a component receives $\langle proceed \rangle$ at most once. In addition the root component is always *inactive* and never receives $\langle proceed \rangle$. Since there are at most $|V|$ components and the root component never receives any $\langle proceed \rangle$, therefore at most $|V|-1$ number of $\langle proceed \rangle$ is generated. □

**Claim 5.5.2.** *D-PCST algorithm generates the action of* $\langle back \rangle$ *at most* $|V|-1$ *times.*

*Proof.* A non-root component $C$ decides to take the action of $\langle back \rangle$ only if $CS(C) = inactive$ and it finds its $\epsilon(C) = \infty$. Lemma 5.5.1 proves that if $CS(C) = inactive$ and $\epsilon_1(C) = \infty$ then $CS(C_k) = inactive$ for each neighboring component $C_k$ of $C$. Also while the leader of $C$ generates the action of $\langle back \rangle$ then $ES(e) \neq refind$ for each edge $e \in \delta(C)$. These facts indicate that all neighboring components of $C$ are explored and therefore action of $\langle back \rangle$ is taken by the leader node in search of a component whose state is still *sleeping*. The component $C$ sends $\langle back \rangle$ to a neighboring component say $C'$ which sent $\langle proceed \rangle$ to $C$ or to a subcomponent of $C$ in some early stages of the algorithm. Since for each action of $\langle proceed \rangle$ at most one $\langle back \rangle$ action is generated and by Claim 5.5.1 the D-PCST algorithm generates the action of $\langle proceed \rangle$ at most $|V|-1$ times, therefore the D-PCST algorithm generates the action of $\langle back \rangle$ at most $|V|-1$ times. □

**Claim 5.5.3.** *If none of the four consecutive rounds of proc_initiate() initiates the action of sending ⟨back⟩ then any one of the following two events is guaranteed to happen: (i) the number of components decrease (ii) the number of sleeping or active components decreases by one.*

*Proof.* In the D-PCST algorithm the leader of a component $C$ starts finding its $\epsilon(C)$ by executing the procedure *proc_initiate*(). Depending on the current state of $C$ i.e. $CS(C)$ and its computed value $\epsilon(C)$, the leader of $C$ decides to take any one of the following actions: (i) *merging* (ii) *deactivation* (iii) *sending* ⟨*proceed*⟩, (iv) *sending* ⟨*back*⟩, and (v) *pruning*. If the action of sending ⟨*back*⟩ is not taken by any one of the four consecutive rounds of *proc_initiate*() then within those rounds of *proc_initiate*() any one of the events mentioned in the Claim 5.5.3 is guaranteed to happen.

First consider the action of *merging*. Before the action of *merging*, the leader of the component $C$ computes its $\epsilon(C)$ in one round of *proc_initiate*(). After that it sends a ⟨*merge*⟩ to the corresponding component say $C'$. If $CS(C') = inactive$ then $C'$ immediately merges with $C$ and in this case merge happens in one round of *proc_initiate*(). As a result one of the components decreases in the graph. If $CS(C') = sleeping$ then $C'$ takes one round of *proc_initiate*() to decide whether to merge with $C$ or deactivate itself. If it decides to merge with $C$ then the number of component decreases by one in the graph. On the other hand if it decides to deactivate itself then a *sleeping* component vanishes in the graph. Therefore whenever the action of *merging* takes place then in at most two rounds of *proc_initiate*() any one of the events mentioned in the Claim 5.5.3 is guaranteed to happen.

We know that only an *active* component can decide to deactivate itself. In this case a component $C$ finds its $\epsilon(C) = \epsilon_2(C)$ in exactly one round of *proc_initiate*(). As a result in exactly one round of *proc_initiate*() one *active* component decreases in the graph.

A component $C$ initiates the action of sending ⟨*proceed*⟩ only if it is in *inactive* state. For this action, first the leader of $C$ computes its $\epsilon_1(C)$ which takes one round of *proc_initiate*(). After that it sends ⟨*proceed*⟩ to the corresponding neighboring component say $C'$. Upon receiving ⟨*proceed*⟩ from $C$, depending on its current state, the component $C'$ does the followings.

(i) $CS(C') = sleeping$. $C'$ starts finding its $\epsilon(C')$ to decide whether to *merge* with some other component or *deactivate* itself. In case of merging, it takes at most two additional rounds of *proc_initiate*() and as a result one component decreases, i.e. from the point of the time of finding $\epsilon_1(C)$ at $C$ upto the merging of the component $C'$ with a neighboring component it takes at most three rounds of *proc_initiate*(). In case of deactivation, $C'$ takes one round of *proc_initiate*() for which one sleeping component decreases and overall it takes two rounds of *proc_initiate*() from the point of the time of finding $\epsilon_1(C)$ at $C$ upto the deactivation of $C'$.

(ii) $CS(C') = inactive$. In this case $C'$ receives ⟨*proceed*⟩ because there exists an edge $e$ such that $e \in \delta(C') \cap \delta(C)$ and the state the edge $e \in \delta(v)$ at some node $v \in C$ must be *refind*. This is because in some early stages of the algorithm the component $C$ or a sub-component of $C$ sent a ⟨*connect*⟩ to $C'$ and in response to that, $C'$ became *inactive* and as a result $C'$ sent back a ⟨*refind_epsilon*⟩ to $C$ or to a sub-component of $C$. Now there should be at least one component $C''$ such that $CS(C'') = sleeping$ in the neighborhood of $C'$. Otherwise $C'$ has to take the action of *sending* ⟨*back*⟩ which is not possible according to our assumption. Since $CS(C') = inactive$, $C'$ takes one round of *proc_initiate*() to compute its $\epsilon_1(C')$ to take the action of *sending* ⟨*proceed*⟩ to a neighboring component $C''$ such that $CS(C'') = sleeping$. After that $C''$ follows at most two rounds of *proc_initiate*() to decide the action of either *merging* or *deactivation* which guarantees the occurring of any one of the mentioned events. Therefore from the point of finding $\epsilon_1(C)$ at $C$ upto any one of the events to be happened takes at most four rounds of *proc_initiate*().

(iii) $CS(C') = active$. The *Property 1* ensures that this condition does not hold.

In case of the action of *pruning*, before the start of the pruning phase the root component $C_r$ takes exactly one round of *proc_initiate*() to compute $\epsilon(C_r)$ which must be equal to $\infty$. After that the growth phase terminates and the round *proc_initiate*() is never initiated by any node in the network. Hence the claim holds. □

**Lemma 5.5.3.** *The* growth *phase of the D-PCST algorithm terminates after at most* $9|V|-7$ *rounds of* proc_initiate().

*Proof.* Initially the state of the root component $C_r$ is *inactive* and it takes one round of *proc_initiate*() to compute its $\epsilon_1(C_r)$ (an inactive component $C$ never computes its $\epsilon_2(C)$). After that $C_r$ sends ⟨*proceed*⟩ to a neighboring component to take further actions of the algorithm. Claim 5.5.3 ensures that in the worst-case at most $4(|V|-1)$

rounds of *proc_initiate*() is required to decrease the number of components and becomes one or at most $4(|V|-1)$ rounds of *proc_initiate*() is required to change the state of each *sleeping* or *active* component to *inactive* state. Claim 5.5.2 ensures that the action of $\langle back\rangle$ is generated at most $|V|-1$ times. If the root node $r \in C_r$ finds that $\epsilon_1(C_r)=\infty$ then instead of taking the action of $\langle back\rangle$ the root component starts the *pruning* phase which indicates the termination of the *growth* phase. Before the termination of the *growth* phase additionally one round of *proc_initiate*() is required to find $\epsilon_1(C_r)=\infty$. Summing for all the cases we get that the total number of rounds of *proc_initiate*() is equal to $1+4(|V|-1)+4(|V|-1)+(|V|-1)+1 = 9|V|-7$. Therefore it is guaranteed that after at most $9|V|-7$ rounds of *proc_initiate*() the initiation of a round of *proc_initiate*() stops. Once the initiation of *proc_initiate*() stops, no more messages related to the *growth* phase are exchanged in the network. This ensures that the *growth* phase eventually terminates after at most $9|V|-7$ rounds of *proc_initiate*(). □

**Lemma 5.5.4.** *Pruning phase of the D-PCST algorithm eventually terminates after generating at most* $3(|V|-1)$ *messages.*

*Proof.* In the *pruning* phase two types of messages are generated namely $\langle prune\rangle$ and $\langle backward\_prune\rangle$. A node $v$ sends $\langle prune\rangle$ on an edge $e \in \delta(v)$ if $ES(e) = branch$ or $EPM(e) = true$. We know that the number of *branch* edges of a component $C$ is exactly $|C|-1$. Whenever the *growth* phase terminates then the total number of *branch* edges of all the components in the graph is at most $|V|-1$. It follows that the total number of $\langle prune\rangle$ messages generated is at most $|V|-1$. Similarly at node $v$, the local boolean variable $EPM(e)$ is set to *true* if it sends a $\langle proceed\rangle$ on $e$. Since the Claim 5.5.1 ensures that the total number of $\langle proceed\rangle$ sent in the D-PCST algorithm is at most $|V|-1$ and for each edge $e$ on which $\langle proceed\rangle$ is sent the variable $EPM(e)$ is set to *true*. This ensures that with respect to the variable *EPM* the total number of $\langle prune\rangle$ generated is at most $|V|-1$. Therefore the total number of $\langle prune\rangle$ generated in the *pruning* phase is at most $2|V|-2$.

A $\langle backward\_prune\rangle$ is exchanged within the root component only. Upon receiving $\langle prune\rangle$ a leaf node (which has exactly one incident *branch* edge) of the root component sends a $\langle backward\_prune\rangle$ on the *branch* edge to its parent if it decides to prune itself. This is continued by every leaf node of the tree rooted at the root node until no more prune is possible. Since the possible number of nodes in the root component is at most $|V|$ and the root node never sends a $\langle backward\_prune\rangle$, at most $|V|-1$ number of $\langle backward\_prune\rangle$ is generated in the *pruning* phase. It follows that the *pruning* phase terminates after at most $3(|V| - 1)$ messages. □

**Theorem 5.5.5.** *The D-PCST algorithm eventually terminates.*

*Proof.* Together Lemma 5.5.3 and Lemma 5.5.4 prove that the D-PCST algorithm eventually terminates. □

**Theorem 5.5.6.** *The message complexity of the D-PCST algorithm is $O(mn)$.*

*Proof.* Both Lemma 5.5.2 and Lemma 5.5.3 ensure that growth phase generates at most $(9|V| - 7)(6|V| + 2|E| - 4)$ messages. Similarly Lemma 5.5.4 ensures that pruning phase generates at most $3(|V| - 1)$ messages. Therefore total number of messages generated until the termination of the D-PCST algorithm is at most $(9|V| - 7)(6|V| + 2|E| - 4) + 3(|V| - 1)$ which is equivalent to $O(|V|^2 + |V||E|)$. Since the graph is connected, $|V|^2 \leq |V||E|$. Therefore the overall message complexity of the D-PCST algorithm is $O(mn)$, where $n = |V|$ and $m = |E|$. □

**Theorem 5.5.7.** *The round complexity of the D-PCST algorithm is $O(n^2)$.*

*Proof.* The round complexity of the D-PCST algorithm is $O(n^2)$ and can be shown as follows. The leader of a component $C$ initiates the *proc_initiate*() method to compute $\epsilon(C)$. A *proc_initiate*() method computes $\epsilon(C)$ using $O(|V|)$ rounds. In every operation of *proc_initiate*(), the leader node broadcasts $\langle initiate \rangle$ over the component $C$ ($C$ is essentially a tree structure rooted at the leader node). We know that the *height* of a component $C$ can be at most $|V| - 1$ and therefore after at most $|V| - 1$ rounds all leaf nodes in $C$ must receive $\langle initiate \rangle$. Upon receiving $\langle initiate \rangle$ a frontier node $v \in C$ sends a $\langle test \rangle$ independently on each edge $e \in \delta(v)$ if $ES(e) = basic$ or $ES(e) = refind$. This takes one unit of time to arrive $\langle test \rangle$ at all the receivers. Upon receiving $\langle test \rangle$ on an incident edge $e \in \delta(u)$, the node $u$ immediately replies with $\langle status \rangle$ if $u \notin C$ or with $\langle reject \rangle$ if $u \in C$. Within one unit of time a frontier node receives all $\langle status \rangle$ and $\langle reject \rangle$ and after that it computes its $\epsilon_1(v)$ locally in constant time units. After that if $v$ is a leaf node then it immediately reports $\epsilon_1(v)$ to its parent. On the other hand if $v$ is an intermediate node it waits to arrive $\epsilon_1(u)$ from each of its child nodes $u$. Among all of these values (its own computed value $\epsilon_1(v)$ and all received values from its children) the minimum one is reported to its parent. This is followed by every intermediate node in a component $C$. In this way after $O(|V|)$ rounds $\epsilon_1(C)$ survives and reaches the leader of $C$. Then the leader computes $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C))$. Therefore a round of *proc_initiate*() takes $O(|V|)$ rounds to compute $\epsilon(C)$ in any component $C$.

The completion of a round of *proc_initiate*() is immediately followed by any one of the procedures *merging, deactivation, proceed, back*, and *pruning*, each of which takes $O(|V|)$ units of time in the worst-case. Note that the pruning procedure is performed only once and the termination of the pruning phase implies the termination the D-PCST algorithm. Since Lemma 5.5.3 ensures that the growth phase of the D-PCST algorithm terminates after at most $9|V|-7$ rounds of *proc_initiate*(), and therefore in the worst-case growth phase takes at most $(9|V|-7)O(|V|)$ rounds. Similarly Lemma 5.5.4 ensures that pruning phase terminates in $O(|V|)$ rounds. Therefore in the worst-case the total number of rounds required by the D-PCST algorithm is at most $(9|V| - 7)O(|V|) + O(|V|) = O(|V|^2) = O(n^2)$. □

**Memory efficiency.** We measure the memory complexity of the D-PCST algorithm as follows. Each node in the network executes the same D-PCST algorithm. Each node maintains its own set of variables, consisting of its state (denoted by $SN$, with possible values $find$ and $found$), the state of its component (denoted by $CS$, with possible values *sleeping*, *inactive*, and *active*), the state of the adjacent edges (two variables denoted by $ES$ and $EPM$: the possible values of $ES$ are *basic*, *branch*, *reject*, and *refind*; the possible values of $EPM$ are *true* and *false*), and a constant number of other variables, each of which is polynomially bounded in $n$. Note that except the state variables of the incident edges, the number of other state variables is constant. Regarding the incident edges, since $ES$ has four possible states, two bits suffice to represent them. Similarly, one bit is sufficient to represent all states of the boolean variable $EPM$. Therefore, three bits are required to store all state values of an incident edge. This implies that $O(\Delta)$ bits are sufficient to store all state values for all incident edges (assuming $\Delta$ is the maximum degree of a node in the network). Furthermore during the execution of the algorithm large degree nodes may receive messages from all of their neighbors in a round. This ensures that the memory requirement is $O(\Delta \log n)$ bits per node. As usual we do not count the amount of memory required to store the input data (node's ID and prize value, and weights of incident edges). Regarding message size, each message consists of at most four parameters, each of which is either a node ID or an edge weight or a combination of a constant number of IDs and weights, or a boolean variable; each can be represented by $O(\log n)$ bits. Since the number of parameters in each message is constant, the maximum message size is $O(\log n)$ bits. Therefore, the overall amount of working memory required by a node is $O(\Delta \log n)$ bits.

**Deadlock freeness.** The D-PCST algorithm does not suffer from any deadlock. Consider the case of *merging* of two neighboring components, say $C$ and $C'$. This is the only case where a component needs to wait for another component to proceed further. Assume that the component $C$ sends $\langle$*connect*$\rangle$ to a neighboring component $C'$ for merging. Upon receiving $\langle$*connect*$\rangle$, if $CS(C') =$ *inactive* then $C'$ immediately replies back $\langle$*accept*$\rangle$ to $C$. If $CS(C') =$ *sleeping*, then it first changes $CS(C')$ to *active* and then finds its $\epsilon(C')$ and depending on $\epsilon(C')$ it replies back $\langle$*accept*$\rangle$ or $\langle refind\_epsilon\rangle$ to $C$. Since $C'$ does not depend on any event for finding its $\epsilon(C')$, $C'$ can response to $C$ within a finite units of time. This guarantees that there is no communication deadlock between $C'$ and $C$. Furthermore, in the D-PCST algorithm, parallel merging is not allowed; at a time at most one merging happens between two distinct components. This ensures the deadlock freeness of D-PCST algorithm.

