On the parallel time complexity of undirected connectivity and minimum spanning trees

Ka Wong Chong[†] Yijie Han[‡] Tak Wah Lam[§]

Abstract. We present a new approach to finding minimum spanning trees of weighted undirected graphs on the parallel random access machine (PRAM) without concurrent-write power. This approach gives an algorithm that runs in $O(\log n)$ time using $n+m$ processors on the EREW PRAM, settling a long-standing open problem in the literature.

1 Introduction

Given a weighted undirected graph G with n vertices and m edges, the minimum spanning tree (MST) problem is to find a spanning tree (or forest) of G such that the sum of the edge weights of the spanning tree is minimized. This problem has a long history. Sequential MST algorithms running in $O(m \log n)$ time are well known (see [17] for a survey). A more involved algorithm, developed by Gabow *et al.* [8], runs in $O(m \log \beta(m,n))$ time, where $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$. Very recently, Chazelle [2] further improved the time complexity to $O(m\alpha(m,n) \log \alpha(m,n))$, where $\alpha(m,n)$ is the inverse Ackerman function. On the other hand, Karger *et al.* [13] designed a randomized algorithm running in expected linear time.

In the parallel context, the MST problem is closely related to the connected component problem (i.e., finding the connected components of an undirected graph). The connected component problem actually admits a faster algorithm in the sequential context, yet techniques for solving the two problems in parallel are very similar. If concurrent write is allowed, it is relatively simple to solve both problems in $O(\log n)$ time using $n+m$ processors on the CRCW PRAM [1, 6]. Using randomization, Cole *et al.* [5] were able to improve the processor bound to $(n+m)/\log n$, while maintaining $O(\log n)$ expected time.

For the exclusive write models (i.e., CREW and EREW PRAMs), $O(\log^2 n)$ time algorithms for the connected component and MST problems were developed two decades ago (see [15] for a survey). For a while, it was believed that exclusive write models cannot overcome the $O(\log^2 n)$ time bound. The first breakthrough is due to Johnson and Metaxas; they devised $O(\log^{1.5} n)$ time algorithms for the connected component problem [10] and the MST problem [11]. Their results were later improved by Chong and Lam to $O(\log n \log\log n)$ time [4, 3]. Randomization can help again. Karger *et al.* [14, 12] devised a randomized algorithm running in $O(\log n)$ expected time; more recently, Poon and Ramachandran [16] gave the first randomized algorithm using linear expected work and polylog expected time (precisely, $O(\log n \cdot \log\log n \cdot 2^{\log^* n})$).

Until now, it has been open whether the problems can be solved deterministically in $O(\log n)$ time on the eclusive write models. Notice that $O(\log n)$ is optimal in view of the lower bound results of Cook *et al.* [7].

This paper presents a new parallel algorithm for the MST problem. It runs in $O(\log n)$ time using $n+m$ processors on the EREW PRAM. The algorithm is deterministically in nature and does not require special operations on edge weights (other than comparison). To simplify the discussion, we first focus on the CREW PRAM, showing how to solve the MST problem in $O(\log n)$ time using $(n+m)\log n$ processors. Afterwards techniques for adapting to the EREW PRAM and reducing the processor bound are sketched.

The rest of the paper is organized as follows. Section 2 provides some basic ideas about finding minimum spanning trees in parallel. Section 3 gives an overview of the schedule of the concurrent processes used in the algorithm. Section 4 lays down the phase-by-phase requirement of each process. Section 5 shows the details of the algorithm. The proof of correctness of the algorithm appears in Section 6.

2 Preliminaries

In this section we present a naive approach to finding MST, based on which we can contrast our algorithm with existing ones.

Let G be a weighted undirected graph. For every edge $e \in G$, denote $w(e)$ as its weight. Without loss of generality, we assume that the edge weights are all distinct. Thus, G has a unique minimum spanning tree, which is denoted by T_G^* throughout this paper. We also

[†]Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany. Email: chong@mpi-sb.mpg.de

[‡]Electronic Data Systems, Inc., 750 Tower Drive, Mail Stop 7121, CPS, Troy, MI 48098, USA. Email: yhan01@cps.plnin.gmeds.com

[§]Department of Computer Science, The University of Hong Kong, Hong Kong. Email: twlam@cs.hku.hk

assume that G is connected (otherwise, our algorithm can find the minimum spanning forest of G).

Let B be a subset of edges in G which contains no cycle. B induces a set of trees $F = \{T_1, T_2, \cdots, T_i\}$ in a natural sense: Two vertices in G are in the same tree if they are connected by edges of B. If B contains no edge incident on a vertex v, then v itself forms a tree. Consider a vertex u in a tree T_j and any edge (u, v) in G. If v also belongs to T_j, it is called an *internal* edge; otherwise, it is an *external* edge.

Definition: B is said to be a λ-forest if each $T_j \in F$ has at least λ vertices.

For example, an empty set B of edges is a 1-forest of G and a spanning tree such as T_G^* is an n-forest. Consider a set B of edges chosen from T_G^*. Assume that B is a λ-forest. We can augment B to give a 2λ-forest using a greedy approach: Let F' be an arbitrary subset of F such that F' includes all trees $T_j \in F$ with less than 2λ vertices (F' may contain some trees with 2λ or more vertices). For every tree in F', we pick the minimum external edge. Denote B' as this set of edges.

LEMMA 2.1. $B \cup B'$ is a 2λ-forest and consists of edges in T_G^* only.

Proof. Every tree in $F - F'$ already contains at least 2λ vertices. Consider a tree T_j in F'. Let (u, v) be the minimum external edge of T_j. Then v belongs to another tree $T_{j'} \in F$. With respect to $B \cup B'$, all vertices in T_j and $T_{j'}$ are connected together. Among the trees induced by $B \cup B'$, there is one including T_j and T_j', containing at least 2λ vertices. Therefore, $B \cup B'$ is a 2λ-forest of G. □

Based on Lemma 2.1, we can find T_G^* in $\lfloor \log n \rfloor$ stages as follows:

Notation: Let $B[p, q]$ denote $\cup_{k=p}^{q} B_k$ if $p \leq q$, otherwise an empty set.

procedure Iterative-MST(G)

1. **for** $i = 1$ to $\lfloor \log n \rfloor$ **do** /* Stage i */
   (a) Let F be the set of trees induced by $B[1, i-1]$ on G. Let F' be an arbitrary subset of F such that F' includes all trees $T \in F$ with less than 2^i vertices.
   (b) $B_i \leftarrow \{e \mid e$ is the minimum external edge of $T \in F'\}$
2. **return** $B[1, \lfloor \log n \rfloor]$.

Notice that $B[1, i]$, as computed after i stages, is a subset of T_G^* and a 2^i-forest. Different strategies for choosing the set F' in Step 1(a) may lead to different B_i's. Nevertheless, at the end $B[1, \lfloor \log n \rfloor]$ contains exactly all edges of T_G^*. This algorithm is similar to Borůvka's MST algorithm (see e.g., [17]), which is developed in the sequential context. Using standard parallel algorithmic techniques, each stage can be implemented in $O(\log n)$ time on the EREW PRAM using a linear number of processors (see e.g., [9]). Therefore, T_G^* can be found in $O(\log^2 n)$ time. In fact, many parallel algorithms for finding MST (see e.g., [15, 9, 10, 11, 4, 3]) are based on a similar approach. These algorithms are uniform in the sense that the computation for B_i starts only after B_{i-1} is available (see Figure 1(a)).

An innovative idea exploited by our algorithm is the "multi-thread" computation. It uses concurrent processes to find the B_i's. In particular, the computation for B_i starts long before B_{i-1} is found. After B_{i-1} is found, B_i can be computed in $O(1)$ time (see Figure 1(b)). As a result, T_G^* can be found in $O(\log n)$ time.

