Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Finding Minimum Spanning Trees on the EREW PRAM*

Ka Wong Chong†
University of Hong Kong
Hong Kong

Abstract

This paper presents a parallel algorithm for finding the minimum spanning tree of a weighted undirected graph in $O(\log n \log\log n)$ time using $n+m$ processors on the EREW PRAM. The best previous known results for this problem run in $O(\log^{1.5} n)$ time using a linear number of EREW processors, or $O(\log n \log\log n)$ time using $(n+m)m^{\epsilon}$ processors, where $\epsilon > 0$ is a constant.

1 Introduction

Let $G = (V, E)$ be a connected, weighted undirected graph. For every edge $e \in E$, denote $w(e)$ the weight of it. For any spanning tree T of G, the weight of T is the sum of the weights of all the edges in T. The minimum spanning tree of G is the one with the smallest possible weight. Without loss of generality, we can assume that the weights of the edges are all distinct and hence the minimum spanning tree of G is unique.

The problem of finding minimum spanning trees has long been an interesting problem in the sequential context. In a graph with n vertices and m edges, it can be solved in $O(m \log \beta(n, m))$ time [5, 6], where $\beta(n, m) = \min\{i \mid \log^{(i)} n \leq m/n\}$. In particular, when $m \geq n \log^{(k)} n$ for some constant k, $\beta(n, m)$ is a constant. This problem can also be solved in $O(m)$ expected time if randomization is allowed [12].

In the parallel context, it is closely related to the problem of finding the connected components of an undirected graph. Hirschberg *et al.*'s [7] algorithm, which was designed to find connected components, can be modified to compute minimum spanning trees in $O(\log^2 n)$ time using $n^2/\log n$ CREW processors. Chin *et al.* [1] explicitly gave an algorithm for this problem which improved the processor bound to $n^2/\log^2 n$ processors. After Johnson and Metaxas [9] have developed their $O(\log^{1.5} n)$ time algorithm for finding connected components, they also devise an algorithm for minimum spanning trees which runs in $O(\log^{1.5} n)$ time using a linear number of EREW processors [10]. Recently, Cole *et al.* [4] have designed a *randomized* algorithm that runs in $O(\log n)$ time using $(n+m)/\log n$ CRCW processors.

On the other hand, Karger [11] showed that the problem of finding minimum spanning trees can be reduced to that of connected components. This reduction implies that the parallel time complexity of finding minimum spanning trees is no more than that of connected components. The processor requirement, however, has to blow up by a factor of m^{ϵ} for any $\epsilon > 0$. The best known algorithm for finding connected components on the EREW PRAM is given by Chong and Lam [3], which runs in $O(\log n \log\log n)$ time using $n+m$ EREW processors. Thus, with the reduction technique, we can immediately obtain an algorithm for minimum spanning trees running in $O(\log n \log\log n)$ time using $(n+m)m^{\epsilon}$ EREW processors, where ϵ is any constant bigger than zero.

In this paper, we present a deterministic parallel algorithm for finding minimum spanning trees. It takes $O(\log n \log\log n)$ time using $n+m$ EREW processors. Our algorithm is based on the idea of growth control schedule [9, 3] and of the hooking strategy of [10].

Almost all the algorithms for finding connected components or minimum spanning trees use the *hook-and-contract* approach: Initially each vertex represents a component of itself; then every component repeatedly merges (or hooks) together to form a bigger one by finding an edge connected to other component. In the minimum spanning trees algorithms, the edge chosen by a component to merge is critical (the one with the minimum weight), while any outgoing edge can be used in that of connected components.

In those algorithms running in $o(\log^2 n)$ time [9, 3], they are always only allowed to examine the adjacency lists of vertices in $o(\log n)$ time. Thus a component may not have enough time to pick the edge with the minimum weight to merge with others. It looks unlikely to modify these algorithm to find minimum spanning trees. Johnson and Metaxas [10] improve

*This work forms a part of the Ph.D thesis of the author [2].

†The research was supported in part by the research grant 338/065/0027 of Hong Kong Research Grants Council. The current address of the author: Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Email: chong@mpi-sb.mpg.de.

their connected components algorithm [9] by working on some partial adjacent lists of the vertices. We generalize this idea and together with the growth control schedule in [3] to obtain our algorithm.

Instead of dealing with some partial adjacent lists [10], our algorithm always works on a subgraph of the input; yet it also causes our algorithm to be more elegant than that of [3]. In performing computation in a subgraph, we may, however, leave out edges which are in the minimum spanning tree of the input graph. Then a component should grow cautiously. To make sure the computation is correct, we need a scheme to control the correctness of picking the edges which must be in the minimum spanning tree. This leads to our study of *threshold* graphs. A threshold graph is a weighted undirected graph in which every vertex is associated with some predefined value (called threshold). Instead of finding the minimum spanning tree, we reduce the problem into finding a special set of edges, called d-forest, in a threshold graph in which every vertex has degree no more than d, where $d \leq n$ is an integer (see Section 4).