### 5.5.1 Distributed Preservation of Dual Constraints

During the execution of the D-PCST algorithm whenever an active component $C_a$ merges with an inactive component $C_i$ then only the dual variable $y_{C_a}$ is updated and $y_{C_i}$ remains the same. In this case the preservation of dual packing constraints is obvious since $y_{C_a}$ is effected by the value $|\epsilon(C_a)|$ only which is computed by the component $y_{C_a}$ itself. Similar argument holds whenever an active component decides to deactivate itself. However, there is a more involvement of how a sleeping component computes the values of its dual variables whenever it receives a connect or proceed request from a neighboring component and still preserves dual packing constraints in distributed way. Towards this we prove the following lemma.

**Lemma 5.5.8.** *If $CS(C) =$ sleeping and it receives a $\langle connect(v, W(C'), d_v)\rangle$ or $\langle proceed(d_v)\rangle$ over an edge $e$ from a node $v \in C'$ where $C'$ is a neighboring component of $C$ then both $C$ and $C'$ correctly compute their local variables without violating any of the dual packing constraints.*

*Proof.* Since $CS(C) =$ *sleeping*, $C$ is a single node component. Let it be $\{u\}$. If $u$ receives $\langle connect(v, W(C'), d_v)\rangle$ from a node $v \in C'$ over the edge $e$ then first $C$ becomes active and then $u$ initializes each of its local variables $d_u$ and $W(C)$ to $d_v$. $W(C) = d_v$ indicates that the implicit variable $y_C$ is temporarily initialized to $d_v$. After that $u$ computes $\epsilon(C)$ as follows:

$$\begin{aligned}\epsilon_e =& \frac{w_e - d_u - d_v}{2}\\ \epsilon_2(C) =& TP(C) - W(C) = p_u - W(C)\end{aligned}$$

and $\epsilon(C) = \min(\epsilon_e, \epsilon_2(C))$. Now there are four possible cases we discuss below. For each case, with the help of the local information we show that dual packing constraints are preserved in distributed way without having the global knowledge of the graph.

**Case 1**: $\epsilon(C) = \epsilon_e < 0$. This condition indicates that the dual *edge packing* constraint $\sum_{A:e\in\delta(A)} y_A \leq w_e$ is violated on edge $e$. More specifically dual variables $y_C$ and $y_{C'}$ are excessively increased by $|\epsilon(C)| = |\epsilon(C')|$. To ensure that the dual edge packing constraint remains tight while $C$ and $C'$ merge, the excess value $|\epsilon(C)|$ must be deducted from each of the dual variables $y_C$ and $y_{C'}$. The implicit variables $y_C$ and $y_{C'}$ are updated to $y_C - |\epsilon(C)|$ and $y_{C'} - |\epsilon(C')|$ respectively. Also each node $v \in C \cup C'$ updates its local variables $d_v$ and $W(C \cup C')$ to $d_v - |\epsilon(C)|$ and $W(C) + W(C') - 2|\epsilon(C)|$ respectively. Case 1 is pictorially shown in Figure 5.1.

**Case 2**: $\epsilon(C) = \epsilon_e$ and $\epsilon(C) \geq 0$. This ensures that at most $\epsilon(C)$ can be added to both the dual variables $y_C$ and $y_{C'}$ without violating the dual edge packing constraint $\sum_{A:e\in\delta(A)} y_A \leq w_e$ for edge $e$. Therefore the components $C$ and $C'$ merge through the edge $e$ and form a bigger component $C \cup C'$. Each node $v \in C \cup C'$ updates its local variables $d_v$ and $W(C \cup C')$ to $d_v + \epsilon(C)$ and $W(C) + W(C') + 2\epsilon(C)$ respectively. Also the implicit variables $y_C$ and $y_{C'}$ are updated to $y_C + \epsilon(C)$ and $y_{C'} + \epsilon(C')$ respectively. Note that here $\epsilon(C) = \epsilon(C')$. Case 2 is pictorially shown in Figure 5.2.

**Case 3**: $\epsilon(C) = \epsilon_2(C)$ and $\epsilon(C) < 0$. In this case the dual variable $y_C$ for the component $C$ is excessively increased by $|\epsilon(C)|$ and this indicates that the dual *penalty packing* constraint $\sum_{A\subseteq C} y_A \leq \sum_{v\in C} p_v$ is violated at $C$. Therefore $y_C = y_C - |\epsilon_2(C)|$ and it ensures that the dual *penalty packing* constraint for the component $C$ is not violated and becomes tight. Node $u \in C$ updates its local variables $d_u$ and $W(C)$ to $d_u - |\epsilon(C)|$ and $W(C) - |\epsilon(C)|$ respectively. The values of all the variables in $C'$ remain the same. Case 3 is pictorially shown in Figure 5.3.

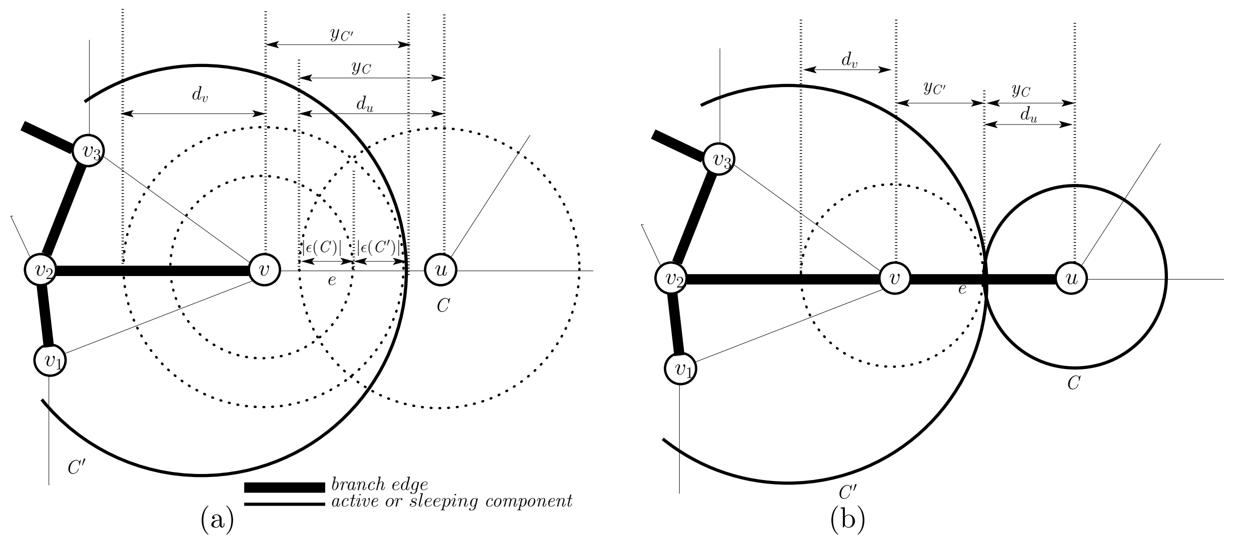


**Figure 5.1:** *A case of $\epsilon(C) = \epsilon_e < 0$. (a) state before merging of components $C'$ and $C$. $C'$ sends a connection request to $C$ (a sleeping component) to merge with it. A lightly dotted circle depicts the current values of some variables corresponding to a component or a sub-component. The initial values of the dual variables $y_C$ and $y'_C$ are $d_u$ and $d_v$ respectively. (b) state after merging. The dual variables $y_C$ and $y'_C$ are updated to $y_C - |\epsilon(C)|$ and $y_{C'} - |\epsilon(C')|$ respectively. Also each node $v \in C \cup C'$ updates its local variables $d_v$ and $W(C \cup C')$ to $d_v - |\epsilon(C)|$ and $W(C) + W(C') - 2|\epsilon(C)|$ respectively.*

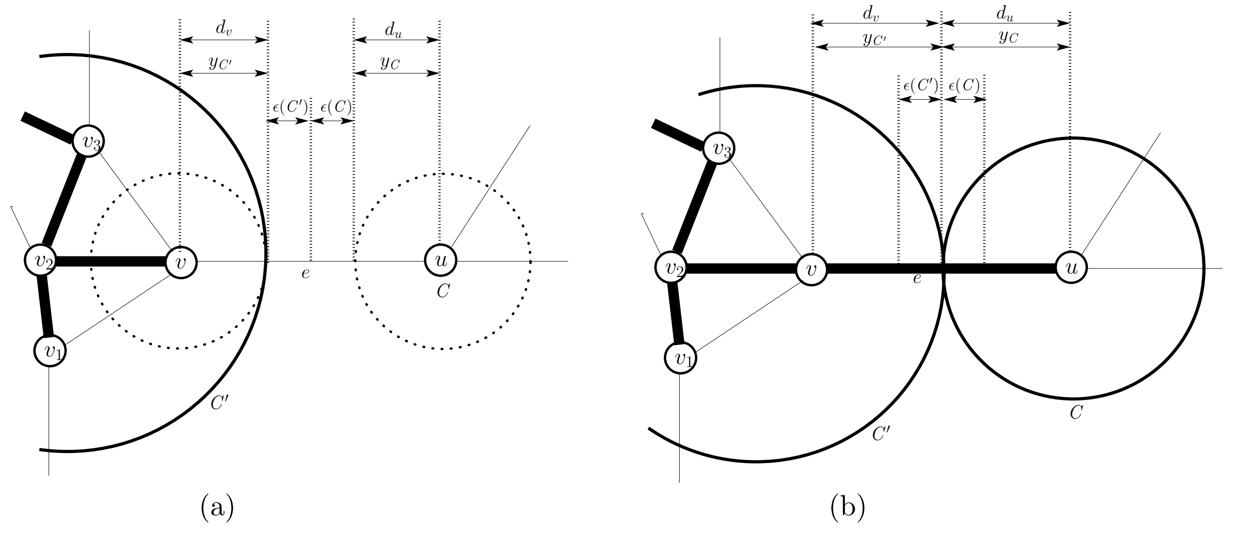


**Figure 5.2:** *A case of $\epsilon(C') = \epsilon(C) \geq 0$. (a) state before merging of components $C'$ and $C$. $C'$ sends a connection request to $C$ to merge with it. The initial values of the dual variables $y_C$ and $y'_C$ are $d_u$ and $d_v$ respectively. (b) state after merging. The dual variables $y_C$ and $y'_C$ are updated to $y_C + \epsilon(C)$ and $y_{C'} + \epsilon(C')$ respectively. Also each node $v \in C \cup C'$ updates its local variables $d_v$ and $W(C \cup C')$ to $d_v + \epsilon(C)$ and $W(C) + W(C') + 2\epsilon(C)$ respectively.*

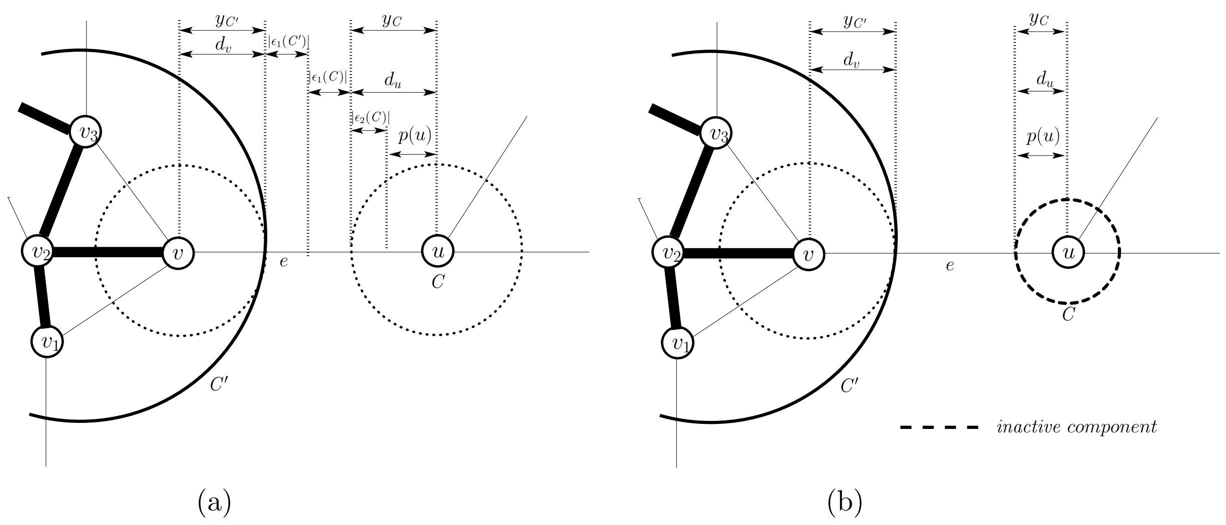


**Figure 5.3:** *A case of $\epsilon(C) = \epsilon_2(C) \leq 0$. (a) state before the deactivation of components $C$. $C'$ sends a connection request to $C$ to merge with it. The initial values of the dual variables $y_C$ and $y'_C$ are equal to the values of $d_u$ and $d_v$ respectively. (b) state after deactivation. The dual variables $y_C$ is updated to $y_C - |\epsilon(C)|$, and $y'_C$ and $d_v$ remain the same. Also $d_u$ is updated to $d_u = d_u - |\epsilon(C)|$.*

**Case 4** : $\epsilon(C) = \epsilon_2(C) \geq 0$. This indicates that at most $\epsilon(C)$ can be added to the dual variable $y_C$ in component $C$ without violating the dual *penalty packing* constraint $\sum_{A \subseteq C} y_A \leq \sum_{v \in C} p_v$. The node $u \in C$ updates its local variables $d_u$ and $W(C)$ to $d_u + \epsilon(C)$ and $W(C) + \epsilon(C)$ respectively. The values of all the variables in $C'$ remain the same. This case is pictorially shown in Figure 5.4.

Therefore whenever an active component $C'$ sends a connect request to a sleeping component $C$ then both $C'$ and $C$ correctly compute their local variables in a distributed way without violating any of the dual packing constraints.

Similarly if a node $u$ receives $\langle proceed(d_v)\rangle$ from a node $v \in C'$ over an edge $e$ then first $C = \{u\}$ becomes active and then $u$ initializes each of its local variables $d_u$ and $W(C)$ to $d_v$. Note that if a component receives a $\langle proceed\rangle$ then by *Property 1* the state of each component in its neighborhood is either *sleeping* or *inactive*. This implies that $CS(C') = inactive$. Let $e' \in \delta(C)$ be the MEOE of $C$ which connects to a node $w \in C_p \neq C$. Considering $d_w = d_u$,

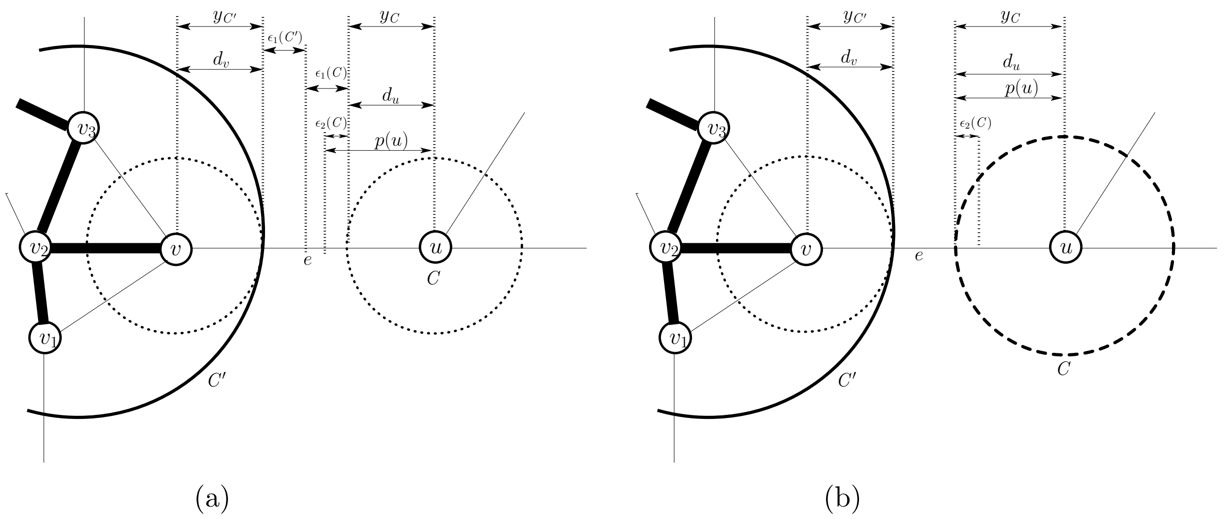


**Figure 5.4:** *A case of $\epsilon(C) = \epsilon_2(C) \geq 0$. (a) state before the deactivation of components $C$. $C'$ sends a connection request to $C$ to merge with it. The initial values of the dual variables $y_C$ and $y'_C$ are $d_u$ and $d_v$ respectively. (b) state after the deactivation. $y_C$ is updated to $y_C + \epsilon_2(C)$, and $y'_C$ and $d_v$ remain the same. Also $d_u$ is updated to $d_u = d_u + \epsilon_2(C)$.*

$C$ computes its $\epsilon(C)$ as follows:

$$\epsilon_1(C) = \begin{cases} \frac{w_{e'} - d_u - d_w}{2}, & \text{if } CS(C_p) = \textit{sleeping} \\ w_{e'} - d_u - d_w, & \text{if } CS(C_p) = \textit{inactive} \end{cases}$$
$$\epsilon_2(C) \;=\; TP(C) - W(C) = p_u - W(C)$$

and $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C))$. Now following the same way as we have shown for the case of receiving $\langle \textit{connect} \rangle$, it can be shown that upon receiving $\langle \textit{proceed}(d_v) \rangle$, $C$ correctly computes each of its local variables without violating any of the dual packing constraints. $\square$

## 5.5.2 Performance of the D-PCST Algorithm

We claim that the approximation factor achieved by the D-PCST algorithm is $(2 - \frac{1}{n-1})$ of the optimal (OPT). This can be proved from the facts that $d_v = \sum_{A:v \in A} y_A$ for each node $v \in V$ and $W(C) = \sum_{A \subseteq C} y_A$ for each component $C$. Let $OPT_{LP}$ and $OPT_{IP}$ be the optimal solutions to (LP) and (IP) of the PCST problem respectively. Then it is obvious that $\sum_{A \subset V} y_A \leq OPT_{LP} \leq OPT_{IP}$. We claim the following theorem for the approximation factor.