Our algorithm takes advantage of some naive properties of the sets $B_1, B_2, \cdots, B_{\lfloor \log n \rfloor}$. These properties actually hold with respect to most of the deterministic algorithms for finding MST, though not having been mentioned explicitly. Recall that for all $1 \leq i \leq \lfloor \log n \rfloor$, each edge in B_i is the minimum external edge of one of the trees induced by $B[1, i-1]$.

PROPOSITION 2.1. Let T be any one of the trees induced by $B[1, k]$, for any $0 \leq k \leq \lfloor \log n \rfloor$. Let (a, p) and (b, q) be two external edges of T, where $a, b \in T$ and are distinct. Then any edge on the path connecting a and b in T has weight smaller than $\max\{w(a, p), w(b, q)\}$.

PROPOSITION 2.2. Let T be any one of the trees induced by $B[1, k]$, for any $0 \leq k \leq \lfloor \log n \rfloor$. Let e_T be the minimum external edge of T. For any subtree (i.e., connected subgraph) S of T, the minimum external edge of S is either e_T or an edge of T.

3 Overview of the algorithm

Our algorithm consists of $\lfloor \log n \rfloor$ processes running concurrently. For $1 \leq i \leq \lfloor \log n \rfloor$, Process i is responsible for finding a set B_i which can be produced at Stage i of the procedure Iterative-MST. To be precise, let F be the set of trees induced by $B[1, i-1]$ and let F' be an arbitrary subset of F including all trees with less than 2^i vertices; B_i contains the minimum external edges of the trees in F'. Process i makes use of the output of Processes 1, $\cdots$, $(i-1)$ (i.e., $B_1, \cdots, B_{i-1}$), but never looks at their computation. We expect that after B_{i-1} is found, Process i can compute B_i in time $O(1)$.

Before showing the detailed schedule of Process i,

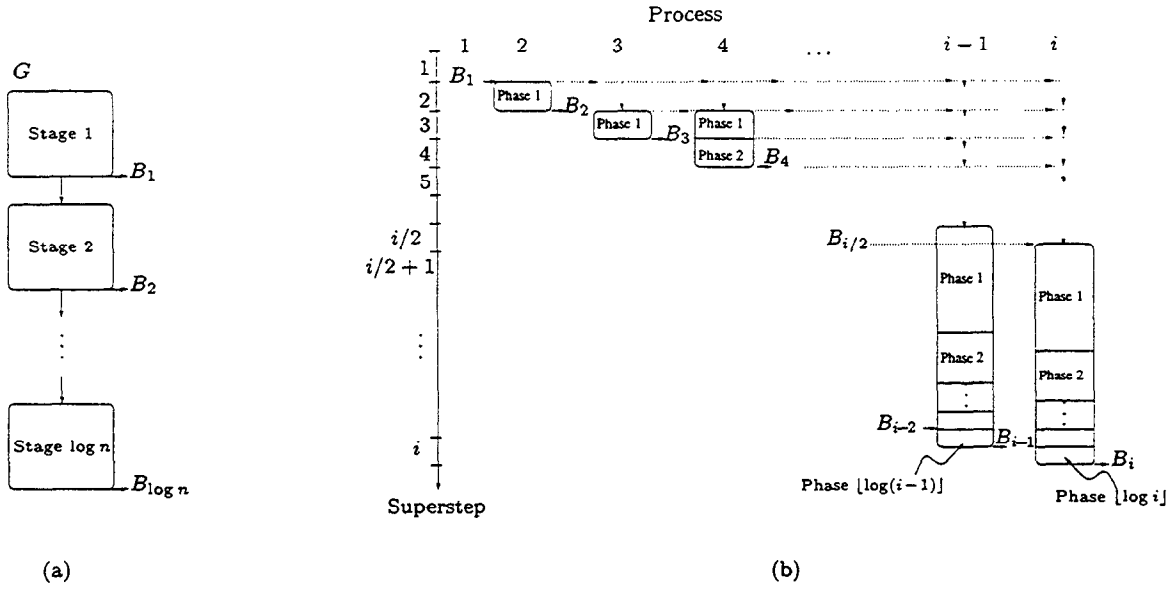

Figure 1: (a) The iterative approach. (b) The multi-thread approach.

we give two examples illustrating some of the main ideas in our algorithm. In Examples 1 and 2, B_i is computed in time ci and $\frac{1}{2}ci$ respectively after B_{i-1} is found, where c is some fixed constant.

Process i starts off with a set Q_0 of adjacency lists, each list contains the $2^i - 1$ smallest edges incident on a vertex in G. Notice that the edges kept in Q_0 are already sufficient for computing B_i (consider each tree with less than 2^i vertices induced by $B[1, i-1]$, its minimum external edge must be one of the $2^i - 1$ smallest edges incident on its vertices).

Example 1: This is a straightforward implementation of Lemma 2.1. Process i starts when $B_1, \cdots, B_{i-1}$ are all available. With respect to $B[1, i-1]$, the vertices of G are partitioned into a set F of trees. We merge the adjacency lists of the vertices in each tree, forming a combined adjacency list. Notice that if a tree T has less than 2^i vertices, its combined adjacency list contains at most $(2^i - 1)^2$ edges. For each combined list with at most $(2^i - 1)^2$ edges, we remove the internal edges and then find the minimum external edge. The collection of such minimum external edges is reported as B_i. The computation takes time at most ci, where c is some suitable constant. To see the correctness, we observe that every tree $T \in F$ with more than $(2^i - 1)^2$ edges on its combined adjacency list must contain at least 2^i vertices and, by definition of B_i, it is not necessary to report the minimum external edge of T.

Example 2: This example is slightly more complex, illustrating how Process i works in an "incremental" manner. Process i starts off as soon as $B_{i/2}$ has been computed. At this point, only $B_1, \cdots, B_{i/2}$ are available and Process i is indeed not ready to compute B_i. Nevertheless, it can perform some preprocessing (called Phase I below) so that when $B_{i/2+1}, \cdots, B_{i-1}$ are available, the computation of B_i can be speeded up to run in time $\frac{1}{2}ci$ only (Phase II).

Phase I: Let $\bar{F}$ be the set of trees induced by $B[1, i/2]$. Following Example 1, we merge the adjacency lists in Q_0 for every tree in $\bar{F}$, forming another set $\bar{Q}$ of adjacency lists. For each tree T with less than 2^i vertices, its combined adjacency list contains no more than $(2^i - 1)^2$ edges. We remove the internal edges and extra multiple external edges to other trees. Afterwards we only retain the $2^{i/2} - 1$ smallest edges (the rest are said to be *truncated*). These edges are sufficient for finding B_i (the argument is an extension of the argument in Example 1). The computation takes time ci only.

Phase II: When $B_{i/2+1}, \cdots, B_{i-1}$ are available, we compute B_i based on $\bar{Q}$ as follows: Edges in $B[i/2 + 1, i-1]$ further connects the trees in $\bar{F}$, forming a set F of bigger trees. We merge the lists in $\bar{Q}$ for every tree in F. Notice that if a tree $T \in F$ contains less than 2^i vertices, T is composed of at most $2^{i/2} - 1$ trees in $\bar{F}$ and its combined adjacency list contains no more than $(2^{i/2} - 1)^2$ edges. As a result, we can remove the internal edges and find the minimum external edge in time at most $\frac{1}{2}ci$. B_i is the set of minimum external edges just found. In conclusion, after B_{i-1} is computed,

B_i is found in time $\frac{1}{2}ci$.

Remark: The set B_i found by Examples 1 and 2 may be different. Yet in either cases, $B_i \cup B[1, i-1] \subseteq T_G^*$ and is a 2^i-forest.