The rest of the paper is organized as follows: Section 2 provides some basic idea of finding minimum spanning trees in the parallel context and the framework of our algorithm will also be highlighted. Section 3 discusses the basic concept of threshold graph. Section 4 shows how to choose a "small" degree subgraph from a threshold graph such that a d-forest can easily be augmented into a d-forest of the input. Section 5 presents the whole algorithm.

2 Preliminaries

Let $G = (V, E)$. For any set of vertices $U \subseteq V$, an outgoing edge of U in G is an edge $(u, v) \in E$ such that $u \in U$ and $v \in V - U$. The *minimum outgoing edge* of U is defined to be the one with the minimum weight. Let T_G^* be the minimum spanning tree of G. The following lemma (a similar one appears in [8]) suggests a simple way to find some edges of T_G^*.

Lemma 1 For any set of vertices U of V, the minimum outgoing edge of U must be in T_G^*.

Proof: Let (u, v) be the minimum outgoing edge of U, where $u \in U$ and $v \in V - U$. Suppose on the contrary that (u, v) does not belong to T_G^*. Consider the path $\langle x_1, x_2, \cdots, x_i \rangle$ connecting u to v in T_G^*, where $x_1 = u$ and $x_i = v$ and $i \geq 3$. Since $u \in U$ and $v \notin U$, there is a vertex x_j such that $x_j \in U$ and $x_{j+1} \notin U$, for some $j \leq i-1$. The edge (x_j, x_{j+1}) is an outgoing edge of U but $w((x_j, x_{j+1})) > w((u, v))$. Then the cycle in $T_G^* \cup \{(u, v)\}$ involves both (u, v) and (x_j, x_{j+1}). Removing (x_j, x_{j+1}) from $T_G^* \cup \{(u, v)\}$ forms a spanning tree of G with weight straightly smaller than T_G^*. A contradiction occurs. □

Note that the union of the minimum outgoing edge of every subset of vertices of G is indeed the minimum spanning tree of it. Instead of enumerating all possible subsets, which are exponential in number, the following is a common schema for finding minimum spanning trees.

By Lemma 1, we can immediately find a set of edges (with size of at least $n/2$) which are in T_G^* by choosing the minimum outgoing edge of every vertex in G. These edges induce a forest in G. We partition the vertices of G according to these edges such that two vertices are in the same partition if and only if they are in the same tree. Also, each partition contains at least two vertices. If there is only one partition (i.e. one tree only), we have already found all the edges of T_G^*. Otherwise, we continue to find the edges of T_G^* in respect of the contracted graph G_c which is constructed as follows: Every tree T_i is contracted into a single vertex r_i and r_i inherits all the neighbors of T_i. Note that the minimum outgoing edge of r_i in G_c is that of T_i in G. Again, by Lemma 1, the minimum outgoing edge of every vertex in G_c is in T_G^*. Thus we can find the edges of T_G^* by repeatedly contracting the graph until there is at most one vertex remained.

The running time of this simple algorithm is $O(\log^2 n)$: A fast growing tree, containing as many as $\Omega(n)$ vertices, requires $\Omega(\log n)$ time for contracting into a single vertex; the algorithm has to iterate $\lceil \log n \rceil$ times for the slow growing one which may involve as little as two vertices.

To obtain a better time bound, we adopt the idea of growth control schedule [9, 10, 3]. We want trees with different sizes grow at different paces. A slow growing (small) tree can grow more frequently in $o(\log n)$ time while a fast growing (large) one may left idle until the slower ones can catch up. However, the time to contract a tree and to find its minimum outgoing edge depend on the total length of the adjacency lists of its vertices. There are cases that a small tree contains as little as two vertices may have an adjacency list with $\Theta(n)$ entries. As a result, the tree requires $\Theta(\log n)$ time to contract and find its minimum outgoing edge.

To alleviate this situation, our algorithm always considers a subgraph H of G in which every vertex has "small" degree and the subgraph captures those "light" edges. Intuitively, working on such subgraph is more efficient. For example, if every vertex in H has no more than s, then a tree with no more than s vertices can always be contracted in $O(\log s)$ time.

However, we face the problem that we may leave out some edges of T_G^* in H. Also, the minimum outgoing edge of a subset of vertices in H may not be in T_G^*. Thus picking the minimum outgoing edges in H become insecure.

To make sure the computation is correct, we introduce a scheme that each vertex in H keeps a value (threshold) which is the weight of the smallest edge incident to it but has been left out from G. Then a minimum outgoing edge e of a set of vertices U of H is in T_G^* if $w(e)$ is smaller than the thresholds of all the vertices in U (see Section 3).

In order to describe the algorithm in a recursive manner, we reduce the problem of finding the minimum spanning trees into finding a d-forest in a threshold graph in which every vertex has degree no more than d, where $d \leq n$. The edges in a d-forest partition the vertices of a threshold graph into a set of trees and each tree contains at least d vertices or its minimum outgoing edge does not belong to T_G^*. In Section 4, we show that by setting the threshold of each vertex in G appropriately, an n-forest of G is equal to the set of edges of T_G^*.