**Theorem 5.5.9.** *D-PCST algorithm selects a set of edges $F'$ and a set of vertices $X$ such that*

$$\sum_{e\in F'} w_e + \sum_{v\in X} p_v \le \left(2-\frac{1}{n-1}\right)\sum_{A\subset V} y_A \le \left(2-\frac{1}{n-1}\right) OPT_{IP} \tag{5.1}$$

*where $n = |V|$ and $OPT_{IP}$ is the optimal solution to the IP of the PCST.*

*Proof.* The proof of the above theorem is inspired by the proof of the approximation ratio of the GW-algorithm [58]. Note that the GW-algorithm is a sequential algorithm and achieved an approximation ratio $\left(2-\frac{1}{n-1}\right)$ of the optimal. However, in our proposed algorithm certain modifications have been made in order to preserve dual packing constraints in a distributed way as it is shown in Lemma 5.5.8 and still we achieve the same approximation factor as that of the GW-algorithm.

In the construction of $F'$ if a node $v \in V$ is not covered by $F'$ then $v$ must belong to some component deactivated at some point of execution of the algorithm. Let $K = \{D_1, D_2, ..., D_z\}$ is the set of deactivated components whose nodes are not covered by $F'$. Therefore $K$ can be considered as a set of disjoint subsets of vertices and each subset is some $D_j$ for $j : 1 \le j \le z$. Since each $D_j$ is a deactivated component, it follows that $\sum_{A\subseteq D_j} y_A = \sum_{v\in D_j} p_v$. For each edge $e \in F'$ it also follows that $\sum_{A:e\in\delta(A)} y_A = w_e$ and this implies $\sum_{e\in F'} w_e = \sum_{e\in F'}\sum_{A:e\in\delta(A)} y_A$. Putting these in the inequality (5.1) we get

$$\sum_{e\in F'}\sum_{A:e\in\delta(A)} y_A + \sum_{j}\sum_{A\subseteq D_j} y_A \le \left(2-\frac{1}{n-1}\right)\sum_{A\subset V} y_A \tag{5.2}$$

The first and second terms in the left hand side of the above inequality correspond to the Steiner part and Penalty part of the PCST respectively. Note that the pruning phase of the algorithm enhances the PCST result computed in the growth phase of the D-PCST-algorithm. Therefore pruning phase, if possible, improves the approximation factor and in the worst-case it would be same as that of the result of the PCST computed in the growth phase. These facts ensures that the approximation factor computed just after the termination of the growth phase is the overall approximation factor of the D-PCST algorithm. Note that the termination of the pruning phase ensures the overall termination of the D-PCST algorithm.

Let $\mathbb{F} = \{\mathbb{C}_r, \mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_k\}$ be the set of components when the growth phase of the algorithm terminates; here $\mathbb{C}_r$ is the root component and all others are non-root components. Also assume that $F$ is the set of branch edges in $\mathbb{F}$. Note that after the termination of the pruning phase, $F' \subseteq F$ is the set of branch edges finally remains in the Steiner part of the PCST. Let $\mathbb{C} = (\mathbb{V}, \mathbb{E})$ be any component in $\mathbb{F}$. During the growth phase $\mathbb{C}$ consists of a set of disjoint sub-components, say $\{C_1, C_2, \ldots, C_b\}$. Then the following inequality always holds for any component $\mathbb{C} \in \mathbb{F}$.

$$\sum_{e \in \mathbb{E}} \sum_{A : e \in \delta(A)} y_A + \sum_{j} \sum_{A \subseteq C_j} y_A \leq \left(2 - \frac{1}{n-1}\right) \sum_{A \subseteq \mathbb{C}} y_A \tag{5.3}$$

Where $C_j \in \mathbb{C}$ is some component deactivated at some point of the time of execution and later joined the component $\mathbb{C}$. Now it can be shown by the method of induction that for each $\epsilon(C) \geq 0$ computed by an active component $C \in \mathbb{C}$, the inequality (5.3) always holds. Note that if $\epsilon(C) \geq 0$ then we assume that each of the dual variables $y_C$ for $C \subseteq \mathbb{C}$ is implicitly increased by an amount of $\epsilon(C)$. On the other hand $\epsilon(C) < 0$ indicates that the value $|\epsilon(C)|$ is infeasible and is not considered as a value for any dual variable. However in case of $\epsilon(C) < 0$, the value of $|\epsilon(C)|$ is used to preserve the dual variables associated with it in distributed way as we have shown in the Subsection 5.5.1.

At the beginning of the growth of the component $\mathbb{C}$ the inequality (5.3) holds since $\mathbb{E} = \emptyset$, $y_C = 0$ for each single node component $C$ in $\mathbb{C}$. Now we prove the induction hypothesis for any instance of the growth phase of the component $\mathbb{C}$. Before that the sub-components of $\mathbb{C}$ are categorized into two types of components namely type $A$ and type $I$ as follows:

- A component $C \in \mathbb{C}$ is of type $A$ if $CS(C) = \textit{active}$ **or** $CS(C) = \textit{sleeping}$.
- A component $C \in \mathbb{C}$ is of type $I$ if $CS(C) = \textit{inactive}$.

The type of a component $C$ is denoted by $\textit{type}(C)$.

Consider an instance of the growth phase in which $C \subseteq \mathbb{C}$ is an active sub-component. At this point of execution of the algorithm the leader of the component $C$ computes its $\epsilon(C)$ to take further actions of the algorithm. Now we construct a special graph termed as $H = (V', E')$ as follows. The set of sub-components of $\mathbb{C}$ is considered as the set of vertices $V'$ of $H$. The set of edges is $E' = \{e \in \delta(C') \cap \mathbb{E} : C' \subseteq \mathbb{C} \wedge \textit{type}(C') = A\}$. All isolated vertices of type $I$ are discarded from the graph $H$. Let $N_A$ denotes the set of vertices of type $A$, $N_I$ denotes the set of vertices of type $I$, and $\psi_v$ denotes the degree of a vertex $v$ in graph $H$. For $\epsilon(C) > 0$, maximum increment in the left hand side of the inequality (5.3) is $\sum_{v \in N_A} \epsilon(C)\psi_v$. On the other hand, the maximum increment in the right hand side of the inequality is $(2 - \frac{1}{n-1})\sum_{v \in N_A} \epsilon(C)$. Therefore we can write,

$$\sum_{v\in N_A} \epsilon(C)\psi_v \leq \left(2-\frac{1}{n-1}\right)\sum_{v\in N_A}\epsilon(C)$$

Rewriting the left hand side of the above inequality in terms of $N_A$ and $N_I$ we get

$$\sum_{v\in N_A}\psi_v \leq \sum_{v\in N_A\cup N_I}\psi_v - \sum_{v\in N_I}\psi_v \tag{5.4}$$

We know that the sum of degrees of all the vertices of a graph of $m$ edges is $2\times m$. Since $H$ is a tree, the total number of edges in $H$ is $|N_A\cup N_I|-1$. Furthermore, since all the vertices in graph $H$ are connected, therefore the sum of degrees of type $I$ vertices in $H$ is at least $|N_I|$. Using these facts in the inequality (5.4) we get

$$\begin{aligned}\sum_{v\in N_A}\psi_v &\leq 2\big(|N_A\cup N_I|-1\big)-|N_I|\\ &=2\big(|N_A|+|N_I|-|N_A\cap N_I|-1\big)-|N_I|\end{aligned}$$

Since $N_A$ and $N_I$ are disjoint, $|N_A\cap N_I|=|\emptyset|=0$. Therefore,

$$\begin{aligned}\sum_{v\in N_A}\psi_v &\leq 2\big(|N_A|+|N_I|-1\big)-|N_I|\\ &=2\big(|N_A|-1\big)+|N_I|\end{aligned}$$

Again note that if $\mathbb{C}\neq\mathbb{C}_r$ then the number of components of type $I$ can be zero, i.e., $|N_I|=0$. In this case all nodes in graph $H$ are of type $A$ which indicates that each of the degrees of vertices in graph $H$ is increased by an amount of $\epsilon(C)$. Therefore the following inequality holds.

$$\sum_{v\in N_A}\psi_v \leq 2(|N_A|-1) = \left(2-\frac{2}{|N_A|}\right)|N_A| \leq \left(2-\frac{2}{n-1}\right)|N_A| \tag{5.5}$$

The inequality (5.5) holds since the total number of type $A$ components in graph $H$ is at most $n-1$ for a graph with $n$ nodes, i.e. $|N_A|\leq n-1$. On the other hand if $\mathbb{C}=\mathbb{C}_r$ then $\mathbb{C}$ contains at least one component of type $I$. Therefore we get

$$\sum_{v\in N_A}\psi_v \leq 2|N_A|-1 = \left(2-\frac{1}{|N_A|}\right)|N_A| \leq \left(2-\frac{1}{n-1}\right)|N_A| \tag{5.6}$$

The inequalities (5.5) and (5.6) together guarantee the correctness of the inequality (5.3) for any component $\mathbb{C} \in \mathbb{F}$. If we sum up the inequalities of all the components in $\mathbb{F}$ then it is clear that (5.2) always holds in $G$. Note that after the termination of the growth phase the application of the pruning phase on $\mathbb{F}$ enhances the PCST result producing the final PCST $(F', X)$. Therefore the D-PCST-algorithm achieves an approximation factor of $(2 - \frac{1}{n-1})$. □

## 5.6 Fast PCST construction

In this section we modify the D-PCST algorithm and show that the modified algorithm can compute a $(2 - \frac{1}{n-1})$-approximate PCST using $O(Dn)$ rounds and $O(mn)$ messages with a memory requirement of $O(\Delta \log n)$ bits per node. For networks with constant or small unweighted diameter ($D = o(n)$), the modified D-PCST algorithm performs better than the original D-PCST algorithm in terms of the round complexity. The correctness of the approximation factor of the modified D-PCST algorithm directly follows from the correctness of the original D-PCST algorithm.

In the modified D-PCST algorithm we assume that there exists a BFS tree $T_r$ of the input graph $G$ rooted at $r$, the root of the PCST (here any node can be considered as a root of the BFS tree, however for simplicity we assume $r$). Note that such a tree is not considered in the original D-PCST algorithm. Also note that a rooted BFS tree can be deterministically computed using $O(D)$ rounds and $O(m)$ messages in the CONGEST model [114].

**Overview of the modified D-PCST algorithm.** Similar to the original D-PCST algorithm, the modified D-PCST algorithm also consists of two phases: growth and pruning. Recall that the growth phase of the original D-PCST algorithm performs a number of calls to *proc_initiate*() method. The *proc_initiate*() method is essentially a combination of broadcast and convergecast. Using an improved broadcast and convergecast technique, we achieve an improved *proc_initiate*() method which is used by the modified D-PCST algorithm. The pruning phase of the modified D-PCST algorithm is performed by applying a procedure called *strong pruning* due to Johnson et al. [72].

Each *proc_initiate*() method computes $\epsilon(C)$ and $MEOE(C)$ for a component $C$ and performs one of the operations: *merging, deactivation, proceed, back*. By leveraging on $T_r$, the modified D-PCST algorithm performs a *proc_initiate*() method using $O(D)$ rounds. The leader of a component $C$ initiates the *proc_initiate*() method by sending an $\langle initiate \rangle$ to $r$

by using $T_r$. Upon receiving an $\langle initiate \rangle$ message, $r$ applies a broadcast and convergecast over $T_r$ to compute $\epsilon(C)$ and $MEOE(C)$. Since the height of $T_r$ is at most $D$, a broadcast and convergecast takes $O(D)$ rounds. Based on the state of $C$ and the value of $\epsilon(C)$, $r$ decides to perform one of the operations: *merging, deactivation, proceed, back*. Node $r$ can perform each such operation by applying a broadcast over $T_r$ using $O(D)$ rounds. Therefore it is clear that the overall round complexity of a *proc_initiate*() method in the modified D-PCST algorithm is $O(D)$.

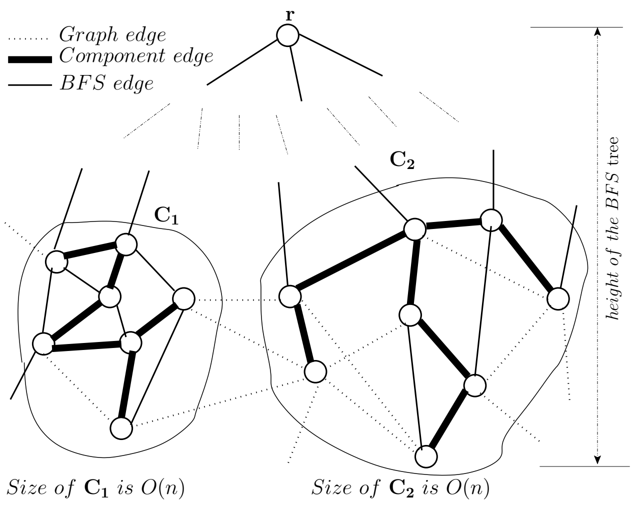


**Figure 5.5:** *A case of merging of two components $C_1$ and $C_2$ where each component has a size of $O(n)$; the original D-PCST algorithm takes $O(n)$ rounds for merging whereas the modified D-PCST algorithm performs the same using $O(h)$ rounds.*

On the other hand a *proc_initiate*() method in the original D-PCST algorithm may require $O(n)$ rounds. For clarity consider a pictorial representation of a merging operation which is depicted in Figure 5.5. The component $C_1$ (of size $O(n)$) decides to merge with a neighboring component $C_2$ (of size $O(n)$). After merging, the size of the resultant component $C_1 \cup C_2$ becomes $O(n)$. In merging operation, each node in $C_1 \cup C_2$ updates its local information as per the new component $C_1 \cup C_2$. Note that $C_1 \cup C_2$ is a tree structure whose height can be $O(n)$. The original D-PCST algorithm applies a broadcast using the tree embedding of $C_1 \cup C_2$ to update the local information in each node of $C_1 \cup C_2$. Therefore for components with size $O(n)$, the merging operation in the original D-PCST algorithm requires $O(n)$ rounds. However the modified D-PCST algorithm can perform the same using $O(h)$ rounds by leveraging on a BFS tree $T_r$ as shown in Figure 5.5, where $h$ is the height of the BFS tree and $h \leq D$. Similarly we claim that each of the other operations of the *proc_initiate*() method in the modified D-PCST algorithm can be performed using $O(h)$ rounds.