The schedule: Our MST algorithm is based on a generalization of the above ideas. Process i divides its computation into $\lfloor \log i \rfloor$ phases, finding B_i in time $O(1)$ after Process $i-1$ has computed B_{i-1}. (See Figure 1(b)).

Globally speaking, the algorithm runs in $\lfloor \log n \rfloor$ supersteps, each superstep lasts for time $4c$, where c is the same constant discussed before. The processes are scheduled according to a "global clock". In particular, Process i delivers B_i at the end of the ith superstep. To ease our discussion of the schedule of Process i, we first consider i a power of two. Phase 1 of Process i starts when $B_1, \cdots, B_{i/2}$ are available. The computation takes no more than $i/4$ supersteps (i.e., from the $(i/2+1)$th to the $(i/2+i/4)$th supersteps). Phase 2 starts when $B_{i/2+1}, \cdots, B_{i/2+i/4}$ are available, using $i/8$ supersteps (i.e., from the $(i/2+i/4+1)$th to $(i/2+i/4+i/8)$th supersteps). Every next phase further reduces the number of supersteps by half. The last phase (Phase $\log i$) starts and finishes within the ith superstep.

In general, Process i runs in $\lfloor \log i \rfloor$ phases. Let $a_0 = 0$ and let $a_j = a_{j-1} + \lceil (i - a_{j-1})/2 \rceil$ for any $j \geq 1$. For example, $a_1 = \lceil i/2 \rceil$ and $a_{\lfloor \log i \rfloor} = i - 1$. Phase j, where $1 \leq j \leq \lfloor \log i \rfloor$, starts from the (a_j+1)th superstep, using $\lceil (i - a_j)/2 \rceil$ supersteps. It has to handle the edges in $B_{a_{j-1}+1}, \cdots, B_{a_j}$, which are not available to previous phases.

4 Requirement of a phase

In this section we discuss what Process i is expected to achieve at the end of each phase.

Initially, Process i constructs a set $\mathcal{Q}_0$ of adjacency lists (Phase 0). For each vertex v in G, $\mathcal{Q}_0$ contains a circular linked list $\mathcal{L}$ including the $2^i - 1$ smallest edges incident on v (if v has degree less than $2^i - 1$, the list contains all incident edges). In addition, $\mathcal{L}$ is assigned a *threshold*, denoted by $h(\mathcal{L})$. If $\mathcal{L}$ contains all edges of v, $h(\mathcal{L}) = +\infty$; otherwise, $h(\mathcal{L}) = w(e_o)$, where e_o is the smallest external edge truncated from $\mathcal{L}$. Intuitively, the threshold records the smallest edge that has been truncated so far. In each subsequent phase, the adjacency lists are further merged in view of the newly arrived B's, and truncated if necessary.

Consider Phase $j \geq 1$. Recall that $B_1, \cdots, B_{a_j}$ are all available when Phase j starts. Denote F_j as the set of trees induced by $B[1, a_j]$. The adjacency lists produced at the end of Phase j, denoted by $\mathcal{Q}_j$, capture some external edges of the trees in F_j. More precisely, $\mathcal{Q}_j$ satisfies the following requirements.

Let $\mathcal{L}$ be a list in $\mathcal{Q}_j$.

R1 (representation): $\mathcal{L}$ uniquely corresponds to a tree $T \in F_j$. In this case, T is also said to be represented by $\mathcal{L}$ in $\mathcal{Q}_j$. Some trees in F_j may not be represented by any lists in $\mathcal{Q}_j$; however, those trees with less than 2^i vertices are all guaranteed.

R2 (external edges): $\mathcal{L}$ contains only external edges of T and for every edge $e \in \mathcal{L}$, $w(e) < h(\mathcal{L})$.

R3 (length): $\mathcal{L}$ is non-empty and contains at most $2^{i-a_j} - 1$ edges.

Let us look at the implication to the last phase (Phase $\lfloor \log i \rfloor$). Notice that $a_{\lfloor \log i \rfloor} = i - 1$. R1, R2, and R3 together imply that every list in $\mathcal{Q}_{\lfloor \log i \rfloor}$ contains exactly one external edge of a tree in $F_{\lfloor \log i \rfloor}$. Moreover, if a tree in $F_{\lfloor \log i \rfloor}$ has less than 2^i vertices, it must be represented by a list in $\mathcal{Q}_{\lfloor \log i \rfloor}$. Therefore, we could report the edges in the lists of $\mathcal{Q}_{\lfloor \log i \rfloor}$ as a solution of B_i. However, these requirements are not sufficient to enforce that the external edge in each list is indeed the minimum one.

Below, we describe additional requirements of the quality of the edges in $\mathcal{Q}_j$. Consider an external edge of a tree $T \in F_j$. Suppose e connects T to another tree $T' \in F_j$. We say that e is *primary* if, among all edges connecting T and T', e has the smallest weight. Otherwise, e is is said to be *duplicate*. If T_G^* does contain an edge between T and T', it must be a primary external edge of T (as well as of T'). Therefore, we focus on retaining primary external edges in $\mathcal{Q}_j$. Yet there are possibly too many primary external edges, some of them must be truncated in order to satisfy the length requirement. We need another notion called *base* to characterize those primary external edges that are really "critical", which would be within the length requirement.

Definition: Let γ be any real number. For any two trees $T, T' \in F_j$, T is said to be γ-accessible to T' if (i) $T = T'$ or (ii) there is another tree $T'' \in F_j$ such that T and T'' are connected by an edge with weight smaller than γ and T'' is γ-accessible to T'.[1]

Notation: Let $e = (u, v)$ be an edge in G. When we refer e as an external edge of a subtree $T \in F_j$, we use the notation either $\langle u, v \rangle$ or $\langle v, u \rangle$ to signify whether u or v is in T.

Definition: Let $e = \langle u, v \rangle$ be an external

[1] By Proposition 2.1, T is γ-accessible to T' if and only if there is a path in G connecting T and T' using edges with weight $< \gamma$.

edge of $T \in F_j$. Define $F_j\text{-}base\langle u,v\rangle = \{T' \mid T' \in F_j$ and T is $w(u,v)$-accessible to $T'\}$. The size of $F_j\text{-}base\langle u,v\rangle$, denoted by $|F_j\text{-}base\langle u,v\rangle|$, is the total number of vertices in the trees involved (instead of the number of trees).

By definition, $F_j\text{-}base\langle u,v\rangle$ may not be equal to $F_j\text{-}base\langle v,u\rangle$. It is easy to show that **(i)** for any two external edges $\langle u,v\rangle$ and $\langle u',v'\rangle$ of T, if $\langle u',v'\rangle$ has a bigger weight, then $F_j\text{-}base\langle u,v\rangle \subseteq F_j\text{-}base\langle u',v'\rangle$; **(ii)** for any $k \leq j$, $F_k\text{-}base\langle u,v\rangle \subseteq F_j\text{-}base\langle u,v\rangle$.

Intuitively, it is not necessary to report in B_i any external edge whose base contains 2^i or more vertices; such edges could be reported in $B_{i'}$ for some $i' > i$ (i.e., by Process i'). Within Phase j, we keep track of external edges whose bases contain less than 2^i vertices; such edges would include those that must be reported in B_i. In particular, for any tree $R \in F_{\lfloor \log i \rfloor}$ containing less than 2^i vertices, the minimum external edge $\langle x,y\rangle$ of R, which must be included in B_i, does satisfy the condition that $|F_{\lfloor \log i \rfloor}\text{-}base\langle x,y\rangle| < 2^i$ (since $F_{\lfloor \log i \rfloor}\text{-}base\langle x,y\rangle$ is exactly R).

For any tree $T \in F_j$, an external edge $\langle u,v\rangle$ is said to be *critical* if (u,v) is primary and $|F_j\text{-}base\langle u,v\rangle| < 2^i$. Denote $K(T)$ as the set of all critical edges of T. We are ready to state an additional requirement of $\mathcal{Q}_j$.