2.1 Framework of the algorithm

Our algorithm is implemented as a recursive procedure, called **Branches**(H, k), where H is a threshold graph and $k \leq \lceil \log \log n \rceil$ is an integer. Also, every vertex in H has degree no more than 2^{2^k}. **Branches**(H, k) finds a set of edges B which is a 2^{2^k}-forest of H. Let $d = 2^{2^k}$. Note that each tree induced by B in H contains at least d vertices or its minimum outgoing edge does not belong to T_G^*. In particular, **Branches**$(G, \lceil \log \log n \rceil)$ finds all the edges of T_G^*.

Branches(H, k) finds a d-forest of H in two phases. In Phase 1, we find a $\sqrt{d}$-forest for H. By choosing a specific subgraph H' in H with every vertex has degree no more than $\sqrt{d}$ (see Section 4), a $\sqrt{d}$-forest of H' can easily be augmented to produce a $\sqrt{d}$-forest for H. Note that some of the trees induced by the $\sqrt{d}$-forest in H may already satisfy the requirement of a d-forest of H. In Phase 2, we contract those "small" trees (nevertheless, they contain at least $\sqrt{d}$ vertices) in H and recursively find a $\sqrt{d}$-forest among the contracted vertices. Then each tree comprising at least $\sqrt{d}$ contracted vertices actually involves at least $\sqrt{d}\sqrt{d} = d$ vertices of H.

Next we are going to look at some general properties of threshold graphs.

3 Threshold graphs

Consider a subgraph H of G. With respect to H, the minimum outgoing edge of a subset of vertices may or may not be an edge in T_G^*. To identify easily which minimum outgoing edges in H are actually in T_G^*, we associate every vertex u in H with a *threshold* $f_H(u)$. The threshold of u keeps track on the weight of the smallest edge of u in G but not in H; if u's minimum outgoing edge in H has a weight less than its threshold then this edge is also u's minimum outgoing edge in G.

Definition: An edge e in H is said to be a *branch* of H if H contains a subset U of vertices such that e is the minimum outgoing edge of U in H and $w(e) < f_H(U)$, where $f_H(U)$ denotes $\min\{f_H(u) \mid u \in U\}$.

By definition, no matter how the thresholds of a graph H are chosen, a branch of H is also an edge in the minimum spanning tree of H (i.e. T_H^*). Moreover, the minimum spanning tree of G can be obtained by finding all the branches of G with some predefined thresholds—Suppose every vertex u of G is associated with a threshold $+\infty$, then every minimum outgoing edge in G is a branch and the set of branches of G forms T_G^*.

In general, our algorithm finds the branches for a threshold graph H by recursively finding branches for two kinds of smaller threshold graphs: a contracted graph of H and a subgraph of H. Below, we show how to set the thresholds in these smaller graphs so that the branches found in these graphs are actually branches of H.

Contracted graph: Consider a contracted graph H_c of H, in which each vertex u is contracted from a subset U of vertices of H and u inherits all neighbors of the vertices in U (if U contains more than one outgoing edge to the same vertex then u retains the one with the smallest weight). Also, u inherits the thresholds of all vertices in U; that is, $f_{H_c}(u) = \min\{f_H(v) \mid v \in U\}$. This is to ensure every branch of H_c is also a branch of H.

Lemma 2 Every branch of H_c is also a branch of H.

Proof: Let e be a branch of H_c. Suppose e is the minimum outgoing edge of some subset W of vertices in H_c and $w(e) < f_{H_c}(W)$. For each vertex $u_i \in W$, let U_i be the set of vertices of H contracted to u_i, then $f_{H_c}(u_i) = \min\{f_H(v) \mid v \in U_i\}$. Let $X = \bigcup_{u_i \in W} U_i$. Note that e is the minimum outgoing edge of W and $w(e) < f_{H_c}(W) = \min\{f_{H_c}(u_i) \mid u_i \in W\} = f_H(X)$. Thus e is also a branch of H. □

Subgraph: Next, we study the case for setting the thresholds for a subgraph H' of H so that every branch of H' is also a branch of H. Intuitively, we do not want a vertex to pick an edge in H' if an even smaller edge has been left out. The threshold of each

vertex u in H', i.e. $f_{H'}(u)$, is defined as $\min(\{f_H(u)\} \cup \{w(e) \mid e \text{ is incident to } u \text{ in } H \text{ but } e \notin H'\})$.

Lemma 3 Every branch of H' is also a branch of H.