### 5.6.1 Properties of the modified D-PCST algorithm

Let $T_v$ denotes a subtree of $T_r$ rooted at $v$. In addition to the notations defined in section 5.4, in this subsection, we use the following notation.

- $bfs\_child(v)$ denotes the set of child nodes of a node $v$ w.r.t. $T_r$.
- $bfs\_\pi(v)$ denotes the parent of a node $v$ in $T_r$.
- $\epsilon_1(T_v) = \min\{\epsilon_1(v), \min\{\epsilon_1(T_u) \mid u \in bfs\_child(v)\}\}$.
- $MEOE(T_v)$ denotes the edge which gives $\epsilon_1(T_v)$.
- $P(T_v) = \sum_{v \in T_v \cap C} p_v$. If $T_v \cap C = \phi$, $P(T_v) = 0$.

**Lemma 5.6.1.** *Let $T_r$ be a BFS tree of the input graph $G$ rooted at the special node $r$. Then for a component $C$, $r$ can compute $\epsilon(C)$ and $MEOE(C)$ using $O(D)$ rounds and $O(m)$ messages.*

*Proof.* In the modified D-PCST algorithm, the leader $L(C)$ of the component $C$ triggers a *proc_initiate*() method by sending an $\langle initiate(L(C), W(C), CS(C))\rangle$ message to $bfs\_\pi(L(C))$. Upon receiving $\langle initiate(L(C), W(C), CS(C))\rangle$, a node $v$ simply forwards it to $bfs\_\pi(v)$. In this way eventually node $r$ receives $\langle initiate(L(C), W(C), CS(C))\rangle$. Now node $r$ broadcasts the message $\langle initiate_r(L(C), W(C), CS(C))\rangle$ over $T_r$ ($\langle initiate_r\rangle$ indicates that the message is sent by $r$). Whenever a leaf node $v$ in $T_r$ receives $\langle initiate_r(L(C), W(C), CS(C))\rangle$, it performs the following.

- if $v \in C$ then it computes $\epsilon_1(v)$ as described in Subsection 5.4.2 and sends a message called $\langle report(v, p_v, \epsilon_1(v), MEOE(v)\rangle$ to $bfs\_\pi(v)$. Note that each $v \in C$ knows $L(C)$ and $bfs\_\pi(v)$.
- if $v \notin C$ then it simply sends $\langle report(v, 0, \infty, nill\rangle$ to $bfs\_\pi(v)$.

Each internal node in $T_r$ waits to arrive $\langle report(u, P(T_u), \epsilon_1(T_u), MEOE(T_u))\rangle$ from each node $u$ in $bfs\_child(v)$. If $v \in C$ is an internal node in $T_r$ then it computes $\epsilon_1(v)$,

$P(T_v) = p_v + \sum_{u \in \mathit{bfs\_child}(v)} P(T_u)$, and $\mathit{MEOE}(v)$. On the other hand if $v \notin C$ is an internal node in $T_r$ then $\epsilon_1(v) = \infty$, $P(T_v) = \sum_{u \in \mathit{bfs\_child}(v)} P(T_u)$, and $\mathit{MEOE}(v) = \mathit{nill}$. After receiving $\langle \mathit{report} \rangle$ from all of its child nodes, an internal node $v$ computes $\epsilon_1(T_v) = \min\{\epsilon_1(v), \min\{\epsilon_1(T_u) \mid u \in \mathit{bfs\_child}(v)\}\}$ and $P(T_v)$ and then sends a $\langle \mathit{report}(v, P(T_v), \epsilon_1(T_v), \mathit{MEOE}(T_v)) \rangle$ message to $\mathit{bfs\_}\pi(v)$. In this way $r$ eventually receives $\langle \mathit{report} \rangle$ messages from all of its child nodes and computes $\epsilon_1(C)$ and $\mathit{MEOE}(C)$. Now node $r$ can also compute $\epsilon_2(C) = P(T_r) - W(C)$ and $\epsilon(C) = \min\{\epsilon_1(C), \epsilon_2(C)\}$. Since the height of $T_r$ can be at most $D$, for a component $C$, $r$ can compute $\epsilon(C)$ and $\mathit{MEOE}(C)$ using $O(D)$ rounds.

Note that during the computation of $\epsilon_1(v)$ and $\mathit{MEOE}(v)$, each node $v$ sends one $\langle \mathit{test} \rangle$ message and receives either $\langle \mathit{status} \rangle$ or $\langle \mathit{reject} \rangle$ on each of the edges in $\delta(v) \cap \delta(C)$ (the detailed description can be found in Subsection 5.4.2). If we consider all the edges in $\delta(C)$ then $|\delta(C)| = O(m)$ in the worst-case. Therefore $O(m)$ messages are generated due to $\langle \mathit{test} \rangle$, $\langle \mathit{status} \rangle$, and $\langle \mathit{reject} \rangle$. Other messages, which are related to the computation of $\epsilon(C)$ and $\mathit{MEOE}(C)$ are $\langle \mathit{initiate} \rangle$ and $\langle \mathit{report} \rangle$. These two messages are sent over the edges of $T_r$ only. On each edge $e$ of $T_r$, $\langle \mathit{initiate} \rangle$ message is sent at least once and at most twice. It is at least once because $r$ broadcasts $\langle \mathit{initiate} \rangle$ message over $T_r$. If $e$ is on the path between $L(C)$ and $r$ then another $\langle \mathit{initiate} \rangle$ message is sent over $e$. The $\langle \mathit{report} \rangle$ message is used for convergecast process over $T_r$. During the computation of $\epsilon(C)$ and $\mathit{MEOE}(C)$ exactly one convergecast of $\langle \mathit{report} \rangle$ message is performed. This ensures that exactly one $\langle \mathit{report} \rangle$ message is sent over each edge in $T_r$. Since $T_r$ has $n-1$ edges, in the worst-case, $O(n)$ messages are generated due to $\langle \mathit{initiate} \rangle$ and $\langle \mathit{report} \rangle$ during the computation of $\epsilon(C)$ and $\mathit{MEOE}(C)$. Therefore total messages generated during the computation of $\epsilon(C)$ and $\mathit{MEOE}(C)$ is $O(m+n)$. Since the given input graph $G$ is connected, $m \geq n-1$. Therefore $O(m+n) = O(m)$. □

Based on the state of $C$ and the value of $\epsilon(C)$, node $r$ triggers one of the following operations.

- Merging: if $\mathit{CS}(C) = \mathit{active}$ and $\epsilon(C) = \epsilon_1(C)$.
- Deactivation: if $\mathit{CS}(C) = \mathit{active}$ and $\epsilon(C) = \epsilon_2(C)$.
- Proceed: if $\mathit{CS}(C) = \mathit{inactive}$ and there exists at least one active neighboring component of $C$.
- Back: if $\mathit{CS}(C) = \mathit{inactive}$ and there exists no active neighboring component of $C$.

**Claim 5.6.1.** *Let the root $r$ of the BFS tree $T_r$ knows $MEOE(C) = (u, v)$, where $u \in C, v \in C'$ and $C \neq C'$. Then node $r$ can find $CS(C')$, $W(C')$, and $L(C')$ using $O(D)$ rounds and $O(n)$ messages.*

**Lemma 5.6.2.** *Let $T_r$ be a BFS tree of the input graph $G$ rooted at the special node $r$. Then the merging operation can be performed using $O(D)$ rounds and $O(n)$ messages.*

*Proof.* Since it is a merging operation, $CS(C) = active$ and $\epsilon(C) = \epsilon_1(C)$. Suppose $C$ merges with $C'$ through $MEOE(C) = (u, v)$. Now node $r$ performs the followings.

- It selects the node with the larger ID between $u$ and $v$ as the leader $L(C'')$ of the resultant component $C'' = C \cup C'$.
- If $C'$ is active then $W(C'') = W(C) + W(C') + 2\epsilon(C)$. On the other hand if $C'$ is inactive then $W(C'') = W(C) + W(C') + \epsilon(C)$.

After that node $r$ broadcasts a message called $\langle merge(L(C), L(C'), L(C''), \epsilon(C), W(C''), MEOE(C))\rangle$ over $T_r$. From the received message $\langle merge(L(C), L(C'), L(C''), \epsilon(C), W(C''), MEOE(C))$, a node can easily verify whether it belongs to $C''$ or not. A node verifies it by comparing its current leader ID with the received parameters $L(C)$ and $L(C')$. Upon receiving $\langle merge(L(C), L(C'), L(C''), \epsilon(C), W(C''), MEOE(C))$, a node $v$ forwards $\langle merge\rangle$ to all of its child nodes (if any) in the $T_r$. However if $v \in C''$ then it also performs the following operations.

- it joins the new component $C''$ by updating the leader information to $L(C'')$.
- it updates the local variables $d_v$ and $W_v$ as per the rules described in the original D-PCST algorithm.
- if $MEOE(C) \in \delta(v)$ then it sets $ES(MEOE(C))$ to *branch*.
- if $v \in C'$ and the local variable $CS = inactive$ then $v$ locally sets $CS$ to *active*.

Since the broadcast of $\langle merge\rangle$ message over $T_r$ takes $h$ rounds, where $h$ is the height of $T_r$ and $h \leq D$, the round complexity of merging operation is $O(D)$. Similarly since $\langle merge\rangle$ message is sent once on each edge of $T_r$, the message complexity of the merging operation is $O(n)$.

□

By leveraging on $T_r$, the deactivation of a component $C$ can be performed as follows.

- node $r$ broadcasts a message called $\langle$*deactivate*$(L(C), \epsilon(C))\rangle$ over $T_r$.
- if $v \in C$ then it performs the following.
    - it updates the local variables $d_v$ to $d_v + \epsilon(C)$ and $W_v$ to $W_v + \epsilon(C)$.
    - it sets the local variable $CS$ to *inactive*.

Since the height of $T_r$ can be at most $D$, the deactivation requires $O(D)$ rounds. Furthermore the $\langle$*deactivate*$\rangle$ message is sent once on each edge of $T_r$. This ensures that the message complexity of a deactivation of a component is $O(n)$. Therefore we claim the following lemma.

**Lemma 5.6.3.** *Let $T_r$ be a BFS tree of the input graph $G$ rooted at the special node $r$. Then the deactivation of a component can be performed using $O(D)$ rounds and $O(n)$ messages.*

Note that if $CS(C)$ is inactive and $\epsilon_1(C) \neq \infty$ then $r$ performs a proceed operation. In proceed operation $r$ sends a $\langle$*proceed*$\rangle$ message to a node $L(C')$, where $MEOE(C) = (u, v)$, $u \in C$ and $v \in C'$ such that $C \neq C'$. It is obvious that by leveraging on $T_r$, node $r$ can send $\langle$*proceed*$\rangle$ to $L(C')$ using $O(D)$ rounds and $O(D)$ messages. Therefore the following lemma holds.

**Lemma 5.6.4.** *Let $T_r$ be a BFS tree of the input graph $G$ rooted at the special node $r$. Then the proceed operation can be performed using $O(D)$ rounds and $O(D)$ messages.*

If $C \neq C_r$ ($C_r$ is the root component) is an inactive component and $\epsilon_1(C) = \infty$ then $r$ performs a back operation. In back operation $r$ sends a $\langle$*back*$\rangle$ message to a node which sends a $\langle$*proceed*$\rangle$ to $C$ or a subcomponent of $C$. Whenever a node sends a $\langle$*proceed*$\rangle$ message to a node, it remembers the time of sending by using a time stamp. By applying a broadcast and convergecast over $T_r$, $r$ can find the node $v$ with the earliest time stamp in the network. This can be performed using $O(D)$ rounds and $O(n)$ messages. After that $r$ can send $\langle$*back*$\rangle$ to $v$ by using $T_r$. This can be performed using $O(D)$ rounds. It is obvious that the back operation can be performed using $O(D)$ rounds and $O(n)$ messages. Therefore we claim the following lemma.

**Lemma 5.6.5.** *Let $T_r$ be a BFS tree of the input graph $G$ rooted at the special node $r$. Then the back operation can be performed using $O(D)$ rounds and $O(n)$ messages.*

Note that each *proc_initiate*() method consists of finding $\epsilon(C)$, $MEOE(C)$ and one of the operations: merging, deactivation, proceed, back. Lemma 5.6.1 ensures that finding of $\epsilon(C)$ and $MEOE(C)$ can be accomplished using $O(D)$ rounds and $O(m)$ messages. It is obvious from Lemma 5.6.2, Lemma 5.6.3, Lemma 5.6.4, and Lemma 5.6.5 that each of the operations (merging, deactivation, proceed, and back) can be performed using $O(D)$ rounds and $O(n)$ messages. Therefore the following theorem holds.

**Theorem 5.6.6.** *The modified D-PCST algorithm performs a proc_initiate() method using $O(D)$ rounds and $O(m)$ messages.*

**Theorem 5.6.7.** *The growth phase of the modified D-PCST algorithm can be performed using $O(Dn)$ rounds and $O(mn)$ messages.*

*Proof.* Until the termination of the growth phase, the modified D-PCST algorithm generates the same number of calls to the *proc_initiate*() method as that of the original D-PCST algorithm. Therefore by using the Lemma 5.5.3 we claim that in the worst-case, the modified D-PCST algorithm generates $O(n)$ number of *proc_initiate*() method. Theorem 5.6.6 ensures that each such *proc_initiate*() method can be performed using $O(D)$ rounds and $O(m)$ messages. Since the modified D-PCST algorithm generates $O(n)$ number of *proc_initiate*() method, this guarantees that the overall round and message complexities of the growth phase of the modified D-PCST algorithm are $O(Dn)$ and $O(mn)$ respectively. □

If $\epsilon_1(C_r) = \infty$ then $r$ performs a pruning operation. Node $r$ can perform the pruning operation by broadcasting a $\langle prune \rangle$ message over $T_r$. Note that each node in the network receives a $\langle prune \rangle$ message after $O(h)$ rounds. This generates $n-1$ messages. Upon receiving a $\langle prune \rangle$ message, a node $v \notin C_r$ prunes itself by setting all of its incident edges to *basic*. Such a node is included in the Penalty part of the PCST.

Note that $C_r$ is a tree structure rooted at $r$. Therefore in $C_r$ pruning can be performed by applying a procedure called *strong pruning* introduced by Johnson et al. [72]. The strong pruning works as follows. It starts at the leaf nodes of the tree rooted at $r$. Suppose $nw(v)$ denotes the sum of prizes of all nodes minus the sum of weights of all the edges in a subtree rooted at $v$. Initially $nw(v) = p_v$ for each node $v$ in the input tree. At the beginning each leaf node $v$ sends its $nw(v)$ to its parent $u$. Upon receiving $nw(v)$, $u$ evaluates the condition $w_{(u,v)} \geq nw(v)$. If this condition holds then $u$ removes the edge $(u, v)$ and the subtree rooted at $v$ from the Steiner part. The prizes of all the nodes in the subtree rooted at $v$ contribute to the Penalty part of the PCST. On the other hand if $w_{(u,v)} < nw(v)$ then $u$ updates $nw(u)$ to $nw(u) + nw(v) - w_{(u,v)}$. Once $u$ performs the above procedure for all of its children, it reports the resultant $nw(u)$ to its parent. This principle is recursively followed by each node in the tree. In the worst-case, strong pruning generates at most $n - 1$ messages. Therefore it has a message complexity of $O(n)$. Considering every message arrives in one unit of time, the total time taken by the strong pruning is the height of the tree. Since the height of a tree can be at most $n - 1$, the round complexity of the strong pruning is $O(n)$. Therefore the following theorem holds.

**Theorem 5.6.8.** *The pruning phase of the modified D-PCST algorithm can be performed using $O(n)$ rounds and $O(n)$ messages.*

Recall that the modified D-PCST algorithm consists of two phases: growth and pruning. It is clear from the Theorem 5.6.7 and Theorem 5.6.8 that the overall round and the message complexities of the modified D-PCST algorithm are dominated by the growth phase. Therefore we claim that the modified D-PCST algorithm deterministically computes a $(2 - \frac{1}{n-1})$-approximate PCST using $O(Dn)$ rounds and $O(mn)$ messages in the CONGEST model of distributed computing.