R4 (base) Every list $\mathcal{L} \in \mathcal{Q}_j$ contains all edges of $K(T)$ where $T \in F_j$ is represented by $\mathcal{L}$.

R4 does not contradict R3 in any case because $K(T)$ cannot be too large.

LEMMA 4.1. $K(T)$ contains at most $2^{i-a_j} - 1$ edges.

Proof. Let $\langle u,v\rangle$ be the edge in $K(T)$ with the biggest weight. By definition of $K(T)$, $|F_j\text{-}base\langle u,v\rangle| < 2^i$. The base of $\langle u,v\rangle$ includes T and at least $K(T)-1$ trees adjacent to T. Recall that $B[1,a_j]$ is a 2^{a_j}-forest and every tree in F_j contains at least 2^{a_j} vertices. We have $|F_j\text{-}base\langle u,v\rangle| \geq 2^{a_j} + (|K(T)|-1)\cdot 2^{a_j} \geq |K(T)|\cdot 2^{a_j}$. Thus, $|K(T)| < 2^{i-a_j}$. □

So far our concern is that at the end of Phase j, every tree $T \in F_j$ with less than 2^i vertices must be represented by a list in $\mathcal{Q}_j$, which contains $K(T)$ and hence the minimum external edge of T. However, for those trees T' with at least 2^i vertices, $K(T')$ is empty by definition, and R4 imposes nothing in this case. Fortunately, it is unnecessary to report (at the end of Phase $\lfloor \log i \rfloor$) the minimum external edge of T' as part of B_i. The following requirement on the threshold kept with any list $\mathcal{L} \in \mathcal{Q}_j$ enforces that if the minimum external edge of T' has been removed in Phase j, we will not report any edge for T' at the end of Phase $\lfloor \log i \rfloor$.

R5 (threshold) Let $\mathcal{L}$ be a list in $\mathcal{Q}_j$ representing T. For any primary external edge e of T not included in $\mathcal{L}$, $h(\mathcal{L}) \leq w(e)$.

R2 and R5 together implies that if the minimum external edge of a tree T' is removed in Phase j, then T' is not represented by any (non-empty) list in $\mathcal{Q}_j$.

It is easy to check that $\mathcal{Q}_0$ satisfies all the five requirements. In summary, the requirements R1 to R5 guarantees that at the end of the last phase, for any tree $T \in F_{\lfloor \log i \rfloor}$, if T has less than 2^i vertices, its minimum external edge e_T is the only edge kept in the adjacency list representing T; otherwise, e_T may not be kept and the second part of R2 guarantees that no such list would exist in $\mathcal{Q}_j$ representing T. Consequently, the remaining edges in $\mathcal{Q}_{\lfloor \log i \rfloor}$ constitutes B_i correctly. Next section gives a parallel algorithm for implementing Process i.

5 The algorithm

As mentioned before, Phase j inherits the adjacency lists from Phase $j-1$ (i.e., $\mathcal{Q}_{j-1}$) and receives the edges $B[a_{j-1}+1, a_j]$. Let $I_j = B[a_{j-1}+1, a_j]$. Denote F_{j-1} and F_j as the set of trees induced by $B[1,a_{j-1}]$ and $B[1,a_j]$ respectively. Notice that a list in $\mathcal{Q}_{j-1}$ represents one of the trees in F_{j-1}. One of the major jobs of Phase j is to merge the adjacency lists of the trees in F_{j-1} that are connected together by the edges of I_j; afterward, some edges or lists are removed in order to satisfy the requirements. Consider an edge $e = (u,v)$ in I_j. Denote W_1 and W_2 as the trees in F_{j-1} containing u and v respectively. Ideally, if e appears in the adjacency lists of W_1 and W_2, we can merge the adjacency lists of W_1 and W_2 easily; the two copies of e simply exchange their successors and the two lists (which are circular) become one [18]. See Figure 2 for an example. However, W_1 or W_2 might already be too large and does not get a representation in $\mathcal{Q}_{j-1}$. Even they are represented, the length requirement of the adjacency lists may not

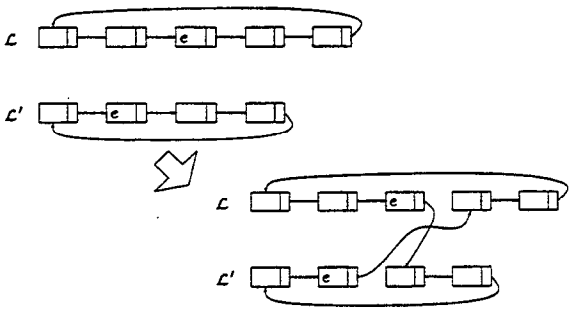

Figure 2: Merging a pair of adjacency lists with respect to a common edge e.

allow e to be included. As a result, e may appear in two

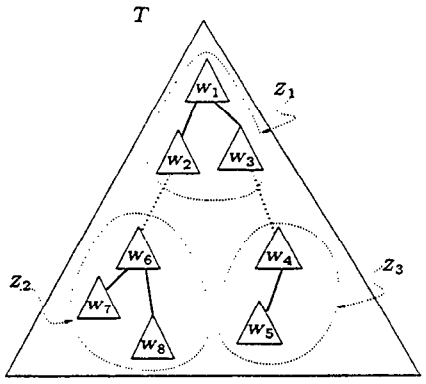

Figure 3: Each W_x represents a tree in F_{j-1}. The dotted and solid lines represent half and full edges in I_j respectively. T is a tree formed by connecting the trees in F_{j-1} with the edges in I_j. Each Z_y (called a cluster) is a subtree of T, formed by connecting some W_x's with full edges only. The adjacency lists of the W_x's within each Z_y can be merged into a single list easily.

separate lists in $\mathcal{Q}_{j-1}$, or in just one, or in even none; we call e a *full*, *half*, and *lost* edge, respectively.

We merge the lists in $\mathcal{Q}_{j-1}$ with respect to the full edges in I_j only. The trees in F_{j-1} that are connected by edges in I_j into a single tree in F_j may have their adjacency lists merged into several lists instead of a single one (see Figure 3). The key observation is that for every tree $T \in F_j$ with less than 2^i vertices, $K(T)$ is included in only one of such lists. More interestingly, all such lists, except the one containing $K(T)$, can be shown to be "obsolete" in the following sense and thus get removed easily.

- If a list contains more than $(2^{i-a_{j-1}}-1)^2$ edges, then it represents a tree comprising at least $2^{i-a_{j-1}}$ trees in F_{j-1}, i.e., at least 2^i vertices. It is not necessary to consider such a big tree in Process i.
- If a list contains a half edge $\langle u,v\rangle$ that is in I_j, we will argue that the base of every external edge in the list is at least 2^i. Thus, the list can be discarded.
- If all edges in a list have weight greater than the updated threshold, it means that the minimum external edge has been lost and the list is useless.

In brief, after merging the lists inherited from $\mathcal{Q}_{j-1}$, we attempt to update the threshold of each combined list, remove those internal edges and duplicate external edges, and truncate those heavy edges to satisfy the length requirement. An interesting idea in our algorithm is that we do not insist to remove all duplicate external edges in a list (otherwise, it requires too much time). We only remove those that are easy to detect.

Definition: Consider any two edges $\langle u,v\rangle$ and $\langle u',v'\rangle$ in a list $\mathcal{L}$. We say that $\langle u,v\rangle$ is a *detectable* duplicate external edge if $w(u',v') < w(u,v)$ and the edges $\langle v,u\rangle$ and $\langle v',u'\rangle$ both appear in another list $\mathcal{L}'$.

External edges that are duplicate but not detectable would possibly remain in $\mathcal{Q}_j$; nevertheless, we will show that removing those detectable ones is already sufficient to guarantee the correctness of our algorithm.