Proof: Assume on the contrary that H' has a branch e' which is not a branch of H. Let U be the subset of vertices of H' such that the minimum outgoing edge of U in H' is e'. Then $w(e') < f_{H'}(U)$. Note that e' is also an outgoing edge of U in H. Since $f_{H'}(U) \leq f_H(U)$, we have $w(e') < f_H(U)$. If e' is not a branch of H, the minimum outgoing edge of U in H is another edge $e = (u, v)$ for some $u \in U$. That is, $w(e) < w(e')$ and $e \notin H'$. By definition, $f_{H'}(u) \leq w(e)$ and $f_{H'}(U) \leq w(e)$. Therefore, $w(e') > w(e) \geq f_{H'}(U)$. A contradiction occurs. □

3.1 Exhausted partitions of H

Let us look at another property of a threshold graph. Suppose we have found some branches of H. Denote S the set of these branches. The vertices of H, in respect of S, are partitioned into a set of trees $\{T_1, T_2, \cdots, T_l\}$, where $l \geq 1$. For each T_i, if its minimum outgoing edge e in H carries a weight less than $f_H(T_i)$, we can report e as another branch of H and e can merge T_i with another tree $T_{i'}$, where $i' \leq l$. Notice that such an edge may not exist for some T_i since T_i may have no outgoing edge or the weight of the minimum outgoing edge of T_i is greater than $f_H(T_i)$.

Definition: T_i is said to be *exhausted* if T_i has no outgoing edge or the weight of the minimum outgoing edge of T_i is greater than $f_H(T_i)$

In this case, we may not be able to merge T_i with any other tree. For those T_i's that are not exhausted, their minimum outgoing edges are branches of H. Consider Q to be a set comprising one or more of such branches. $S \cup Q$ defines another set of trees $\{R_1, R_2, \cdots, R_k\}$ partitioning the vertices of H. Each R_j is actually composed of one or several T_i's linked by their minimum outgoing edges. Note that each R_j may or may not be exhausted.

Lemma 4 Suppose R_j is composed of trees $T_{j_1}, T_{j_2}, \cdots, T_{j_r}$ linked by their minimum outgoing edges. If there is a T_{j_i} that is exhausted, for some $i \leq r$, then R_j is also exhausted in H.

Proof: Let $F = \{T_{j_1}, T_{j_2}, \cdots, T_{j_r}\}$. Note that R_j can have at most one exhausted tree in F because each tree in F that is not exhausted contributes one edge in R_j and there should be $r - 1$ of them.

Let F_1 be the set of all trees in F that their minimum outgoing edges are incident on T_{j_i}. Let T_{j_a} be one of them and note that the minimum outgoing edge e_{j_a} of T_{j_a} is also an outgoing edge of T_{j_i}. As T_{j_i} is exhausted, $w(e_{j_a}) > f_H(T_{j_i})$ and because $w(e_{j_a}) < f_H(T_{j_a})$, we have $f_H(T_{j_i}) < f_H(T_{j_a})$.

Let F_2 be the set of all trees in $F - F_1$ that their minimum outgoing edges are incident on some tree of F_1. Let T_{j_b} be a tree in F_2 and T_{j_a} in F_1 such that the minimum outgoing edge e_{j_b} of T_{j_b} is incident on T_{j_a}. Similarly, we can show that $w(e_{j_b}) > w(e_{j_a}) > f_H(T_{j_i})$ and $f_H(T_{j_i}) < f_H(T_{j_b})$.

Repeatedly applying the same argument, we can show that for every tree T in F, the minimum outgoing edge e of T has weight greater than $f_H(T_{j_i})$ and $f_H(T_{j_i}) \leq f_H(T)$. Then $f_H(R_j) = f_H(T_{j_i})$. Note that the minimum outgoing edge e_j of R_j has weight greater than that of any $T \in F$. Therefore, $w(e_j) > f_H(T_{j_i}) = f_H(R_j)$ and R_j is exhausted. □

4 Finding a d-forest

Let H be a threshold graph. All the branches of H can be computed in $O(\log^2 n)$ time using standard algorithms for minimum spanning trees [7, 1, 8]. However, it is not a trivial task to do it in $o(\log^2 n)$ time. In the following, we define the notion of a d-forest of H that can be computed efficiently in a recursive manner.

Definition: Consider S to be a set of branches of H. Suppose the vertices of H, with respect to S, are partitioned into the trees $\{T_1, T_2, \cdots, T_l\}$. S is said to be a *d-forest* if every T_i not exhausted in H contains at least d vertices.

For example, the set of the minimum outgoing edge of each vertex $u_i \in H$ which is not exhausted forms a 2-forest of H (see Lemma 8). On the other hand, the minimum spanning tree of G can be obtained by finding an n-forest in G in which every vertex is associated with a threshold $+\infty$ (see Section 5).

With a d-forest, we have a tight control on the partitioning of H. Basically, we do not want to have a tree T_i composed of very few ($< d$) vertices yet it can be merged with other trees through some "trivial" branches (the minimum outgoing edge of T_i).

To find a d-forest for H, we consider a particular subgraph H' in which every vertex has "small" degree. Intuitively, working on H' is more efficient. Moreover, H' is chosen in such a way that any d-forest of H' can easily be augmented to produce a d-forest for H. We call such subgraph a d-light subgraph of H, its definition is as follows:

Definition: **(i)** An edge $e \in H$ is said to be d-light if each of its end-points (say, u) contains e as one of the d smallest edges of it (i.e. $|\{e' \in H \mid e' \text{ is incident to } u;\ w(e') < w(e)\}| < d$). **(ii)** The d-light subgraph of H includes all the vertices of H and

all the d-light edges of H. Observe that the degree of every vertex in H' is at most d. The thresholds of H' are set as described in Section 3.