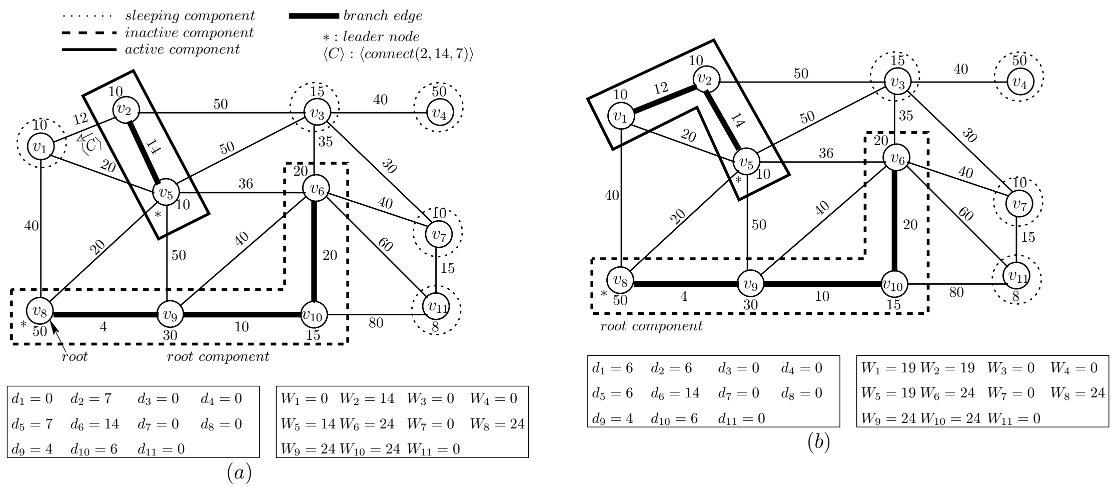
## 5.7 Summary

In this chapter we have proposed two algorithms for the PCST problem in the CONGEST model. First we have proposed D-PCST, a deterministic distributed algorithm that computes a PCST using $O(n^2)$ rounds and $O(mn)$ messages. The second one is a modification of the first one which computes a PCST using $O(Dn)$ rounds and $O(mn)$ messages. Both the algorithms achieve an approximation factor of $\left(2 - \frac{1}{n-1}\right)$. The proposed algorithms are based on the idea of preserving dual constraints in a distributed manner without relying on the knowledge of the global structure of the network. Both the algorithms require $O(\Delta \log n)$ bits of memory in each node.

## 5.A An illustrative example of the D-PCST algorithm

In this section we explain the working principle of the D-PCST algorithm with an example.

Figure 5.6 illustrates the merging of two components. Each node has a prize value that is shown next to the node. For example, the prize of node $v_1$ is 10. Similarly each



|  |  |  |  |
|---|---|---|---|
| $d_1 = 0$ | $d_2 = 7$ | $d_3 = 0$ | $d_4 = 0$ |
| $d_5 = 7$ | $d_6 = 14$ | $d_7 = 0$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 0$ |  |

|  |  |  |  |
|---|---|---|---|
| $W_1 = 0$ | $W_2 = 14$ | $W_3 = 0$ | $W_4 = 0$ |
| $W_5 = 14$ | $W_6 = 24$ | $W_7 = 0$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 0$ |  |

|  |  |  |  |
|---|---|---|---|
| $d_1 = 6$ | $d_2 = 6$ | $d_3 = 0$ | $d_4 = 0$ |
| $d_5 = 6$ | $d_6 = 14$ | $d_7 = 0$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 0$ |  |

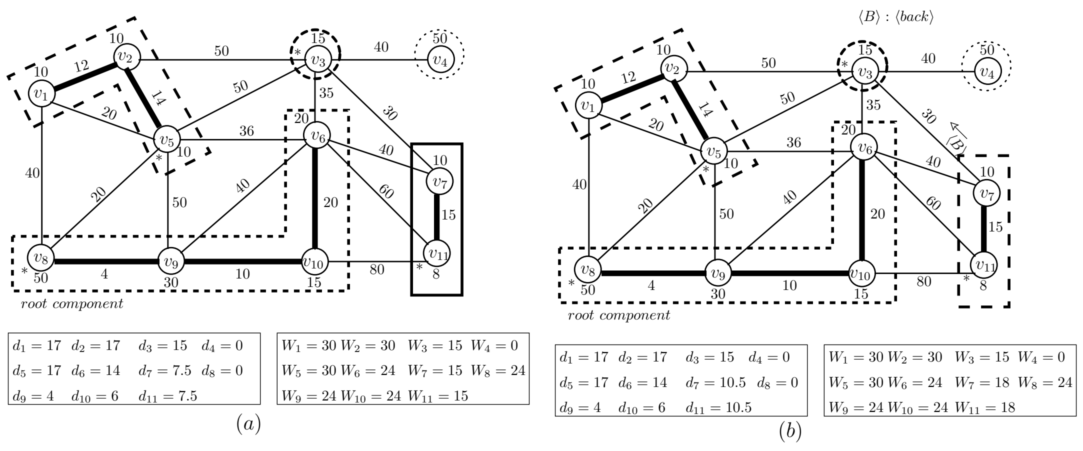
|  |  |  |  |
|---|---|---|---|
| $W_1 = 19$ | $W_2 = 19$ | $W_3 = 0$ | $W_4 = 0$ |
| $W_5 = 19$ | $W_6 = 24$ | $W_7 = 0$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 0$ |  |

**Figure 5.6:** *A case of merging operation. (a) state before merging of components $\{v_2, v_5\}$ and $\{v_1\}$. (b) state after merging.*

edge is labelled with an weight. Figure 5.6(a) shows the graph before the merging of two neighboring components $C = \{v_2, v_5\}$ which is in *active* state and $C' = \{v_1\}$ which is in *sleeping* state. The MEOE of the component $C$ is $(v_2, v_1)$ which gives $\epsilon_1(C) = -1$ (recall that $\epsilon_1(C) = \min_{v \in C} \{\epsilon_1(v)\}$). The leader node $v_5$ also computes $\epsilon_2(C) = 6$ (recall again that $\epsilon_2(C) = \sum_{v \in C} p_v - W(C)$). Hence $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C)) = \epsilon_1(C)$. So $v_2$ sends $\langle connect(v_2, 14, 7)\rangle$ on the MEOE to merge with $C'$. Upon receiving $\langle connect(v_2, 14, 7)\rangle$, $C'$ becomes *active* and finds $\epsilon_1(C') = -1$, $\epsilon_2(C') = 3$, and $(v_1, v_2)$ is the MEOE. Therefore $\epsilon(C') = \epsilon_1(C') = -1$ and it decides to merge with $C$. The new active component $\{v_1, v_2, v_3\}$ is shown in Figure 5.6(b). The rectangular box below the graph shows the values of local variables $d_i$ and $W_i$ for each $v_i \in V$. Here $W_i$ denotes $W(C)$ at node $v_i \in C$.

Figure 5.7 shows the deactivation of an active component $C = \{v_7, v_{11}\}$. In Figure 5.7(a) the leader of $C$ finds that its MEOE is $(v_7, v_3)$ which gives $\epsilon_1(C) = 7.5$. $C$ also computes its $\epsilon_2(C)$ which is equal to 3. Since $\epsilon(C) = \min(\epsilon_1(C), \epsilon_2(C)) = \epsilon_2(C)$, therefore the component $C$ deactivates itself. Each node of $C$ sets its local boolean variable *labelled_flag* = *true*. The graph after the deactivation of $C$ is shown in Fig 5.7(b).

Figure 5.8 shows the case of proceed operation. Since the state of the component $C = \{v_3\}$ is inactive therefore it has to send $\langle proceed\rangle$ to one of the neighboribg components to take further actions of the algorithm. The MEOE of $C$ is $(v_3, v_4)$ for which $\epsilon_1(C) = 10$. Since $C$ is an inactive component, therefore by *Property 2* of the algorithm it never computes its $\epsilon_2(C)$. The component $C$ sends $\langle proceed(15)\rangle$ (denoted by $\langle P(15)\rangle$



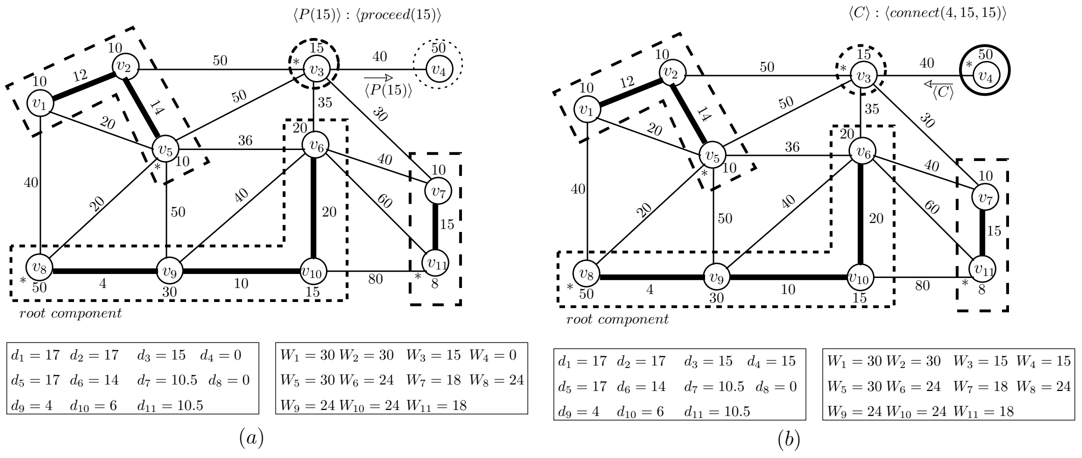
| | | | |
|---|---|---|---|
| $d_1 = 17$ | $d_2 = 17$ | $d_3 = 15$ | $d_4 = 0$ |
| $d_5 = 17$ | $d_6 = 14$ | $d_7 = 7.5$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 7.5$ | |

| | | | |
|---|---|---|---|
| $W_1 = 30$ | $W_2 = 30$ | $W_3 = 15$ | $W_4 = 0$ |
| $W_5 = 30$ | $W_6 = 24$ | $W_7 = 15$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 15$ | |

| | | | |
|---|---|---|---|
| $d_1 = 17$ | $d_2 = 17$ | $d_3 = 15$ | $d_4 = 0$ |
| $d_5 = 17$ | $d_6 = 14$ | $d_7 = 10.5$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 10.5$ | |

| | | | |
|---|---|---|---|
| $W_1 = 30$ | $W_2 = 30$ | $W_3 = 15$ | $W_4 = 0$ |
| $W_5 = 30$ | $W_6 = 24$ | $W_7 = 18$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 18$ | |

**Figure 5.7:** *A case of deactivation. (a) state before the deactivation of the active component* $\{v_7, v_{11}\}$*. (b) state after the deactivation .*



| | | | |
|---|---|---|---|
| $d_1 = 17$ | $d_2 = 17$ | $d_3 = 15$ | $d_4 = 0$ |
| $d_5 = 17$ | $d_6 = 14$ | $d_7 = 10.5$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 10.5$ | |

| | | | |
|---|---|---|---|
| $W_1 = 30$ | $W_2 = 30$ | $W_3 = 15$ | $W_4 = 0$ |
| $W_5 = 30$ | $W_6 = 24$ | $W_7 = 18$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 18$ | |

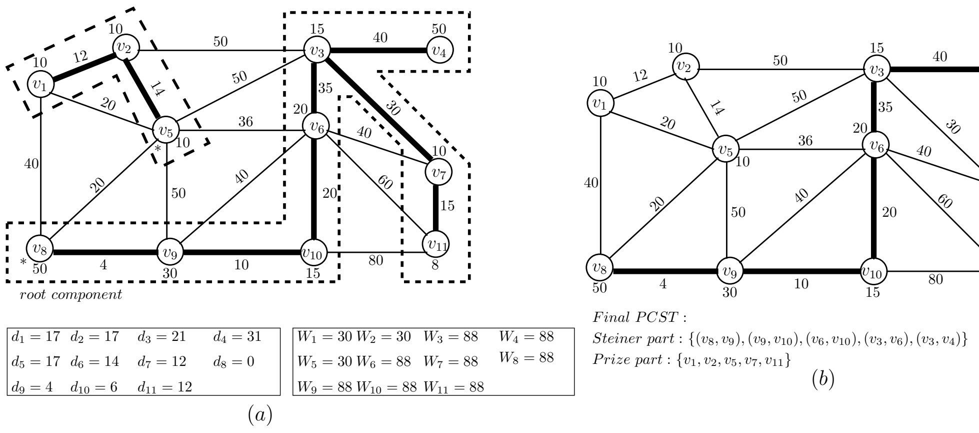
| | | | |
|---|---|---|---|
| $d_1 = 17$ | $d_2 = 17$ | $d_3 = 15$ | $d_4 = 15$ |
| $d_5 = 17$ | $d_6 = 14$ | $d_7 = 10.5$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 10.5$ | |

| | | | |
|---|---|---|---|
| $W_1 = 30$ | $W_2 = 30$ | $W_3 = 15$ | $W_4 = 15$ |
| $W_5 = 30$ | $W_6 = 24$ | $W_7 = 18$ | $W_8 = 24$ |
| $W_9 = 24$ | $W_{10} = 24$ | $W_{11} = 18$ | |

**Figure 5.8:** *A case of proceed operation. (a) state of sending* $\langle proceed(15)\rangle$ *by the inactive component* $\{v_3\}$*. (b) state after the component* $\{v_4\}$ *receives* $\langle proceed\rangle$*.*

in the figure) on its MEOE to the component $C' = \{v_4\}$ which is shown in Figure 5.8(a). Upon receiving $\langle P\rangle$, the *sleeping* component $C'$ becomes *active* and initializes its each of the local variables $d_4$ and $W_4$ to 15. Since $\epsilon(C') = \epsilon_1(C') = 10$, therefore $C'$ sends a connection request to $C$ which is shown in Figure 5.8(b).

Figure 5.9 shows the case of pruning operation performed in all components of the graph. In Figure 5.9(a), inside each of the non-root inactive components the state of each *branch* edge changes to *basic*. In the root component $C_r$, nodes $v_7$ and $v_{11}$ are pruned. The component $C = \{v_7, v_{11}\}$ was deactivated at some early stage of the growth phase of the algorithm. At $v_{11}$ the local variable *prize_flag* is set to *true* and $ES(v_{11}, v_7)$ is set to *basic*.



| | | | |
|---|---|---|---|
| $d_1 = 17$ | $d_2 = 17$ | $d_3 = 21$ | $d_4 = 31$ |
| $d_5 = 17$ | $d_6 = 14$ | $d_7 = 12$ | $d_8 = 0$ |
| $d_9 = 4$ | $d_{10} = 6$ | $d_{11} = 12$ | |

| | | | |
|---|---|---|---|
| $W_1 = 30$ | $W_2 = 30$ | $W_3 = 88$ | $W_4 = 88$ |
| $W_5 = 30$ | $W_6 = 88$ | $W_7 = 88$ | $W_8 = 88$ |
| $W_9 = 88$ | $W_{10} = 88$ | $W_{11} = 88$ | |

*Final PCST :*
*Steiner part :* $\{(v_8, v_9), (v_9, v_{10}), (v_6, v_{10}), (v_3, v_6), (v_3, v_4)\}$
*Prize part :* $\{v_1, v_2, v_5, v_7, v_{11}\}$

**Figure 5.9:** *A case of pruning phase. (a) state before pruning. (b) state after pruning phase which is the final solution to the PCST. The pruned components are* $\{v_1, v_2, v_5\}$ *and* $\{v_7, v_{11}\}$.

Similarly the node $v_7$ and its corresponding adjacent *branch* edges are also pruned from the root component. Figure 5.9(b) shows the state of the graph after the completion of the pruning phase which is the final solution to the PCST.