Process i

Input: G; B_k, where $1 \le k \le i-1$, is available at the end of the kth superstep
Output: B_i

◇ /* **Phase 0** *(Initialization)* */
Construct $\mathcal{Q}_0$ from G; $a_0 \leftarrow 0$

◇ For $j = 1$ to $\lfloor \log i \rfloor$ do /* **Phase** j */

1. $a_j \leftarrow a_{j-1} + \lceil (i-a_{j-1})/2 \rceil$; denote I_j as $B[a_{j-1}+1, a_j]$.
2. /* *Merging Step* */
   (a) Edges in I_j that are full in $\mathcal{Q}_{j-1}$ are activated to merge the lists in $\mathcal{Q}_{j-1}$. Let $\mathcal{Q}$ be the set of combined adjacency lists.
   (b) For each list $\mathcal{L} \in \mathcal{Q}$, if $\mathcal{L}$ contains at most $(2^{i-a_{j-1}}-1)^2$ edges, $h(\mathcal{L}) \leftarrow \min\{h(\mathcal{L}') \mid \mathcal{L}' \in \mathcal{Q}_{j-1}$ and $\mathcal{L}'$ composes $\mathcal{L}\}$; otherwise, remove $\mathcal{L}$ from $\mathcal{Q}$.
3. /* *Remove obsolete edges and lists from* $\mathcal{Q}$ */
   For each list $\mathcal{L} \in \mathcal{Q}$,
   (a) if $\mathcal{L}$ contains an edge in I_j that is a half edge, remove $\mathcal{L}$ from $\mathcal{Q}$;
   (b) for each edge $\langle u,v\rangle \in \mathcal{L}$, if $\langle v,u\rangle$ also appears in $\mathcal{L}$, remove both edges from $\mathcal{L}$;
   (c) for each $\langle u,v\rangle \in \mathcal{L}$, if $w(u,v) \ge h(\mathcal{L})$, remove $\langle u,v\rangle$ from $\mathcal{L}$;
   (d) if $\mathcal{L}$ becomes empty, remove it from $\mathcal{Q}$.
4. For each list $\mathcal{L} \in \mathcal{Q}$, detect and remove duplicate external edges from $\mathcal{L}$.
5. /* *Truncate each list if necessary* */
   For each list $\mathcal{L} \in \mathcal{Q}$, if $\mathcal{L}$ contains more than $2^{i-a_j}-1$ edges, retain the $2^{i-a_j}-1$ smallest ones and update $h(\mathcal{L})$ to $w(e_o)$, where e_o is the smallest edge just truncated from $\mathcal{L}$.
6. $\mathcal{Q}_j \leftarrow \mathcal{Q}$.

◇ $B_i \leftarrow \{(u,v) \mid$ there is a list in $\mathcal{Q}_{\lfloor \log i \rfloor}$ containing $\langle u,v\rangle$ or $\langle v,u\rangle\}$.

The correctness of the algorithm is left to Section 6.

Time and processor complexity: The merging of adjacency lists in Step 2(a) takes $O(1)$ time. In Step 2(b), testing the length of a list ($\le (2^{i-a_{j-1}})^2$) takes $O(i-a_{j-1})$ time. After that, all adjacency lists

left have length at most $(2^{i-a_{j-1}})^2$ and the computation in each subsequent step requires $O(i - a_{j-1})$ time. Recall that each superstep lasts for time 4c for some suitable constant. Therefore, Phase j can complete its computation in at most $(i - a_{j-1})/4 \leq (i - a_j)/2$ supersteps, satisfying the schedule defined in Section 2. Process i uses at most $n+m$ processors and $(n+m)\log n$ processors are sufficient for the whole algorithm.

Adaptation to EREW PRAM: Consider Phase j of Process i. If $B[a_{j-1}+1, a_j]$ is given, all steps can be implemented on the EREW PRAM. The concurrent read is used only in accessing the edges of $B[a_{j-1}+1, a_j]$, which may also be read by many other processes at the same time. We remove the concurrent read as follows: The time in each superstep is doubled (i.e., each superstep takes 8c time instead of 4c). Each phase is divided into two parts of equal time. The first part is to read $B[a_{j-1}+1, a_j]$ into its own memory, and the computation starts in the second part using its own copy of B's. In the kth superstep of the first part of Phase j, Process i reads $B[a_{j-1}+(4k-3), a_{j-1}+4k]$ (or $B[a_{j-1}+(4k-3), a_j]$ when $a_{j-1}+4k > a_j$). Note that each B is accessed by at most four different processes in a single superstep and hence concurrent read can be eliminated.

Linear processors: The computation is divided into $\log\log n$ stages, each stage adopts the multi-thread strategy using a linear number of processors. In particular, the kth stage involves 2^k concurrent processes (instead of $\lfloor \log n \rfloor$) and takes $O(2^k)$ time. The total time of the algorithm is still $O(\log n)$.

6 The correctness

We use an inductive argument to prove that at the end of Phase j of any Process i, $\mathcal{Q}_j$ satisfies the requirements R1 to R5. Therefore, $\mathcal{Q}_{\lfloor \log i \rfloor}$ constitutes B_i correctly. To ease the induction, we maintain an invariant for Phase j. Let us define one more notion. In a particular phase, an edge $\langle u, v\rangle$ is said to be removed on a *self* basis if it is not in $\mathcal{Q}_0$ or its removal does not trigger the removal of $\langle v, u\rangle$; i.e., $\langle u, v\rangle$ is removed in Step 2(b), 3(a), 3(c), or 5 (see the contrast between $\langle u, v\rangle$ and the edges removed in Step 3(b) or 4).

Invariants of Phase j: Consider a tree $T \in F_j$. Let $\langle u, v\rangle$ be an external edge of T and $\langle u, v\rangle$ is removed on a self basis in some Phase k, where $k \leq j$. Then,

Inv1: $|F_j\text{-}base\langle u, v\rangle| \geq 2^i$; and
Inv2: if T is represented by a list $\mathcal{L} \in \mathcal{Q}_j$, $h(\mathcal{L}) \leq w(u, v)$.

The following lemma is an immediate application of the invariants. Furthermore, Inv1 and Inv2 imply the requirements R4 and R5 of $\mathcal{Q}_j$ respectively (see Lemma 6.2).

LEMMA 6.1. For any external edge $\langle p, q\rangle$ of T, if $\langle p, q\rangle$ is primary and does not appear in $\mathcal{Q}_j$, then (i) $|F_j\text{-}base\langle p, q\rangle| \geq 2^i$ and (ii) if T is represented by a list $\mathcal{L} \in \mathcal{Q}_j$, $h(\mathcal{L}) \leq w(p, q)$.

Proof. Note that edges removed in Step 3(b) or 4 are either internal or dulpicate external. If $\langle p, q\rangle$ does not appear in $\mathcal{Q}_j$, it is removed on a self basis in some Phase k, where $k \leq j$. By Inv1, $|F_j\text{-}base\langle p, q\rangle| \geq 2^i$. If T is represented by a list $\mathcal{L} \in \mathcal{Q}_j$, by Inv2, $h(\mathcal{L}) \leq w(p, q)$. □

LEMMA 6.2. R4 and R5 follow from Inv1 and Inv2 of Phase j respectively.

Proof. **Inv1 of Phase j ⇒ R4:** Consider a list $\mathcal{L} \in \mathcal{Q}_j$ representing a tree $T \in F_j$. Assume to the contrary that there is an edge $\langle u, v\rangle$ in $K(T)$ but not in $\mathcal{L}$. As $\langle u, v\rangle$ is a primary external edge, By Lemma 6.1, $|F_j\text{-}base\langle u, v\rangle| \geq 2^i$. However, $\langle u, v\rangle \in K(T)$ implies $|F_j\text{-}base\langle u, v\rangle| < 2^i$. A contradiction occurs. Therefore, Inv1 of Phase j implies R4.