Next, let us see how a d-forest of H' can be used to form a one for H. Consider a d-forest S' of H'. Suppose S' partitions the vertices of H' (or equivalently, H) into a set of trees $F = \{T_1, T_2, \cdots, T_l\}$. Note that any T_i exhausted in H' may not be exhausted in H. If T_i is exhausted in H or T_i contains more than d vertices, then it satisfies the requirement of a d-forest with respect to H. We only need to worry with those trees that do not fulfill the requirement. The following lemma states that, for such a tree the minimum outgoing edge in H is not included in H' (i.e., not a d-light edge); thus we can add one more branch to H to merge this tree with other trees.

Lemma 5 Let $T_i \in F$ be a tree that is exhausted in H' and contains less than d vertices. Then the minimum outgoing edge of T_i in H is not d-light in H.

Proof: Let e be the minimum outgoing edge of T_i in H. T_i is not exhausted in H, so $w(e) < f_H(T_i)$, i.e. for any $u \in T_i$, $w(e) < f_H(u)$. Assume on the contrary that e is d-light and is included in H'. Then e is an outgoing edge of T_i in H'. Since T_i contains less than d vertices, T_i must be exhausted in H' and $w(e) > f_{H'}(T_i)$. In other words, there is a vertex $v \in T_i$ such that $w(e) > f_{H'}(v)$. Since $w(e) < f_H(v)$, $f_{H'}(v)$ should be less than $f_H(v)$. In this case, there is an edge $e_v \in H$ but not in H' incident to v such that $f_H(v) > w(e_v)$ and $f_{H'}(v) = w(e_v)$. As a result, $w(e) > f_{H'}(v) = w(e_v)$. On the other hand, e_v is not the minimum outgoing edge of T_i in H, thus e_v is an internal edge of T_i. One of the end-points of e_v must have at least d edges, each with weight less than $w(e_v)$. These edges are also internal edges of T_i in H (otherwise, e should not be the minimum outgoing edge of T_i). Therefore, T_i contains at least d vertices. A contradiction occurs. □

In respect of H, consider all T_i's that are not exhausted and contain less than d vertices; let Q be the set of the minimum outgoing edges of *all* these T_i's. Again, $S' \cup Q$ partitions the vertices of H into a set of trees $\{R_1, R_2, \cdots, R_k\}$. The following lemma shows that $S' \cup Q$ is a d-forest of H.

Lemma 6 For each R_j, if R_j is not exhausted in H then R_j contains at least d vertices.

Proof: Let R_j be a tree not exhausted in H. Assume on the contrary that R_j contains less than d vertices. R_j is composed of $x \geq 2$ trees $T_{j_1}, T_{j_2}, \cdots, T_{j_x}$ linked by their minimum outgoing edges (which are in Q) and contains exactly $x - 1$ edges. Each T_{j_i} is not exhausted in H (otherwise, by Lemma 4, R_j is exhausted). Also, each T_{j_i} contains less than d vertices. We conclude that each T_{j_i} should have contributed its minimum outgoing edges in H to Q. All these edges are now internal edges of R_j. As R_j contains $x - 1$ edges of Q, there must exist two distinct trees T_{j_i} and $T_{j_{i'}}$ such that their minimum outgoing edges in H actually refer to the same edge. Let $e = (u, v)$ be this edge, where $u \in T_{j_i}$ and $v \in T_{j_{i'}}$. By Lemma 5, e is not a d-light edge, or equivalently, e is not one of the d smallest edges of one of e's end points, say, u. On the other hand, e is the minimum outgoing edge of T_{j_i}, any edge incident to u with weight less than $w(e)$ must be internal edges of T_{j_i}, and there are at least d such edges. Therefore, there are at least d vertices in T_{j_i}. A contradiction occurs. □

By Lemma 6, $S' \cup Q$ is a d-forest of H.

4.1 A generalization

Now we consider a more general scenario in which our algorithm will encounter. Again, consider H to be a weighted undirected graph with predefined thresholds. Let U be a set of distinguished vertices in H. Intuitively, U denotes some vertices that we are not interested to process at this moment (for example, U includes vertices that are exhausted in H). We only want to consider the smaller graph $H \setminus U$ and its d-light subgraph, where $H \setminus U$ refers to the subgraph of H induced by the vertices of H that are not in U.

Let H' be the d-light subgraph of $H \setminus U$ and S' a d-forest of H'. Consider the partitioning of the vertices of $H \setminus U$ with respect to S'. Let T be one of the trees induced. With respect to H, if T is not exhausted and T contains less than d vertices, we would like to add the minimum outgoing edge of T to H (instead of $H \setminus U$). Let Q be the set of all these minimum outgoing edges. Note that Q, unlike S', may involve edges with end-points in U.