## 5.B Pseudo-code of the D-PCST algorithm

**D-PCST algorithm** for node $v$ upon receiving a message on $e \in \delta(v)$ or no message

```
1:  upon receiving no message
2:  call $initialization()$
3:  if $v = r$ then                                   ▷ Spontaneous awaking of the root node
4:      $CS \leftarrow inactive; root\_flag \leftarrow true; prize\_flag \leftarrow false;$
5:      call $proc\_initiate()$
6:  else
7:      $CS \leftarrow sleeping; root\_flag \leftarrow false; prize\_flag \leftarrow true;$
8:  end if

9:  procedure $initialization()$
10:     for each $e \in \delta(v)$ do
11:         $ES(e) \leftarrow basic; EPM(e) \leftarrow false;$          ▷ $EPM$ : Edge for Prune Message
12:     end for
13:     $d_v \leftarrow 0; W \leftarrow 0; leader\_flag \leftarrow false; labelled\_flag \leftarrow false;$
```

```
14:     $prune\_msg\_count \leftarrow 0; proceed\_flag \leftarrow false; received\_ts \leftarrow \infty; proceed\_in\_edge \leftarrow \emptyset;$
15: end procedure

16: procedure $proc\_initiate()$
17:     $SN \leftarrow find; find\_count \leftarrow 0; best\_epsilon \leftarrow \infty; best\_edge \leftarrow \emptyset;$
18:     $L \leftarrow v; TP \leftarrow 0; PF \leftarrow false; back\_edge \leftarrow \emptyset; TS \leftarrow \infty;$    ▷ $L$: Leader node
19:     for each $e \in \delta(v)$ do
20:         if $ES(e) = branch$ then
21:             send $\langle initiate(L, SN) \rangle$ on $e$
22:             $find\_count \leftarrow find\_count + 1;$ ▷ Count number of $\langle initiate \rangle$ that are sent
23:         end if
24:     end for
25:     if $SN = find$ then
26:         call $proc\_test()$
27:     end if
28: end procedure

29: upon receiving $\langle initiate(L, S) \rangle$ on edge $e$
30: $SN \leftarrow S; find\_count \leftarrow 0; best\_epsilon \leftarrow \infty; best\_edge \leftarrow \emptyset; L \leftarrow L;$
31: $TP \leftarrow 0; PF \leftarrow false; back\_edge \leftarrow \emptyset; in\_branch \leftarrow e;$
32: for each $e' \in \delta(v) : e' \neq e$ do
33:     if $ES(e') = branch$ then
34:          send $\langle initiate(L, S) \rangle$ on $e'$
35:         $find\_count \leftarrow find\_count + 1;$ ▷ Count the number of $\langle initiate \rangle$ that are sent
36:     end if
37: end for
38: if $S = find$ then
39:     call $proc\_test()$
40: end if

41: procedure $proc\_test()$
42:     $test\_count \leftarrow 0;$
43:     for each $e \in \delta(v)$ do
44:         if $ES(e) = basic$ or $ES(e) = refind$ then
45:             send $\langle test(L) \rangle$ on $e$
46:             $test\_count \leftarrow test\_count + 1;$    ▷ Count the number of $\langle test \rangle$ that are sent
47:         end if
48:     end for
49: end procedure
```

```
50: upon receiving ⟨$test(L')$⟩ on edge $e$
51: if $L = L'$ then
52:     send ⟨$reject$⟩ on $e$
53: else
54:     send ⟨$status(CS, d_v)$⟩ on $e$.
55: end if

56: upon receiving ⟨$status(NS, d_u)$⟩ on edge $e$
57: $test\_count \leftarrow test\_count - 1$;
58: if $CS = active$ and $NS = sleeping$ then
59:     $\epsilon_1 \leftarrow \frac{w_e - 2d_v}{2}$;
60: else if $CS = active$ and $NS = inactive$ then
61:     $\epsilon_1 \leftarrow w_e - d_v - d_u$;
62: else if $CS = inactive$ and $NS = sleeping$ then
63:     $\epsilon_1 \leftarrow w_e - 2d_v$;
64: else if $CS = inactive$ and $NS = inactive$ then
65:     if $ES(e) = refind$ then
66:         $\epsilon_1 \leftarrow w_e - d_v - d_u$;
67:     else
68:         $\epsilon_1 \leftarrow \infty$;
69:     end if
70: end if
71: if $\epsilon_1 < best\_epsilon$ then
72:     $best\_epsilon \leftarrow \epsilon_1; best\_edge \leftarrow e$;
73: end if
74: call $proc\_report()$

75: upon receiving ⟨$reject$⟩ on edge $e$
76: $test\_count \leftarrow test\_count - 1$;
77: $ES(e) \leftarrow rejected$;
78: if $proceed\_in\_edge = e$ then                ▷ The edge $e$ becomes a rejected edge
79:     $proceed\_in\_edge \leftarrow \emptyset; proceed\_flag \leftarrow false$;
80: end if
81: call $proc\_report()$

82: procedure $proc\_report()$
83:     if $find\_count = 0$ and $test\_count = 0$ then      ▷ Receives responses for each
    ⟨$initiate$⟩ and ⟨$test$⟩
84:         $SN \leftarrow found$;
85:         if $CS = active$ then
86:             $TP \leftarrow TP + p_v$          ▷ $TP$: (Total Prize) of the subtree rooted at $v$
87:         end if
88:         if $proceed\_flag = true$ then
89:             $PF \leftarrow true$;
```

```
90:             if $TS > received\_ts$ then
91:                 $TS \leftarrow received\_ts; back\_edge \leftarrow \emptyset$;
92:             end if
93:         end if
94:         if $in\_branch \neq \emptyset$ then
95:             send $\langle report(best\_epsilon, TP, PF, TS)\rangle$ on $in\_branch$
96:         else                                   ▷ leader/root node of the component
97:             call $proc\_merge\_or\_deactivate\_or\_proceed()$
98:         end if
99:     end if
100: end procedure

101: upon receiving $\langle report(\epsilon_1, T, P, temp\_ts)\rangle$ on edge $e$
102: $find\_count = find\_count - 1$;
103: if $P = true$ then
104:     $PF \leftarrow true$;
105:     if $TS > temp\_ts$ then
106:         $TS \leftarrow temp\_ts; back\_edge \leftarrow e$;
107:     end if
108: end if
109: if $CS = active$ then
110:     $TP \leftarrow TP + T$;
111: end if
112: if $\epsilon_1 < best\_epsilon$ then
113:     $best\_epsilon \leftarrow \epsilon_1; best\_edge \leftarrow e$;
114: end if
115: call $proc\_report()$

116: procedure $proc\_merge\_or\_deactivate\_or\_proceed()$
117:     $\epsilon_1 \leftarrow best\_epsilon$
118:     if $root\_flag = false$ and $CS = active$ then
119:         $\epsilon_2 \leftarrow TP - W$;
120:         if $\epsilon_1 < \epsilon_2$ then
121:             $best\_epsilon \leftarrow \epsilon_1$
122:         else
123:             $best\_epsilon \leftarrow \epsilon_2$
124:         end if
125:         if $best\_epsilon = \epsilon_2$ then
126:             $CS \leftarrow inactive; d_v \leftarrow d_v + \epsilon_2; W \leftarrow W + \epsilon_2; labelled\_flag \leftarrow true$;
127:             $deactivate\_flag \leftarrow true$;      ▷ $deactivate\_flag$ is a temporary variable
128:             send $\langle update\_info(\epsilon_2, root\_flag, deactivate\_flag, W)\rangle$ on all $e \in \delta(v)$ such that $ES(e) = branch$
129:             call $proc\_initiate()$                 ▷ Compute $\epsilon_1$ to send $\langle proceed\rangle$
```

```
130:         else                                ▷ Start the merge procedure at the leader node
131:             if $ES(best\_edge) = branch$ then
132:                 send $\langle merge(best\_epsilon)\rangle$ on $best\_edge$
133:             else
134:                 send $\langle connect(v, W, d_v)\rangle$ on $best\_edge$;
135:             end if
136:         end if
137:     else if $CS = inactive$ then
138:         if $(\epsilon_1 = \infty)$ then
139:             if $TS = \infty$ then
140:                 if $v = r$ then                 ▷ Starts of pruning phase at the root node $r$.
141:                     for each $e \in \delta(v)$ do
142:                         if $ES(e) = branch$ or $EPM(e) = true$ then
143:                             send $\langle prune\rangle$ on $e$
144:                         end if
145:                         if $ES(e) = branch$ then
146:                             $prune\_msg\_count \leftarrow prune\_msg\_count + 1$;
147:                         end if
148:                     end for
149:                 end if
150:             else                                            ▷ Start of sending $\langle back\rangle$
151:                 if $back\_edge \neq \emptyset$ then
152:                     send $\langle back\rangle$ on $back\_edge$
153:                 else if $back\_edge = \emptyset$ and $proceed\_flag = true$ then
154:                     send $\langle back\rangle$ on $proceed\_in\_edge$
155:                     $proceed\_in\_edge \leftarrow \emptyset; proceed\_flag = false$;
156:                 end if
157:             end if
158:         else if $(\epsilon_1 \neq \infty)$ then
159:             send $\langle proceed(d_v)\rangle$ on $best\_edge$
160:             if $ES(best\_edge) = basic$ then
161:                 $EPM(best\_edge) \leftarrow true$;
162:             end if
163:             if $ES(best\_edge) = refind$ then
164:                 $ES(best\_edge) \leftarrow basic$;
165:             end if
166:         end if
167:     end if
168: end procedure

169: upon receiving $\langle merge(\epsilon)\rangle$ on edge $e$
170: if $ES(best\_edge) = branch$ then           ▷ Receiving node is an intermediate node
171:     send $\langle merge(\epsilon)\rangle$ on $best\_edge$
```

```
172: else                                                ▷ Receiving node is a frontier node
173:     send $\langle connect(v, W, d_v)\rangle$ on $best\_edge$;
174: end if

175: upon receiving $\langle back\rangle$ on edge $e$
176: if $back\_edge \neq \emptyset$ then
177:     send $\langle back\rangle$ on $back\_edge$
178: else if $back\_edge = \emptyset$ and $proceed\_flag = true$ then
179:     send $\langle back\rangle$ on $proceed\_in\_edge$
180:     $proceed\_in\_edge \leftarrow \emptyset; proceed\_flag = false$;
181: else if $back\_edge = \emptyset$ and $in\_branch \neq \emptyset$ then
182:     send $\langle back\rangle$ on $in\_branch$
183: else
184:     call $proc\_initiate()$
185: end if

186: upon receiving $\langle proceed(d_k)\rangle$ on edge $e$
187: if $ES(e) = branch$ and $in\_branch = e$ then
188:     send $\langle proceed(d_k)\rangle$ on $best\_edge$
189:     if $ES(best\_edge) = basic$ then
190:         $EPM(best\_edge) \leftarrow true$;
191:     end if
192:     if $ES(best\_edge) = refind$ then
193:         $ES(best\_edge) \leftarrow basic$;
194:     end if
195: else if $ES(e) = basic$ then
196:     $proceed\_flag \leftarrow true; proceed\_in\_edge \leftarrow e$;
197:     if $CS = sleeping$ then
198:         call $wakeup(d_k)$
199:     else if $CS = inactive$ then
200:         if $in\_branch \neq \emptyset$ then
201:             send $\langle proceed(d_k)\rangle$ on $in\_branch$
202:         end if
203:     end if
204: else if $ES(e) = branch$ and $in\_branch \neq e$ then
205:     if $in\_branch \neq \emptyset$ then
206:         send $\langle proceed(d_k)\rangle$ on $in\_branch$
207:     else                                            ▷ Receiving node is the leader node
208:         call $proc\_initiate()$
209:     end if
210: end if

211: procedure $wakeup(d_k)$
212:     $CS \leftarrow active; d_v \leftarrow d_k; W \leftarrow d_k$;
```

```
213:     call $proc\_initiate()$
214: end procedure

215: upon receiving $\langle connect(NID, WN, d_k)\rangle$ on edge $e$        ▷ $NID$: Neighbor's ID,
     $WN$: Weight of Neighboring component
216: if $CS = sleeping$ then
217:     $CS \leftarrow active; d_v \leftarrow d_k; W \leftarrow d_k;$
218:     $\epsilon_1 \leftarrow \frac{w_e - d_v - d_k}{2}; \epsilon_2 \leftarrow p_v - W;$
219:     if $\epsilon_1 < \epsilon_2$ then
220:         if $v > NID$ then
221:             $leader\_flag \leftarrow true;$
222:         else
223:             $leader\_flag \leftarrow false;$
224:         end if
225:         $d_v \leftarrow d_v + \epsilon_1; W \leftarrow W + WN + 2 \times \epsilon_1; ES(e) \leftarrow branch;$
226:         send $\langle accept(leader\_flag, root\_flag, W)\rangle$ on $e$
227:         if $leader\_flag \leftarrow true$ then
228:             call $proc\_initiate()$
229:         end if
230:     else                                                              ▷ $\epsilon_1 \geq \epsilon_2$
231:         $CS \leftarrow inactive; W \leftarrow W + \epsilon_2; d_v \leftarrow d_v + \epsilon_2; labelled\_flag \leftarrow true;$
232:         send $\langle refind\_epsilon\rangle$ on $e$
233:     end if
234: else if $CS = inactive$ then
235:     if $root\_flag = true$ then
236:         $leader\_flag \leftarrow true;$
237:     else
238:         $CS \leftarrow active;$
239:         if $v > NID$ then
240:             $leader\_flag \leftarrow true;$
241:         else
242:             $leader\_flag \leftarrow false;$
243:         end if
244:     end if
245:     $\epsilon_1 = w_e - d_v - d_k; W \leftarrow W + WN + \epsilon_1;$
246:     $deactivate\_flag = false;$                   ▷ $deactivate\_flag$ is a temporary variable
247:     send $\langle update\_info(0, root\_flag, deactivate\_flag, W)\rangle$ on all $e' \in \delta(v) : e' \neq e$
     and $ES(e') = branch$
248:     $ES(e) \leftarrow branch;$
249:     send $\langle accept(leader\_flag, root\_flag, W)\rangle$ on $e;$
250:     if $(leader\_flag = true)$ then
251:         call $proc\_initiate()$
252:     end if
253: end if
```

```
254: upon receiving $\langle refind\_epsilon \rangle$ on edge $e$
255: if $ES(e) = basic$ then
256:     $ES(e) \leftarrow refind$;
257: else
258:     if $in\_branch \neq \emptyset$ then
259:         send $\langle refind\_epsilon \rangle$ on $in\_branch$
260:     else
261:         call $proc\_initiate()$
262:     end if
263: end if

264: upon receiving $\langle accept(LF, RF, TW) \rangle$ on edge $e$        ▷ $TW$: Total Weight
265: $ES(e) \leftarrow branch; root\_flag \leftarrow RF; d_v \leftarrow d_v + best\_epsilon; W \leftarrow TW$;
266: if $RF = true$ then
267:     $CS \leftarrow inactive; prize\_flag = false$;
268: else
269:     $CS \leftarrow active$;
270: end if
271: if $proceed\_in\_edge = e$ and $proceed\_flag = true$ then
272:     $proceed\_in\_edge \leftarrow \emptyset; proceed\_flag \leftarrow false$;
273: end if
274: $deactivate\_flag = false$;        ▷ $deactivate\_flag$ is a temporary variable
275: send $\langle update\_info(best\_epsilon, root\_flag, deactivate\_flag, TW) \rangle$ on all $e' \in \delta(v)$ :
     $e' \neq e$ and $ES(e') = branch$
276: if $LF = false$ then
277:     call $proc\_initiate()$
278: end if

279: upon receiving $\langle update\_info(EV, RF, DF, TW) \rangle$ on edge $e$
280: if $RF = true$ and $DF = false$ then
281:     $CS \leftarrow inactive; prize\_flag \leftarrow false$;
282: else if $RF = false$ and $DF = true$ then
283:     $CS \leftarrow inactive; labelled\_flag \leftarrow true$;
284: else if $RF = false$ and $DF = false$ then
285:     $CS \leftarrow active$;
286: end if
287: $root\_flag \leftarrow RF; d_v \leftarrow d_v + EV; W \leftarrow TW$;
288: send $\langle update\_info(EV, RF, DF, TW) \rangle$ on all $e' \in \delta(v) : e' \neq e$ and $ES(e') = branch$
289: if $v = r$ then        ▷ $r$ is the root node
290:     call $proc\_initiate()$
291: end if
292: $ES(e) \leftarrow basic$;
```

```
293: upon receiving $\langle prune \rangle$ on edge $e$
294: if $root\_flag = true$ then                          ▷ Pruning inside the root component
295:     if ($labelled\_flag = true$) and ($ES(e') = basic$ for each $e' \in \delta(v) : e' \neq e$) then
296:         $prize\_flag \leftarrow true; root\_flag \leftarrow false$;
297:         send $\langle backward\_prune \rangle$ on $e$
298:     else
299:         for each $e' \in \delta(v) : e' \neq e$ do
300:             if $ES(e') = branch$ or $EPM(e') = TRUE$ then
301:                 send $\langle prune \rangle$ on $e'$
302:                 if $ES(e') = branch$ then
303:                     $prune\_msg\_count \leftarrow prune\_msg\_count + 1$;
304:                 end if
305:             end if
306:         end for
307:     end if
308: else                                        ▷ Pruning inside a non-root inactive component
309:     for each $e' \in \delta(v) : e' \neq e$ do
310:         if $ES(e') = branch$ or $EPM(e') = TRUE$ then
311:             send $\langle prune \rangle$ on $e'$
312:             if $ES(e') = branch$ then
313:                 $ES(e') \leftarrow basic$;
314:             end if
315:         end if
316:     end for
317: end if

318: upon receiving $\langle backward\_prune \rangle$ on edge $e$
319: $prune\_msg\_count \leftarrow prune\_msg\_count - 1; ES(e) \leftarrow basic$;
320: if $labelled\_flag = true$ and $prune\_msg\_count = 0$ then
321:     if $in\_branch \neq \emptyset$ then
322:         $prize\_flag \leftarrow true; root\_flag \leftarrow false$;
323:         send $\langle backward\_prune \rangle$ on $in\_branch$
324:         $ES(in\_branch) \leftarrow basic$;
325:     end if
326: end if
```

# 6 Conclusions and Future Perspectives

In this chapter we summarize the work done, highlight the contributions, and suggest the directions for possible future work.