Inv2 of Phase j ⇒ R5: Let $\langle p, q\rangle$ be a primary external edge of T but $\langle p, q\rangle \notin \mathcal{L}$. By Lemma 6.1, $h(\mathcal{L}) \leq w(p, q)$ and R5 is implied. □

The following is the main theorem regarding the correctness.

THEOREM 6.1. For $0 \leq j \leq \lfloor \log i \rfloor$, the invariants of Phase j hold and $\mathcal{Q}_j$ satisfies the requirements R1, R2, and R3 at the end of Phase j.

By definition of $\mathcal{Q}_0$, the invariants of Phase 0 hold and $\mathcal{Q}_0$ satisfies all requirements at the end of Phase 0. Suppose, for $k \leq j-1$, the invariants of Phase k hold and $\mathcal{Q}_k$ satisfies all requirements at the end of Phase k. It is easy to observe that at the end of Phase j, R3 is satisfied in an obvious way because in Step 5 every list is shortened to contain at most $2^{i-a_j} - 1$ edges. In the rest of the proof, we concentrate on how to maintain the invariants and show that the requirements R1 and R2 are satisfied.

Below we study in a step-by-step fashion how the set $\mathcal{Q}$ of adjacency lists is modified and the properties of those edges removed on a self basis. We divide the proof into four parts, showing the following statements. Consider a tree $T \in F_j$.

Part I: For any external edges $\langle u, v\rangle$ of T, if $\langle u, v\rangle$ is removed in Step 2(b), 3(a), or 3(c), then $|F_j\text{-}base\langle u, v\rangle| \geq 2^i$.

Part II: At the end of Step 3, $\mathcal{Q}$ satisfies R1 and R2.

Part III: For any external edge $\langle u,v\rangle$ of T, if $\langle u,v\rangle$ is removed in Step 5, then $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

Part IV: Inv2 of Phase j.

Parts I and III together imply that Inv1 holds. By Part II, $\mathcal{Q}$ satisfies R1 and R2 at the end of Step 3. Notice that Steps 4 and 5 may further remove edges from a list in $\mathcal{Q}$, but cannot empty it. Thus, $\mathcal{Q}$ still satisfies R1 and R2 at the end of Step 5. Therefore, at the end of Phase j, we can conclude that $\mathcal{Q}_j$ satisfies R1, R2, and R3.

We begin our proofs by looking at the adjacency lists formed by merging. Let I'_j be the set of edges in I_j that are full in $\mathcal{Q}_{j-1}$. Let T be a tree in F_j. Assume that T is composed of the trees $W_1, W_2, \cdots, W_t$ in F_{j-1}. The edges in I'_j connect $W_1, W_2, \cdots, W_t$ into one or more trees, all included in T. These trees are called *clusters* of T below (see Figure 3). In Step 2(a) of Phase j, we merge the adjacency lists of $W_1, W_2, \cdots, W_t$ in F_{j-1} (if exist in $\mathcal{Q}_{j-1}$) with respect to I'_j, producing adjacency lists representing some clusters of T. Denote e_T as the minimum external edge of T and similarly e_z for a cluster Z of T. Let Z^* be the cluster of T which contains e_T as an external edge.

6.1 Part I We show that for each external edge $\langle u,v\rangle$ of T, if it is removed in Step 2(b), 3(a), and 3(c), then $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$ (see Lemmas 6.3, 6.4, and 6.5 respectively). Consider a cluster Z of T and suppose there is a corresponding adjacency list $\mathcal{L}(Z)$ in $\mathcal{Q}$ at the end of Step 2(a).

LEMMA 6.3. If $\mathcal{L}(Z)$ is discarded from $\mathcal{Q}$ in Step 2(b), then for any external edge $\langle u,v\rangle$ of T kept in $\mathcal{L}(Z)$, $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

Proof. Recall that for every tree $W \in F_{j-1}$ composing Z, $\mathcal{L}(W) \in \mathcal{Q}_{j-1}$ contains at most $2^{i-a_{j-1}} - 1$ edges. If $\mathcal{L}(Z)$ is discarded in Step 2(b), then Z is composed of at least $2^{i-a_{j-1}}$ trees from F_{j-1}. As each tree in F_{j-1} contains at least $2^{a_{j-1}}$ vertices, then Z, as well as T, contains at least 2^i vertices. Therefore, any external edge of T in this case has base of size at least 2^i. □

Suppose $\mathcal{L}(Z)$ is not too long and remains in $\mathcal{Q}$ at the end of Merging step. Some edges in $\mathcal{L}(Z)$ (or the whole list) may be removed in the subsequence steps. Lemma 6.4 shows that if $\mathcal{L}(Z)$ is discarded from $\mathcal{Q}$ in Step 3(a), every external edge of T kept in $\mathcal{L}(Z)$ must have base of size at least 2^i.

LEMMA 6.4. If $\mathcal{L}(Z)$ is discarded from $\mathcal{Q}$ in Step 3(a), then for any external edge $\langle u,v\rangle$ of T that is kept in $\mathcal{L}(Z)$, $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

Proof. Consider the minimum external edge $e_z = \langle p,q\rangle$ of Z. By Proposition 2.2, e_z is an edge of T or e_T. This also means that e_z is a primary external edge of a tree $W \in F_{j-1}$ composing Z. If e_z is not in $\mathcal{L}(Z)$ (i.e., not in $\mathcal{L}(W)$), by Lemma 6.1, $|F_{j-1}\text{-}base\langle p,q\rangle| \geq 2^i$ and Z, as well as T, contains at least 2^i vertices and hence $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

Suppose $\langle p,q\rangle$ appears in $\mathcal{L}(Z)$ at the end of Step 2. Since $\langle p,q\rangle$ is also a half edge in $\mathcal{Q}_{j-1}$, $\langle q,p\rangle$ is not in $\mathcal{Q}_{j-1}$ and should be removed on a self basis in some Phase k, where $k \leq j-1$. By Inv1 of Phase $(j-1)$, $|F_{j-1}\text{-}base\langle q,p\rangle| \geq 2^i$ and so does $|F_j\text{-}base\langle p,q\rangle|$. As $w(u,v) > w(q,p)$, $F_j\text{-}base\langle u,v\rangle$ includes all trees of $F_j\text{-}base\langle q,p\rangle$ and hence has size at least 2^i. □

Suppose $\mathcal{L}(Z)$ has not been discarded from $\mathcal{Q}$ in Step 3(a). Then some edges in $\mathcal{L}(Z)$ may further be removed in Step 3(b) or 3(c). We are interested in those edges removed on a self basis, i.e., in Step 3(c). Lemma 6.5 shows that if an edge in $\mathcal{L}(Z)$ has weight $\geq h(\mathcal{L}(Z))$, its base has size at least 2^i. Then any edge removed in Step 3(c) has base of size at least 2^i.

LEMMA 6.5. Consider $\langle u,v\rangle$ to be an external edge of T that is kept in $\mathcal{L}(Z)$ at the end of Step 2. If $w(u,v) \geq h(\mathcal{L}(Z))$, then $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

Proof. Let (c,d) be the edge such that $w(c,d) = h(\mathcal{L}(Z))$. Then $\langle c,d\rangle$ is either not in $\mathcal{Q}_0$ or truncated from $\mathcal{L}(X)$ for some $X \in F_k$ in (Step 5 of) Phase k, where $k \leq j-1$. By Inv1 of Phase k, $|F_k\text{-}base\langle c,d\rangle| \geq 2^i$. Let $X' \in F_k$ be a tree in $F_k\text{-}base\langle c,d\rangle$; i.e., there is a path in G connecting a vertex in X and a vertex in X' using edges smaller than $w(c,d)$. Let $T' \in F_j$ such that $X' \subseteq T'$. Observe that the same path connects the same pair of vertices in T and T' respectively and every edge on this path is smaller than $w(c,d) \leq w(u,v)$. By definition of base, $F_j\text{-}base\langle u,v\rangle$ contains T'. Therefore, $|F_j\text{-}base\langle u,v\rangle| \geq |F_k\text{-}base\langle c,d\rangle| \geq 2^i$. □

6.2 Part II Using the invariants of Phase $(j-1)$, we can deduce the following properties of a cluster Z, which is assumed to be represented by a list $\mathcal{L}(Z) \in \mathcal{Q}$ at the end of the Merging Step (i.e., Step 2).