Lemma 7 Let R be any tree induced by $S' \cup Q$ on the vertices of H that does not involve any vertex in U. If R is not exhausted in H then T contains at least d vertices.

Proof: The proof is similar to that of Lemma 6. □

5 The algorithm

Our algorithm is implemented as a recursive procedure called `Branches`(H, k); the input H is a threshold graph in which the degree of every vertex is at most 2^{2^k}, and the output is a 2^{2^k}-forest of H. Given a graph G with n vertices and m edges, we

find the minimum spanning tree of G by executing **Branches**(G, k_0), where $k_0 = \lceil \log \log n \rceil$ and the threshold of every vertex of G is $+\infty$. As to be shown later, **Branches**(G, k_0) requires time $O(k_0 2^{k_0})$ (i.e. $O(\log n \log \log n)$) using $n+m$ EREW processors.

Basically, for any $k > 0$, **Branches**(H, k) works in two phases. Let $p = 2^{2^k}$ (and then $\sqrt{p} = 2^{2^{k-1}}$). In Phase I, we find a $\sqrt{p}$-forest of H recursively. Note that some of the trees induced by this $\sqrt{p}$-forest may have already satisfied the requirement for a p-forest for H, i.e. they are exhausted in H or contain at least p vertices. We only need to worry those "small" trees that are not exhausted (nevertheless, each of them contains at least $\sqrt{p}$ vertices). In Phase II, we contract every such "small" tree into a single vertex and find a $\sqrt{p}$-forest among these contracted vertices recursively. Roughly speaking, each tree induced by this new $\sqrt{p}$-forest, if not exhausted, composed of at least $\sqrt{p}$ contracted vertices, thus involving at least p vertices of H.

procedure **Branches**(H, k)
Input: a threshold graph H in which the degree of every vertex is at most $p\,(= 2^{2^k})$
Output: a p-forest of H

if $k = 0$, find the minimum outgoing edge of every vertex of H which is not exhausted. Let S be the set of such edges. **return** S. (S is a 2-forest of H.)

else (i.e. $k > 0$)

Phase I

(a) Let U_1 be the set including all vertices that are exhausted in H.

(b) Let H' be the $\sqrt{p}$-light subgraph of $H \setminus U_1$. Find a $\sqrt{p}$-forest S_1 of H' by invoking **Branches**$(H', k-1)$.

(c) For each tree induced by S_1 on the vertices of $H \setminus U_1$, if it is not exhausted in H and contains less than $\sqrt{p}$ vertices, find its minimum outgoing edge in H. Let Q_1 be the set of these edges. ($S_1 \cup Q_1$ is a $\sqrt{p}$-forest of H.)

Phase II

(a) Let $R_1, R_2, \cdots, R_l$ be the trees induced by $S_1 \cup Q_1$ on the vertices of H. Let U_2 be the set of vertices belonging to some R_i that is exhausted or contains at least p vertices.

(b) Construct a graph H_c from H by contracting every tree R_i that is not exhausted and contains less than p vertices into a single vertex. All vertices in U_2 remain in H_c.

(c) Let H'' be the $\sqrt{p}$-light subgraph of $H_c \setminus U_2$. Find a $\sqrt{p}$-forest S_2 of H'' by invoking **Branches**$(H'', k-1)$.

(d) For each tree induced by S_2 on the vertices of $H_c \setminus U_2$, if it is not exhausted in H_c and contains less than $\sqrt{p}$ vertices, find its minimum outgoing edge in H_c. Let Q_2 be the set of these edges. ($S_1 \cup Q_1 \cup S_2 \cup Q_2$ is a p-forest of H.)

return $S_1 \cup Q_1 \cup S_2 \cup Q_2$.

Let us explain why **Branches**(H, k) computes a p-forest of H. First of all, we look at the case when $k = 0$. **Branches**$(H, 0)$ reports a set S comprising the minimum outgoing edges of all vertices not exhausted in H.

Lemma 8 S is a 2-forest of H.

Proof: Let T be a tree induced by S on the vertices of H. If T is not exhausted in H, by Lemma 4, every vertex in T is not exhausted in H; thus, every vertex should have contributed its minimum outgoing edge to S and T must contain at least two vertices. Therefore, S is a 2-forest of H. □

Assume that **Branches**$(H, k-1)$ finds a $\sqrt{p}$-forest of H correctly. Then we can show that **Branches**(H, k) does find a p-forest of H. Our proof is divided into two parts in respect of the two phases of **Branches**(H, k).

Phase I: In Phase I, we construct two sets of branches, S_1 and Q_1, of H. Let $F = \{R_1, R_2, \cdots, R_l\}$ be the set of trees induced by $S_1 \cup Q_1$ on the vertices of H.

Lemma 9 $S_1 \cup Q_1$ is a $\sqrt{p}$-forest of H.