## 6.1 Conclusions

In this thesis we have proposed a set of distributed algorithms for two combinatorial optimization problems: the Steiner tree (ST) and the prize-collecting Steiner tree (PCST). In particular we have proposed the following algorithms.

1. **DST**. A deterministic distributed algorithm for ST in the CONGEST model.
2. **Modified DST**. It is a modified version of the DST algorithm.
3. **STCCM-A**. A deterministic distributed algorithm for ST in the CONGESTED CLIQUE model.
4. **STCCM-B**. A modified version of the STCCM-A algorithm.
5. **D-PCST**. A primal-dual based deterministic distributed algorithm for PCST in the CONGEST model.

6. **Modified D-PCST**. A modified version of the D-PCST algorithm.

In Chapter 3 we presented the DST and the modified DST. The DST algorithm computes a $2(1-1/\ell)$-approximate ST with the round and message complexities of $O(S+\sqrt{n}\log^* n)$ and $O(Sm + n^{3/2})$ respectively. The round complexity of the DST algorithm is better than the best known round complexity of the ST construction known so far [90]. The modified DST algorithm computes a $2(1-1/\ell)$-approximate ST with the round and message complexities of $\tilde{O}(S+\sqrt{n})$ and $\tilde{O}(Sm)$ respectively. The polylogarithmic factors involved with the round and message complexities of the modified DST algorithm are $O(\log n)$ and $O(\log^2 n)$ respectively. The modified DST algorithm improves the message complexity of the DST algorithm by dropping the additive term of $O(n^{3/2})$ at the expense of a logarithmic multiplicative factor in the round complexity. We also provided the proof of correctness for both the algorithms.

In Chapter 4 we studied the ST problem in the CONGESTED CLIQUE model of distributed computing and presented two algorithms: STCCM-A and STCCM-B. The STCCM-A algorithm computes a $2(1-1/\ell)$-approximate ST using $\tilde{O}(n^{1/3})$ rounds and $\tilde{O}(n^{7/3})$ messages. The polylogarthmic factor involved with each of the round and message complexities is $\log n$. The STCCM-B algorithm computes a $2(1-1/\ell)$-approximate ST using $O(S + \log\log n)$ rounds and $O(Sm+n^2)$ messages. Proof of correctness are given for both the algorithms. For graphs with $S = \omega(n^{1/3}\log n)$, the STCCM-A algorithm performs better than the STCCM-B algorithm in terms of the round complexity. On the other hand, for graphs with $S = \tilde{o}(n^{1/3})$, the STCCM-B algorithm outperforms the STCCM-A algorithm in terms of the round complexity. To the best of our knowledge, till date, this is the first work to study the ST problem in the CONGESTED CLIQUE model of distributed computing.

Finally, in Chapter 5 we presented two deterministic distributed algorithms for the PCST in the CONGEST model: D-PCST and modified D-PCST. Both the algorithms are based on the primal-dual technique, appropriately tailored for the distributed setting to achieve an approximation factor of $\left(2-\frac{1}{n-1}\right)$. The D-PCST algorithm computes a PCST using $O(n^2)$ rounds and $O(mn)$ messages. The modified D-PCST algorithm computes a PCST using $O(Dn)$ rounds and $O(mn)$ messages. For networks with constant or small unweighted diameter ($D = o(n)$), the modified D-PCST algorithm performs better than the original D-PCST algorithm in terms of the round complexity. Moreover both the algorithms preserve the dual constraints in a distributed manner without relying on the knowledge of the global structure of the network. Furthermore both the algorithms require $O(\Delta \log n)$ bits of memory in each node, where $\Delta$ is the maximum degree of a node in the graph. We also provided the proof of correctness for both the algorithms.

## 6.2 Future Perspectives

The work reported in the chapters of this thesis provide ample scope and promulgate several clear directions for future research endeavors. For instance one can think of designing a randomized version of each of the algorithms proposed in this thesis to improve the round and message complexities without compromising the approximation factor. We note that all the algorithms proposed in this thesis are deterministic in nature.

Elkin [39] showed that approximating MST within any constant factor on graphs of small unweighted diameter ($D = O(\sqrt{n})$) requires $\Omega(\sqrt{n/B})$ rounds (assuming $B$ bits can be sent through each edge in each round). Das Sarma et al. [31] achieved an unconditional lower bound on round complexity of the MST problem and showed that approximating MST within any constant factor requires $\Omega(D + \sqrt{n/(B \log n)})$ rounds. The well-known lower bound on message complexity to find an exact MST is $\Omega(m)$ [86]. Since the ST problem is a generalization of the MST problem, the above lower bounds of the MST problem are also applicable to the ST problem. Recently Byrka et al. [20] showed that in the centralized setting, the ST problem can be approximated upto a factor of $\ln(4) + \epsilon \approx 1.386 + \epsilon$ (for any constant $\epsilon > 0$) of the optimal. Therefore, there are open research directions of improvement of the approximation factor, the round complexity, and the message complexity of the ST construction in the CONGEST model of distributed computing.

For special class of graphs, the ST problem has been extensively studied in the centralized setting [19, 21, 28, 121]. In particular, for planar graphs, Borradaile et al. [19] proposed a PTAS for ST problem whose running time is $O(n \log n)$. Recently Byrka et al. [21] presented a PTAS for the ST problem which holds for *map graphs*. However these results hold for centralized setting only. Therefore it will be interesting to investigate the technique that we used in the DST algorithm, in a direction of obtaining a distributed version of PTAS of Borradaile et al. [19] for planar graphs or Byrka et al. [21] for map graphs.

The work presented in Chapter 4 of this thesis is the first one to study the ST problem in the CONGESTED CLIQUE model and is still in nascent form. The best known deterministic and randomized round complexities for MST construction in the CONGESTED

CLIQUE model are $O(\log\log n)$ [96] and $O(1)$ [73] respectively. Pemmaraju and Sardeshmukh [116] showed that in the CONGESTED CLIQUE model, an MST can be computed using $o(m)$ messages with the round complexity of $O(\log^* n)$ with high probability. Since the ST problem is a generalized version of the MST problem, we believe that it is also possible to achieve the above results for ST construction in the CONGESTED CLIQUE model. Therefore there are ample scopes on further improvement of the approximation factor, the round and message complexities of the ST construction in the CONGESTED CLIQUE model of distributed computing.

The work in Chapter 5 of this thesis presented two algorithms for the PCST problem in the CONGEST model of distributed computing, which are based on the techniques of primal-dual and distributed preservation of the dual constraints. We believe that our techniques can find further applications in obtaining memory-efficient distributed versions of primal-dual based algorithms for other tree problems. In addition, our algorithms, being distributed in nature, can possibly be adapted to tolerate local changes (node or link additions or deletions, weight changes), yielding algorithms of low incremental complexities for the fully dynamic setting. Since no algorithm is known so far for that setting this might be a meaningful approach. Finally, we believe that D-PCST and the modified D-PCST can serve as a first step and a basis for further improvements of round complexity, message complexity, as well as of the approximation ratio for distributed PCST. In particular, we would like to investigate the applicability of our techniques in the direction of obtaining a distributed version of the PTAS of Bateni et al. [11] for PCST in planar graphs; such a result would be of great theoretical and practical interest.

# Disseminations out of this Work

## Journals

1. **Parikshit Saikia** and Sushanta Karmakar. "Distributed approximation algorithms for Steiner tree in the CONGESTED CLIQUE", *International Journal of Foundations of Computer Science* (IJFCS), Vol. 31, No. 7, 2020.

2. **Parikshit Saikia**, Sushanta Karmakar, and Aris Pagourtzis. "Primal-Dual based distributed approximation algorithm for prize-collecting Steiner tree", *Discrete Mathematics, Algorithms and Applications* (DMAA). [Accepted on August 6, 2020, Online ready]

3. **Parikshit Saikia** and Sushanta Karmakar. "Improved distributed approximation for Steiner tree in the CONGEST model", *Journal of Parallel and Distributed Computing* (JPDC). [Under Review]

## Conferences

1. **Parikshit Saikia** and Sushanta Karmakar, "A Simple $2(1 - 1/\ell)$-factor Distributed Approximation Algorithm for Steiner Tree in the CONGEST model", *Proceedings of the 20th International Conference on Distributed Computing and Networking* (ICDCN), January 2019, Bangalore, India.

2. **Parikshit Saikia** and Sushanta Karmakar, "$2(1 - 1/\ell)$-factor Steiner tree approximation in $\tilde{O}(n^{1/3})$ rounds in the CONGESTED CLIQUE", *The Seventh International Symposium on Computing and Networking* (CANDAR), November 2019, Nagasaki, Japan. [Best Paper Award]

3. **Parikshit Saikia** and Sushanta Karmakar, "Round-Message Trade-off in Distributed Steiner Tree Construction in the CONGEST model", *16th International Conference on Distributed Computing and Internet Technology* (ICDCIT), January 2020, Bhubaneswar, India.

# References

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM journal of Computing*, 24(3):440–456, 1995. [Pg.11], [Pg.28]

[2] E. Álvarez Miranda, A. Candia, X. Chen, X. Hu, and B. Li. Efficient Algorithms for the Prize Collecting Steiner Tree Problems with Interval Data. *International Conference on Algorithmic Applications in Management*, pages 13–24, 2010. [Pg.28], [Pg.30]

[3] E. Álvarez Miranda, I. Ljubić, and P. Toth. Exact approaches for solving robust prize-collecting Steiner tree problems. *European Journal of Operational Research*, 229(3):599–612, 2013. [Pg.30]

[4] A. Archer, M. H. Bateni, and M. T. Hajiaghayi. Improved Approximation Algorithms for Prize-Collecting Steiner Tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011. [Pg.9], [Pg.11], [Pg.28], [Pg.30], [Pg.91], [Pg.97]

[5] B. Awerbuch, O. Goldreich, R. Vainish, and D. Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990. [Pg.viii], [Pg.7]

[6] N. Bacrach, K. Censor-Hillel, M. Dory, Y. Efron, D. Leitersdorf, and A. Paz. Hardness of distributed optimization. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 238–247, 2019. [Pg.9], [Pg.92]

[7] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989. [Pg.28]

[8] R. Bar-Yehuda, E. Kantor, S. Kutten, and D. Rawitz. Growing Half-Balls: Minimizing Storage and Communication Costs in CDNs. pages 416–427, 2012. [Pg.4], [Pg.6]

[9] L. Barenboim, M. Elkin, and F. Kuhn. Distributed $(\delta + 1)$-coloring in linear (in $\delta$) time. *SIAM J. Comput.*, 43(1):72–95, 2014. [Pg.1]

[10] M. Bateni, C. Chekuri, A. Ene, M. Hajiaghayi, N. Korula, and D. Marx. Improved combinatorial algorithms for facility location problems. *ACM-SIAM symposium on Discrete Algorithms*, pages 1028–1049, 2011. [Pg.28]

[11] M. Bateni, C. Chekuri, A. Ene, M. T. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1028–1049, 2011. [Pg.29], [Pg.142]

[12] F. Bauer and A. Varma. Distributed Algorithms for Multicast Path Setup in Data Networks. *IEEE/ACM Transaction Networks*, 4(2):181–191, 1996. [Pg.22], [Pg.23], [Pg.24], [Pg.25]

[13] M. Bauer, G. W. Klau, and K. Reinert. Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics*, 8(271), 2007. [Pg.5]

[14] P. Berman and V. Ramaiyer. Improved Approximation for the Steiner Problem. *Journal of Algorithms*, 9:381–408, 1994. [Pg.19], [Pg.20]

[15] A. Berns, J. Hegeman, and S. V. Pemmaraju. Super-fast Distributed Algorithms for Metric Facility Location. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, ICALP'12, pages 428–439, 2012. [Pg.8], [Pg.68]

[16] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993. [Pg.28], [Pg.96], [Pg.97]

[17] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 67–74, 2007. [Pg.18]

[18] A. Borchers and D. Du. The k-Steiner ratio in graphs. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 641–649, 1995. [Pg.19]

[19] G. Borradaile, C. Kenyon-Mathieu, and P. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1285–1294, 2007. [Pg.21], [Pg.141]

[20] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An Improved LP-based Approximation for Steiner Tree. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 583–592, 2010. [Pg.10], [Pg.19], [Pg.20], [Pg.30], [Pg.68], [Pg.141]

[21] J. Byrka, M. Lewandowski, S. M. Meesum, J. Spoerhase, and S. Uniyal. PTAS for Steiner tree on map graphs. *https://arxiv.org/abs/1912.00717*, Dec 2019. [Pg.21], [Pg.141]

[22] S. Canuto, M. Resende, and C. Ribeiro. Local search with perturbation for prize-collecting Sreiner tree problem in graphs. *Networks*, 38(1):50–58, 2001. [Pg.31]

[23] K. Censor-Hillel, M. Dory, J. H. Korhonen, and D. Leitersdorf. Fast approximate shortest paths in the congested clique. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 74–83, 2019. [Pg.74]

[24] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela. Algebraic Methods in the Congested Clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 143–152, 2015. [Pg.8], [Pg.68], [Pg.72], [Pg.73], [Pg.74], [Pg.75]

[25] D. Chakrabarty, N. R. Devanur, and V. V. Vazirani. New Geometry-Inspired Relaxations and Algorithms for the Metric Steiner Tree Problem. *Integer Programming and Combinatorial Optimization (IPCO)*, pages 344–358, 2008. [Pg.20], [Pg.21]

[26] P. Chalermsook and J. Fakcharoenphol. Simple Distributed Algorithms for Approximating Minimum Steiner Trees. In *Proceedings of the 11th Annual International*

*Conference on Computing and Combinatorics*, COCOON'05, pages 380–389, 2005. [Pg.23], [Pg.26]

[27] G. H. Chen, M. E. Houle, and M. T. Kuo. The Steiner problem in distributed computing systems. *Information Sciences*, 74(1-2):73–96, 1993. [Pg.22], [Pg.23], [Pg.42], [Pg.59]

[28] M. Chlebík and J. Chlebíková. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207 – 214, 2008. [Pg.20], [Pg.21], [Pg.141]

[29] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. A Faster Implementation of the Goemans-Williamson Clustering Algorithm. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'01, pages 17–25, 2001. [Pg.28], [Pg.29]

[30] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, July 1986. [Pg.63]

[31] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 363–372, 2011. [Pg.2], [Pg.7], [Pg.9], [Pg.92], [Pg.141]

[32] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008. [Pg.68]

[33] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Muller. Identifying functional modules in protein-protein interaction networks: an integrated exact approach. *Intelligent Systems for Molecular Biology (ISMB)*, 24(13):119–141, 2008. [Pg.6]

[34] D. Dolev, C. Lenzen, and S. Peled. "Tri, Tri Again": Finding Triangles and Small Subgraphs in a Distributed Setting. In *Proceedings of the 26th International Conference on Distributed Computing*, DISC'12, pages 195–209, 2012. [Pg.8], [Pg.68]

[35] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971. [Pg.17], [Pg.18]

[36] A. Drucker, F. Kuhn, and R. Oshman. The communication complexity of distributed task allocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 67–76, 2012. [Pg.8], [Pg.68]