LEMMA 6.6. At the end of Step 2, **(P1)** for any primary external edge $\langle u,v\rangle$ of Z, if $\langle u,v\rangle \notin \mathcal{L}(Z)$, then $h(\mathcal{L}(Z)) \leq w(u,v)$; **(P2)** for any internal edge (u,v) of Z, if $\langle u,v\rangle \in \mathcal{L}(Z)$, then either $\langle v,u\rangle \in \mathcal{L}(Z)$ or $h(\mathcal{L}(Z)) \leq w(u,v)$.

Proof. **(P1):** Note that each primary external edge $\langle u,v\rangle$ of Z is that of a subtree $W \in F_{j-1}$ constituting Z. As $\langle u,v\rangle$ not in $\mathcal{L}(Z)$, it is not in $\mathcal{L}(W)$. By Lemma 6.1, $h(\mathcal{L}(W)) \leq w(u,v)$ and so does $h(\mathcal{L}(Z))$.

(P2): Let (u,v) be an internal edge of Z and $\langle u,v\rangle \in \mathcal{L}(Z)$. Suppose $\langle v,u\rangle \notin \mathcal{L}(Z)$. Then $\langle v,u\rangle$ is an external edge of a subtree $W' \in F_{j-1}$ constituting Z but $\langle v,u\rangle \notin \mathcal{L}(W')$. Then $\langle v,u\rangle$ is removed on a self basis in some Phase k, where $k \leq j-1$. By Inv2 of Phase $(j-1)$, $h(\mathcal{L}(W')) \leq w(u,v)$ and so does $\mathcal{L}(Z)$. □

We show that for any cluster Z that $e_Z \neq e_T$, $\mathcal{L}(Z)$ is either removed in Step 3(a) or is emptied by Steps 3(b) and 3(c) and afterward removed in Step 3(d). Therefore, at the end of Step 3, there is at most one cluster of T whose adjacency list is remained in $\mathcal{Q}$. Thus, each list in $\mathcal{Q}$ represents uniquely a tree in F_j.

LEMMA 6.7. For every cluster Z, if $e_Z \neq e_T$, the adjacency list of Z (if exists) must be removed in Step 3.

Proof. As $e_Z \neq e_T$, by Proposition 2.2, e_Z is an edge in T and hence it belongs to I_j. Note that e_Z is a half or lost edge in $\mathcal{Q}_{j-1}$. At the end of Step 2, if $e_Z \in \mathcal{L}(Z)$ (i.e., a half edge), $\mathcal{L}(Z)$ is removed in Step 3(a).

Suppose $e_Z \notin \mathcal{L}(Z)$. By P1 of Lemma 6.6, $h(\mathcal{L}(Z)) \leq w(e_Z)$. Since every external edge of Z has weight $\geq w(e_Z)$, Step 3(c) removes all external edges of Z left in $\mathcal{L}(Z)$. For each internal edge (a,b) of Z, if $\langle a,b\rangle$ and $\langle b,a\rangle$ appears in $\mathcal{L}(Z)$, both of them are discarded in Step 3(b). If either one of them appears in $\mathcal{L}(Z)$, P2 of Lemma 6.6 guarantees that $w(a,b) \geq h(\mathcal{L}(Z))$ and hence it is removed in Step 3(c). Therefore, $\mathcal{L}(Z)$ is empty afterwards and is removed in Step 3(d). □

Next we show that if T contains less than 2^i vertices, there is a cluster of T whose adjacency list survives at the end of Step 3. Recall that Z^* is the cluster of T which contains e_T as an external edge. By Lemma 6.7, $\mathcal{L}(Z^*)$ is the only possible cluster of T whose adjacency list can remain at the end of Step 3.

LEMMA 6.8. At the end of Step 3, if T contains less than 2^i vertices, $\mathcal{L}(Z^*)$ in $\mathcal{Q}$ representing T.

Proof. Since T contains less than 2^i vertices, $K(T)$ is non-empty. Recall that for any edge $\langle u,v\rangle$ in $K(T)$, $|F_j\text{-}base\langle u,v\rangle| < 2^i$. We know that any external edge of T removed in Step 2(b), 3(a), or 3(c) has base of size at least 2^i (see Lemmas 6.3, 6.4, and 6.5 respectively). In other words, none of the edges in $K(T)$ has been removed in these steps. Then there is exactly one cluster of T whose adjacency list remains in $\mathcal{Q}$ at the end of Step 3 for representing T. Since $\mathcal{L}(Z^*)$ is the only possible list that can survive at the end of Step 3, $\mathcal{L}(Z^*)$ remains in $\mathcal{Q}$ representing T. □

Now we argue that at the end of Step 3, $\mathcal{Q}$ satisfies R2 by showing that if $\mathcal{L}(Z^*)$ is in $\mathcal{Q}$, every edge in $\mathcal{L}(Z^*)$ is an external edge of T (where T may contain 2^i or more vertices).

LEMMA 6.9. At the end of Step 3, every edge in $\mathcal{L}(Z^*)$ is an external edge of T.

Proof. Note that at the end of Step 3(c), every remaining edge in $\mathcal{L}(Z^*)$ has weight smaller than $h(\mathcal{L}(Z^*))$. Let $\langle u,v\rangle$ be an edge in $\mathcal{L}(Z^*)$. Assume to the contrary that $\langle u,v\rangle$ is an internal edge of T. By Lemma 6.6, $\langle u,v\rangle$ is an external edge of Z^* and as $\mathcal{L}(Z^*)$ survives at the end of Step 3(a), $\langle u,v\rangle$ is not an edge in I_j. Then v must belong to another cluster Z of T. Let P be the path in T connecting u and v. As (u,v) is not in T (i.e., not in T_G^*), $w(u,v)$ is bigger than every edge on P. In particular, there is an edge e in P that is an external edge of Z^* (which must be a half edge in I_j). Since $e \notin \mathcal{L}(Z^*)$ and e is primary external, by Lemma 6.6, $h(\mathcal{L}(Z^*)) \leq w(e) < w(u,v)$. A contradiction occurs. □

6.3 Part III Suppose at the end of Step 3, $\mathcal{L}(Z^*)$ still survives, representing T. Notice that $\mathcal{L}(Z^*)$ might contain some other edges $\langle p,q\rangle \notin K(T)$ but $|F_j\text{-}base\langle p,q\rangle| < 2^i$. It is even possible that $w(p,q)$ is smaller than some edge in $K(T)$. Such an edge must be duplicate external edges of T. The following lemma further shows that $\langle p,q\rangle$ is detectable.

LEMMA 6.10. Consider any edge $\langle p,q\rangle$ in $\mathcal{L}(Z^*)$. Suppose $\langle p,q\rangle$ is a duplicate external edge of T with $|F_j\text{-}base\langle p,q\rangle| < 2^i$. Let T' be the tree in F_j containing the vertex q. At the end of Step 3, the edge $\langle q,p\rangle$ is left in the adjacency list in $\mathcal{Q}_j$ representing T'.