Proof: Consider any R_i in F. If R_i involves no vertex of U_1, then, by Lemma 7, R_i is either exhausted or contains at least $\sqrt{p}$ vertices. In case R_i involves some vertex in U_1, we show that R_i must be exhausted in H. Observe that such R_i is composed of some trees T_i's induced by S_1 on the vertices of $H \setminus U_1$, as well as a vertex of U_1, all linked by some edges of Q_1, which are the minimum outgoing edges of T_i's. Each vertex $u \in U_1$ can be regarded as a one-vertex tree exhausted in H. By Lemma 4, R_i is exhausted in H.

In conclusion, if each R_i is not exhausted in H then it contains at least $\sqrt{p}$ vertices. Thus, $S_1 \cup Q_1$ is a $\sqrt{p}$-forest of H. □

Phase II: In Phase II, we work on a smaller graph H_c, which is constructed from H by contracting the vertices of every R_i that is not exhausted and contains less than p vertices into a single vertex. We call such a vertex in H_c a contracted vertex, it corresponds to at least $\sqrt{p}$ vertices of H. Note that the vertices of U_2, i.e. the vertices of those R_i's that are exhausted or contain at least p vertices, all remain in H_c. Based on $H_c \setminus U_2$, we compute two sets of branches, S_2 and Q_2, for H_c.

Lemma 10 $S_1 \cup Q_1 \cup S_2 \cup Q_2$ is a p-forest of H.

Proof: Consider the graph H_c. Let $L \subseteq S_1 \cup Q_1$ be the branches that make up U_2. Let Z be a tree induced by $S_2 \cup Q_2 \cup L$ in H_c. Recall that each vertex in $H_c \setminus U_2$ is contracted from some vertices (at least $\sqrt{p}$) of H. By expanding each contracted vertex, we can find the corresponding tree of Z, say, Y, induced by $S_1 \cup Q_1 \cup S_2 \cup Q_2$ in H and vice versa. We consider whether Z involves any vertex in U_2.

Z does not involve any vertex of U_2: By Lemma 7, Z is exhausted or Z contains at least $\sqrt{p}$ vertices of H_c. In the former case, Y is also exhausted in H. In the latter case, Y contains at least $\sqrt{p}\sqrt{p} = p$ vertices of H as each vertex in $H_c \setminus U_2$ is contracted from at least $\sqrt{p}$ vertices of H.

Z involves some vertices of U_2: Let $u \in U_2$ be such a vertex. According to the construction, u is participated in a tree R_i induced by $S_1 \cup Q_1$ in H and R_i is either exhausted or has at least p vertices of H. If R_i contains at least p vertices of H, the corresponding tree Y of Z, of course, contains at least p vertices. Otherwise, we show that Z is exhausted in H_c. Observe that Z is formed by connecting R_i and a set of trees $\{X_1, X_2, \cdots, X_r\}$ induced by S_2 in H_c using the minimum outgoing edges of the X_j's (which are edges of Q_2). By Lemma 4, Z is exhausted in H_c. Hence, Y is exhausted in H.

In conclusion, if each Y is not exhausted in H, then it contains at least p vertices. Therefore, $S_1 \cup Q_1 \cup S_2 \cup Q_2$ is a p-forest of H. □

Theorem 11 **Branches**(G, k_0) finds the minimum spanning tree of G, where $k_0 = \lceil \log \log n \rceil$.

Proof: For every vertex in G, its degree is at most $n \leq 2^{2^{k_0}}$ and its threshold is $+\infty$. By Lemma 10, **Branches**(G, k_0) finds a $2^{2^{k_0}}$-forest of G. Denote S this $2^{2^{k_0}}$-forest. By definition, every branch in a $2^{2^{k_0}}$-forest is an edge in the minimum spanning tree of G. Below, we show that the number of trees induced by S on the vertices of G is exactly one; thus, S spans all the vertices of G.

Suppose on the contrary that S induces more than one tree on G. Let T be any one of them. As G is connected, T has at least one outgoing edge. Note that $f_G(T) = +\infty$. T is not exhausted and contains at least $2^{2^{k_0}}$ vertices. Since G is partitioned into more than one of such trees, it must contain at least $2^{2^{k_0}} + 1 > n$ vertices. A contradiction occurs. □

Time complexity: We show that **Branches**(H, k) can be done in $O(k2^k)$ time. Let $F(k)$ be the time complexity of **Branches**(H, k). For $k = 0$, **Branches**$(H, 0)$ just finds the minimum outgoing edge of every vertex. As every vertex has a constant degree and the branch can be found in constant time using n EREW processors. Thus $F(0) = b$ for some constant $b > 0$.