[37] A. Drucker, F. Kuhn, and R. Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 367–376, 2014. [Pg.68]

[38] M. Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72(8):1282 – 1308, 2006. [Pg.viii], [Pg.7]

[39] M. Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal of Computing*, 36(2):433–456, 2006. [Pg.7], [Pg.92], [Pg.141]

[40] M. Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC '17, pages 757–770, 2017. [Pg.1]

[41] M. Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 157–163, 2017. [Pg.viii], [Pg.7], [Pg.8], [Pg.12], [Pg.39], [Pg.62], [Pg.63], [Pg.64], [Pg.65]

[42] G. Even, M. Ghaffari, and M. Medina. Distributed set cover approximation: Primal-dual with optimal locality. In *32nd International Symposium on Distributed Computing, DISC 2018*, pages 22:1–22:14, 2018. [Pg.10], [Pg.14], [Pg.92], [Pg.93]

[43] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(4):485–497, 2004. [Pg.27]

[44] P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina. Primal-Dual Approximation Algorithms for the Prize-Collecting Steiner Tree Problem. *Information Processing Letter*, 103(5):195–202, 2007. [Pg.11], [Pg.28], [Pg.29], [Pg.31], [Pg.96], [Pg.97]

[45] D. Fernandez-Baca. The Perfect Phylogeny Problem. *Combinatorial Optimization*, 11:203–234, 2004. [Pg.6]

[46] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory*, SWAT '71, pages 129–131, 1971. [Pg.72], [Pg.74]

[47] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1150–1162, 2012. [Pg.1]

[48] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, and X. Wang. Dynamic Programming for Minimum Steiner Trees. *Theory of Computing Systems*, 41(3):493–500, 2007. [Pg.17], [Pg.18]

[49] B. Fuchs, W. Kern, and X. Wang. Speeding up the Dreyfus–Wagner algorithm for minimum Steiner trees. *Mathematical Methods of Operations Research*, 66(1):117–125, 2007. [Pg.17], [Pg.18]

[50] R. G. Gallager, P. A. Humblet, and P. M. Spira. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems*, 1:66–77, 1983. [Pg.viii], [Pg.7], [Pg.26], [Pg.35], [Pg.53]

[51] J. A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal of Computing*, 27(1):302–316, 1998. [Pg.7], [Pg.37], [Pg.38], [Pg.53], [Pg.62]

[52] L. Gatani, G. Lo Re, and S. Gaglio. An efficient distributed algorithm for generating and updating multicast trees. *Parallel Computing*, 32(11-12):777–793, 2006. [Pg.22], [Pg.23], [Pg.26]

[53] J. Gehweiler, C. Lammersen, and C. Sohler. A Distributed O(1)-Approximation Algorithm for the Uniform Facility Location Problem. *Algorithmica*, 68(3):643–670, 2014. [Pg.8], [Pg.68]

[54] J. Geunes, R. Levi, H. E. Romeijn, and D. B. Shmoys. Approximation algorithms for supply chain planning and logistics problems with market choice. *Mathematical Programming*, 130(1):85–106, Nov 2011. [Pg.30], [Pg.91]

[55] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 270–277, 2016. [Pg.1]

[56] M. Ghaffari and M. Parter. MST in Log-Star Rounds of Congested Clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 19–28, 2016. [Pg.8], [Pg.68], [Pg.71], [Pg.78]

[57] E. N. Gilbert and H. O. Pollak. Steiner Minimal Trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968. [Pg.19], [Pg.20], [Pg.33]

[58] M. X. Goemans and D. E. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Applied Mathematics*, 24(2):296–317, 1995. [Pg.11], [Pg.14], [Pg.20], [Pg.28], [Pg.29], [Pg.31], [Pg.35], [Pg.91], [Pg.93], [Pg.95], [Pg.96], [Pg.117]

[59] F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. Primal-dual based distributed algorithms for vertex cover with semi-hard capacities. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 118–125, 2005. [Pg.10], [Pg.14], [Pg.92], [Pg.93]

[60] F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM J. Comput.*, 38(3):825–840, May 2008. [Pg.10], [Pg.14], [Pg.92], [Pg.93]

[61] S. Gueron and R. Tessler. The Fermat-Steiner Problem. *The American Mathematical Monthly (May, 2002)*, 109(5):443–451, 2002. [Pg.2]

[62] B. Haeupler, D. E. Hershkowitz, and D. Wajc. Round- and message-optimal distributed graph algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 119–128, 2018. [Pg.viii], [Pg.7], [Pg.8], [Pg.12], [Pg.39], [Pg.62], [Pg.65]

[63] M. Haouari, S. B. Layeb, and H. D. Sherali. Algorithmic expedients for the Prize Collecting Steiner Tree Problem. *Discrete Optimization*, 7:32–47, 2010. [Pg.28], [Pg.30]

[64] M. Hauptmann and M. Karpinski. A Compendium on Steiner Tree Problems. *http://theory.cs.uni-bonn.de/info5/steinerkompendium/netcompendium.html*, 2015. [Pg.3]

[65] J. W. Hegeman, G. Pandurangan, S. V. Pemmaraju, V. B. Sardeshmukh, and M. Scquizzato. Toward Optimal Bounds in the Congested Clique: Graph Connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 91–100, 2015. [Pg.78]

[66] J. W. Hegeman and S. V. Pemmaraju. Lessons from the Congested Clique Applied to MapReduce. *Theoretical Computer Science*, 608(P3):268–281, 2015. [Pg.68]

[67] J. W. Hegeman, S. V. Pemmaraju, and V. B. Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *Proceedings of the 28th International Conference on Distributed Computing*, DISC'14, pages 514–530, 2014. [Pg.8], [Pg.68], [Pg.71]

[68] S. Holzer and N. Pinsker. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, volume 46, pages 1–16, 2016. [Pg.8], [Pg.68]

[69] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 355–364, 2012. [Pg.1]

[70] S. Hougardy and H. J. Prömel. 1.598 Approximation Algorithm for the Steiner Problem in Graph. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 448–453, 1999. [Pg.19], [Pg.20]

[71] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 448–457, 1998. [Pg.20]

[72] D. S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 760–769, 2000. [Pg.4], [Pg.11], [Pg.28], [Pg.31], [Pg.33], [Pg.34], [Pg.96], [Pg.97], [Pg.120], [Pg.126]

[73] T. Jurdziński and K. Nowicki. MST in O(1) Rounds of Congested Clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2620–2632, 2018. [Pg.8], [Pg.68], [Pg.71], [Pg.78], [Pg.142]

[74] R. M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972. [Pg.2]

[75] M. Karpinski and A. Zelikovsky. New Approximation Algorithm for Steiner Tree Problem. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997. [Pg.19], [Pg.20]

[76] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 263–272, 2008. [Pg.2], [Pg.22], [Pg.23], [Pg.27]

[77] M. Khan and G. Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. *Distributed Computing*, 20(6):391–402, 2008. [Pg.41]

[78] G. W. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In *Genetic and Evolutionary Computation, GECCO '04*, pages 1304–1315, 2004. [Pg.28], [Pg.32]

[79] G. W. Klau, I. Ljubić, P. Mutzel, U. Pferschy, and R. Weiskircher. The Fractional Prize-Collecting Steiner Tree Problem on Trees. In *European Symposium on Algorithms - ESA'03*, pages 691–702, 2003. [Pg.28]

[80] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 391–410, 2015. [Pg.68]

[81] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Two Distributed Algorithms for Multicasting Multimedia Information. *IEEE/ACM Transaction on Networking*, 1:286–292, 1993. [Pg.22], [Pg.23], [Pg.24], [Pg.25]

[82] J. Könemann, D. Pritchard, and K. Tan. A Partition-based Relaxation for Steiner Trees. *Math. Program.*, 127(2):345–370, Apr. 2011. [Pg.20], [Pg.21]

[83] L. Kor, A. Korman, and D. Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory of Computing Systems*, 53(2):318–340, 2013. [Pg.2]

[84] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15(2):141–145, 1981. [Pg.10], [Pg.19], [Pg.20], [Pg.51], [Pg.61], [Pg.64], [Pg.68], [Pg.75], [Pg.82]

[85] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 7–15, 2006. [Pg.1]

[86] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. On the complexity of universal leader election. *Journal of the ACM*, 62(1):7:1–7:27, 2015. [Pg.7], [Pg.9], [Pg.38], [Pg.92], [Pg.141]

[87] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40 – 66, 1998. [Pg.viii], [Pg.2], [Pg.7], [Pg.37], [Pg.38], [Pg.53], [Pg.62]

[88] T. Lengaue. Combinatorial Algorithms for Integrated Circuit Layout. *Wiley, Chichester, England*, 1990. [Pg.4]

[89] C. Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 42–50, 2013. [Pg.8], [Pg.68], [Pg.73]

[90] C. Lenzen and B. Patt-Shamir. Fast partial distance estimation and applications. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 153–162, 2015. [Pg.viii], [Pg.2], [Pg.7], [Pg.11], [Pg.22], [Pg.23], [Pg.27], [Pg.37], [Pg.38], [Pg.65], [Pg.140]

[91] C. Lenzen and D. Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 375–382, 2013. [Pg.42]

[92] Y. Li, D. Du, N. Xiu, and D. Xu. Improved approximation algorithms for the facility location problems with linear/submodular penalties. *Algorithmica*, 73(2):460–482, Oct. 2015. [Pg.30], [Pg.91]

[93] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. [Pg.6]

[94] I. Ljubić. Exact and memetic algorithms for two network design problems (unpublished doctoral dissertation). *Faculty of Computer Science, Technische Universität Wien*, 2004. [Pg.32], [Pg.34]

[95] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical Programming*, Series B(105):427–449, 2006. [Pg.28], [Pg.29]

[96] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-Weight Spanning Tree Construction in O(Log Log N) Communication Rounds. *SIAM Journal on Computing*, 35(1):120–131, July 2005. [Pg.8], [Pg.13], [Pg.67], [Pg.68], [Pg.70], [Pg.71], [Pg.75], [Pg.78], [Pg.82], [Pg.142]

[97] C. L. Lu, C. Y. Tang, and R. C. Lee. The full Steiner tree problem in phylogeny. In *International Computing and Combinatorics Conference (COCOON)*, pages 107–116, 2002. [Pg.6]

[98] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. Averaging in Distributed Primal-dual Optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1973–1982, 2015. [Pg.93]

[99] T. L. Magnanti and R. T. Wong. Network design and transportation planning: models and algorithms. *Transportation Science*, 18:1–55, 1984. [Pg.4]

[100] D. Mölle, S. Richter, and P. Rossmanith. A Faster Algorithm for the Steiner Tree Problem. In *Symposium on Theoretical Aspects of Computer Science*, pages 561–570, 2006. [Pg.17], [Pg.18]

[101] P. Moscato. Memetic algorithms: A short introduction. *New Ideas in Optimization*, pages 219–234, 1999. [Pg.32]

[102] T. Moscibroda and R. Wattenhofer. Facility location: Distributed approximation. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 108–117, 2005. [Pg.10], [Pg.14], [Pg.92], [Pg.93]

[103] I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971. [Pg.72], [Pg.74]

[104] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 565–573, 2014. [Pg.8], [Pg.68]

[105] R. Novak and J. Rugelj. Distribution of Constrained Steiner Tree Computation in Shortest-Delay Networks. *In: MELECON'96, 8th Mediterranean Electrotechnical Conference*, 2:0–3, 1996. [Pg.23]

[106] R. Novak, J. Rugelj, and G. Kandus. A note on distributed multicast routing in point-to-point networks. *Computers and Operation Research*, 28(12):1149–1164, 2001. [Pg.22], [Pg.25]

[107] K. Nowicki. A Deterministic Algorithm for the MST Problem in Constant Rounds of Congested Clique. *https://arxiv.org/abs/1912.04239*, 2020. [Pg.8], [Pg.68]

[108] S. Pai and S. V. Pemmaraju. Connectivity lower bounds in broadcast congested clique. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 256–258, 2019. [Pg.68]

[109] S. Pandit and S. Pemmaraju. Return of the primal-dual: Distributed metric facility location. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, pages 180–189, 2009. [Pg.10], [Pg.14], [Pg.92], [Pg.93]

[110] G. Pandurangan, P. Robinson, and M. Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC '17, pages 743–756, 2017. [Pg.viii], [Pg.7], [Pg.8], [Pg.12], [Pg.39], [Pg.62], [Pg.63], [Pg.65]

[111] B. Patt-Shamir and M. Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *Stabilization, Safety, and Security of Distributed Systems*, pages 1–17, 2017. [Pg.68]

[112] B. Patt-Shamir and M. Teplitsky. The round complexity of distributed sorting: Extended abstract. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Sympo-*

*sium on Principles of Distributed Computing*, PODC '11, pages 249–256, 2011. [Pg.8], [Pg.68]

[113] D. Peleg. Distributed matroid basis completion via elimination upcast and distributed correction of minimum-weight spanning trees. In *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming, ICALP'98*, pages 164–175, 1998. [Pg.53]

[114] D. Peleg. Distributed Computing: A Locality-Sensitive Approach. *SIAM : Discrete Mathematics and Applications*, 2000. [Pg.6], [Pg.9], [Pg.11], [Pg.40], [Pg.43], [Pg.94], [Pg.120]

[115] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing*, 30(5):1427–1442, 2000. [Pg.6], [Pg.7], [Pg.38]

[116] S. V. Pemmaraju and V. B. Sardeshmukh. Super-Fast MST Algorithms in the Congested Clique Using o(m) Messages. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, pages 47:1–47:15, 2016. [Pg.78], [Pg.142]

[117] A. Prodon, S. DeNegre, and T. M. Liebling. Locating leak detecting sensors in a water distribution network by solving prize-collecting Steiner arborescence problems. *Mathematical Programming*, 124(1):119–141, 2010. [Pg.6]

[118] H. J. Promel and A. Steger. A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio $\frac{5}{2}$. *Journal of Algorithms*, 36(1):89–101, 1997. [Pg.19], [Pg.20]

[119] S. Rajagopalan and V. V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 742–751, 1999. [Pg.20], [Pg.21]

[120] N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986. [Pg.29]

[121] G. Robins and A. Zelikovsky. Improved Steiner Tree Approximation in Graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 770–779, 2000. [Pg.19], [Pg.20], [Pg.21], [Pg.141]

[122] N. G. Rossetti. A First Attempt on the Distributed Prize-Collecting Steiner Tree Problem. In *M.Sc. thesis, University of Iceland, Reykjavik*, 2015. [Pg.ix], [Pg.8], [Pg.11], [Pg.33], [Pg.35]

[123] J. Schneider and R. Wattenhofer. An optimal maximal independent set algorithm for bounded-independence graphs. *Distributed Computing*, 22(5):349–361, 2010. [Pg.1]

[124] G. Singh and K. Vellanki. A distributed protocol for constructing multicast trees. In *2nd International Conference On Principles Of Distributed Systems*, OPODIS '98, pages 61–76, 1998. [Pg.22], [Pg.23], [Pg.25]

[125] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematics*, 24:573–577, 1980. [Pg.19], [Pg.20]

[126] V. V. Vazirani. Approximation Algorithms. *Sringer*, 2004. [Pg.18]

[127] S. Voß. Steiners problem in graphs: heuristic methods. *Discrete Applied Mathematics*, 40:45–72, 1992. [Pg.24]

[128] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2trees, and minimum IFI networks. *Networks*, 13:159–167, 1983. [Pg.30]

[129] P. Winter and J. S. MacGregor. Path-distance heuristics for the Steiner problem in undirected networks. *Algorithmica*, 7(1):309–327, 1992. [Pg.22]

[130] Y. F. Wu, P. Widmayer, and C. K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta Informatica*, 23(2):223–229, 1986. [Pg.10], [Pg.51], [Pg.52], [Pg.68]

[131] D. Yuan, S. Xu, and H. Zhao. Distributed Primal-Dual Subgradient Method for Multiagent Optimization via Consensus Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(6):1715–1724, 2011. [Pg.93]

[132] A. Zelikovsky. An 11/6-Approximation Algorithm for the Network Steiner Problem. *Algorithmica*, 9:463–470, 1993. [Pg.19], [Pg.20]

[133] A. Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. *Technical report, CS-96-06, University of Virginia.*, 1995. [Pg.19], [Pg.20]

**Department of Computer Science and Engineering**

**Indian Institute of Technology Guwahati**

**Guwahati 781039, India**