Proof. Let (r,s) be the primary external edge of T and T', where $r \in T$ and $s \in T'$. As $\langle p,q\rangle$ is a duplicate external edge of T, $w(r,s) < w(p,q)$. By definition of base, $F_j\text{-}base\langle p,q\rangle = F_j\text{-}base\langle q,p\rangle \supseteq F_j\text{-}base\langle r,s\rangle$. Since $|F_j\text{-}base\langle q,p\rangle| < 2^i$, T' contains less than 2^i vertices. There is a list $\mathcal{L}' \in \mathcal{Q}_j$ representing T' at the end of Step 3.

Assume to the contrary that $\langle q,p\rangle \notin \mathcal{L}'$. Then it is removed on a self basis (because $\langle p,q\rangle$ still exists). If it is removed in some Phase k, where $k \leq j-1$, then by Inv1 of Phase $j-1$, $|F_{j-1}\text{-}base\langle q,p\rangle| \geq 2^i$ and so do $|F_{j-1}\text{-}base\langle p,q\rangle|$ and $|F_j\text{-}base\langle p,q\rangle|$. A contradiction occurs. If it is removed in Phase j, then it can only be done in Step 3(a), or 3(c). By Lemmas 6.4 and 6.5, $|F_j\text{-}base\langle q,p\rangle| \geq 2^i$ and so does $|F_j\text{-}base\langle p,q\rangle|$. It contradicts our assumption. □

Lemma 6.10 implies that after we have removed those detectable duplicate external edges from $\mathcal{L}(Z^*)$ in Step 4, each remaining edge in $\mathcal{L}(Z^*)$ that does not belong to $K(T)$ has base of size at least 2^i. In other words, edges in $K(T)$ are the smallest edges in $\mathcal{L}(Z)$. Moreover, every edge truncated in Step 5 has base of size at least 2^i.

COROLLARY 6.1. Let $\langle u,v\rangle$ be an external edge of T. If $\langle u,v\rangle$ is removed in Step 5, $|F_j\text{-}base\langle u,v\rangle| \geq 2^i$.

6.4 Part IV Finally, we show that Inv2 of Phase j holds. Suppose $\mathcal{L}(Z^*)$ of T remains in $\mathcal{Q}$ at the end of Step 3. Let $\langle u,v\rangle$ be an external edge of T that is removed on a self basis in some Phase k, where $k \leq j$. We consider whether $\langle u,v\rangle$ belongs to Z^* or other cluster of T (see Lemmas 6.11 and 6.12 respectively).

LEMMA 6.11. Let $\langle u,v\rangle$ be an external edge of T that is also an external edge of Z^*. If $\langle u,v\rangle$ is removed on a self basis in some Phase k, where $k \leq j$, then $w(u,v) \geq h(\mathcal{L}(Z^*))$.

Proof. Observe that $\langle u,v\rangle$ can be removed either in Phase k, where $k \leq j-1$, or in Step 3(c) or 5 of Phase j. In the first case, let $W \in F_{j-1}$ be the tree of which $\langle u,v\rangle$ is an external edge. By Inv2 of Phase $j-1$, $w(u,v) \geq h(\mathcal{L}(W))$, where $\mathcal{L}(W) \in \mathcal{Q}_{j-1}$. As Z^* inherits the threshold of W in Step 2(b), $h(\mathcal{L}(Z^*)) \leq h(\mathcal{L}(W))$, we have $w(u,v) \geq h(\mathcal{L}(Z^*))$.

If $\langle u,v\rangle$ is removed in Step 3(c) of Phase j, $w(u,v) \geq h(\mathcal{L}(Z^*))$ at the end of Step 3. Note that the threshold of $\mathcal{L}(Z^*)$ may change in Step 5, but it is no bigger than in Step 3(c). If $\langle u,v\rangle$ is removed in Step 5, $w(u,v) \geq w(e_o) \geq h(\mathcal{L}(Z^*))$, where e_o is the smallest edge truncated from $\mathcal{L}(Z^*)$. □

LEMMA 6.12. Let Z be any cluster of T such that $e_Z \neq e_T$. For any external edge $\langle u,v\rangle$ of T that is also an external edge of Z, $w(u,v) > h(\mathcal{L}(Z^*))$.

Proof. Let P be the shortest path in T connecting a vertex in Z to a vertex in Z^*. Let (a,b) be an edge on P such that $a \in Z^*$ but not b. Note that (a,b) is a half or lost edge in I_j. As $\mathcal{L}(Z^*)$ has not been discarded from $\mathcal{Q}$ in Step 3(a), $\langle a,b\rangle$ is not in $\mathcal{L}(Z^*)$ (otherwise, it is removed in Step 3(a)). By Lemma 6.6, $h(\mathcal{L}(Z^*)) \leq w(a,b)$.

Let T_b be the subtree of T including all the vertices connected to b without using the edge (a,b). Note that Z is a connected subtree of T_b and e_T is not an external edge of T_b. By Proposition 2.2, $\langle b,a\rangle$ is the minimum external edge of T_b. As (u,v) is also an external edge of T_b, we have $w(u,v) > w(a,b) \geq h(\mathcal{L}(Z^*))$. □

References

[1] B. Awerbuch and Y. Shiloach, New Connectivity and MSF Algorithms for Shuffle-exchange Network and PRAM, *IEEE Trans. Comput.*, 36(1987), pp. 1258-1263.

[2] B. Chazelle, A Faster Deterministic Algorithm for Minimum Spanning Trees, FOCS'97, pp. 22-31.

[3] K.W. Chong, Finding Minimum Spanning Trees on the EREW PRAM, *Proc. International Computer Symposium*, 1996, pp. 7-14.

[4] K.W. Chong and T.W. Lam, Finding Connected Components in $O(\log n \log\log n)$ time on the EREW PRAM, SODA'93, pp. 11-20.

[5] R. Cole, P.N. Klein, and R.E. Tarjan, Finding minimum spanning forests in logarithmic time and linear work using random sampling, SPAA'96, pp. 243-250.

[6] R. Cole and U. Vishkin, Approximate and Exact Parallel Scheduling with Applications to List, Tree, and Graph Problems, FOCS'86, pp. 478-491.

[7] S.A. Cook, C. Dwork, and R. Reischuk, Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.*, 1986, 15(1):87-97.

[8] H. Gabow, Z. Galil, T. Spencer and R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, *Combinatorica* 6:2 (1986) 109-122.

[9] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[10] D.B. Johnson and P. Metaxas, Connected Components in $O(\lg^{3/2}|V|)$ Parallel Time for the CREW PRAM, FOCS'91, pp. 688-697.

[11] D.B. Johnson and P. Metaxas, A Parallel Algorithm for Computing Minimum Spanning Trees, SPAA'92, pp. 363-372.

[12] D.R. Karger, *Random sampling in Graph Optimization Problems*, PhD thesis, Department of Computer Science, Stanford University, 1995.

[13] D.R. Karger, P.N. Klein, and R.E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees, *Journal of the ACM*, vol. 42 (1995), pp. 321-328.

[14] D.R. Karger, N. Nisan, and M. Parnas, Fast Connected Components Algorithms for the EREW PRAM, SPAA'92, pp. 373-381.

[15] R.M. Karp and V. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, Handbook of Theoretical Computer Science, vol A, J. van Leeuwen Ed., MIT Press, 1990, pp. 869-941.

[16] C.K. Poon and V. Ramachandran, A Randomized Linear Work EREW PRAM Algorithm to Find a Minimum Spanning Forest, *Proc. 8th Annual International Symposium on Algorithms and Computation*, 1997, pp. 212-222.

[17] R.E. Tarjan, Data Structures and Network Algorithms, SIAM, Philadelphia, Pa., 1983.

[18] R.E. Tarjan and U. Vishkin, An Efficient Parallel Biconnectivity Algorithm, *SIAM J. Comput.*, 14(1985), pp. 862-874.