Consider the case for $k \geq 1$. In Phase I, Step(a), identifying the exhausted vertices U_1 in H can be done in $O(\log p)$ time by examining the adjacency list of the vertices. The $\sqrt{p}$-light subgraph of $H \setminus U_1$ can be constructed as follows: Each vertex selects the first $\sqrt{p}$ smallest edges from its adjacency list. Then an edge e is in H' if both of its copy are selected by its end-points. Thus H' can be created in $O(\log p)$ time using a linear number of processors. After finding the $\sqrt{p}$-forest S_1 in H', we determine whether the size of each (unrooted) tree induced by S_1 is smaller than p vertices. We construct a combined adjacent list for each tree induced by S_1 in H in constant time (see Appendix). Let T be one of the tree induced. We then test the length of the list using $O(\log p)$ time. Note that every vertex in H has degree no more than p. If T contains no more than p vertices, its combined adjacent list contains no more than p^2 entries and the list can be contracted in the time allowed. Otherwise, T must contain more than p vertices and we do not need to take care of them in the rest of **Branches**(H, k). We find the minimum outgoing edges of those "small" trees (that are not exhausted and have no more than p vertices) in H in Step(c). The minimum outgoing edge of T can be found in $O(\log p)$ time. As a result, Phase I, except for the recursive call to **Branches**$(H', k-1)$, can be done in $O(\log p)$ time using $n + m$ processors. The computation in Phase II are similar to that of Phase I and also can be done within the same resource bound.

Hence the time used in all steps of **Branches**(H, k), excluding the two calls of **Branches**$(H, k-1)$, is $b2^k$ and therefore

$$F(k) = 2F(k-1) + b2^k$$

$$
\begin{aligned}
&= 2^2 F(k-2) + 2b\,2^k \\
&\vdots \\
&= 2^k F(0) + k\,b\,2^k \\
&= O(k2^k).
\end{aligned}
$$

For $k_0 = \lceil \log\log n \rceil$, $F(k_0) = O(\log n \log\log n)$. Therefore, the algorithm runs in $O(\log n \log\log n)$ time using $n + m$ EREW processors.

References

[1] F.Y. Chin, J. Lam, and I-N. Chen, Efficient Parallel Algorithms for some Graph Problems, *Comm. ACM*, 25(1982), pp. 659-665.

[2] K.W. Chong, *Improved Algorithms for Some Classical Graph Problems*, Ph.D thesis, Department of Computer Science, University of Hong Kong, 1996.

[3] K.W. Chong and T.W. Lam, Finding Connected Components in $O(\log n \log\log n)$ time on the EREW PRAM, *J. Algorithms*, 1995, pp. 378-402.

[4] R. Cole, P.N. Klein, and R.E. Tarjan, Finding minimum spanning forests in logarithmic time and linear work using random sampling, *SPAA* 1996, pp. 243-250.

[5] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their used in improved network optimization algorithms, *J. ACM*, 1987, 34(3):596-615.

[6] H.N. Gabow, Z. Galil, T. Spencer, Efficient implementation of graph algorithms using contraction, *FOCS* 1984, pp. 347-357.

[7] D.S. Hirschberg, A.K. Chandra, and D.V. Sarwate, Computing Connected Components on Parallel Computers, *Comm. ACM*, 22(1979), pp. 461-464.

[8] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992, pp. 180.

[9] D.B. Johnson and P. Metaxas, Connected Components in $O(\lg^{3/2}|V|)$ Parallel Time for the CREW PRAM, *FOCS* 1991, pp. 688-697.

[10] D.B. Johnson and P. Metaxas, A Parallel Algorithm for Computing Minimum Spanning Trees, *J. Algorithms*, 1995, pp. 383-401.

[11] D.R. Karger, Approximating, verifying, and constructing minimum spanning forests, *manuscript*, 1992.

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

Appendix

Let H be a graph in which every vertex has degree of no more than s^c for some constant $c > 0$. Let S be a set of branches of H. We want to form a combined adjacency list for each tree T induced by S, in constant time using a linear number of EREW processors.

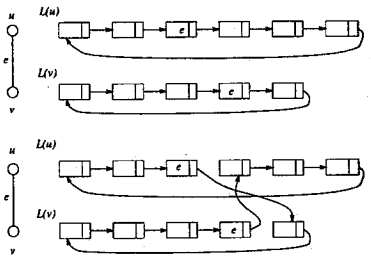

Figure 1:

We assume that the graph is given in the form of adjacency lists. We first construct a combined adjacency list (a circular linked list) for each tree T in constant time. We adopt the Euler tour technique [8] as follows: For each edge $e \in S$, let u and v be its endpoints in H and $L(u)$ and $L(v)$ the adjacency lists of u and v respectively. In particular, $L(u)$ and $L(v)$ are circular linked lists, i.e., the last entry of each list is pointed to the first entry. The entry of e in $L(u)$ set its *next* pointer to the next entry of e in $L(v)$, and vice versa (see Figure 1). Figure 2 gives an example of a combined adjacency list of a tree.

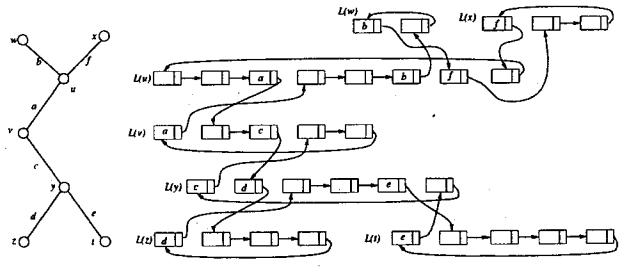

Figure 2:

The combined adjacency list of each tree actually is an Euler tour of it.