1996

Optimal Algorithms for the Single and Multiple Vertex Updating Problems of a Minimum Spanning Tree

Donald B. Johnson
Dartmouth College

P. Takis Metaxas
Wellesley College, pmetaxas@wellesley.edu

Recommended Citation

Optimal Algorithms for the Single and Multiple Vertex Updating Problems of a Minimum Spanning Tree, with D.B. Johnson. Algorithmica, 16 (6):633-648 (1996).

Optimal Algorithms for the Single and Multiple Vertex Updating Problems of a Minimum Spanning Tree

Donald B. Johnson[*] Panagiotis Metaxas[†]
Dartmouth College[‡]

Abstract

The vertex updating problem for a minimum spanning tree (MST) is defined as follows: Given a graph $G = (V, E_G)$ and an MST T for G, find a new MST for G to which a new vertex z has been added along with weighted edges that connect z with the vertices of G. We present a set of rules that produce simple optimal parallel algorithms that run in $O(\lg n)$ time using $n/\lg n$ EREW PRAM processors, where $n = |V|$. These algorithms employ any valid tree-contraction schedule that can be produced within the stated resource bounds. These rules can also be used to derive simple linear-time sequential algorithms for the same problem. The previously best known parallel result was a rather complicated algorithm that used n processors in the more powerful CREW PRAM model. Furthermore, we show how our solution can be used to solve the multiple vertex updating problem: Update a given MST when k new vertices are introduced simultaneously. This problem is solved in $O(\lg k \cdot \lg n)$ parallel time using $\frac{k \cdot n}{\lg k \cdot \lg n}$ EREW PRAM processors. This is optimal for graphs having $\Omega(kn)$ edges.

Keywords: Optimal parallel and sequential algorithms, EREW PRAM model, vertex updating, minimum spanning tree.

[*]email address: djohnson@cardigan.dartmouth.edu

[†]Current address: Department of Computer Science, Wellesley College, Wellesley, MA 02181-8289, email: takis@bambam.wellesley.edu

[‡]Department of Mathematics and Computer Science, Hanover, NH 03755-3551

Address for Correspondence: Panagiotis T. Metaxas, Department of Computer Science, Wellesley College, Wellesley MA 02181-8289.

1 Introduction

Definition. The vertex updating problem for a minimum spanning tree (MST) is defined as follows: We are given a weighted graph $G = (V, E_G)$, along with an MST $T = (V, E)$. The graph is augmented by a new vertex z and n weighted edges connecting z to every vertex in V. We want to compute a new MST $T' = (V \cup \{z\}, E')$.

In this paper we present an optimal yet simple solution for the EREW PRAM model. Our solution works in $O(\lg n)$ time ($\lg n$ denotes $\log_2 n$) using $n/\lg n$ parallel processors, where $n = |V|$, the number of vertices in the graph. Brent's theorem applies and implies that, using $p \leq n/\log n$ processors, the running time is $O(\lg n + n/p)$. In the last section, we show how this solution can be used to solve the multiple vertex updating problem (an existing MST T is updated simultaneously by k new vertices having weighted edges to T).

The model of parallel computation we will use throughout this paper is the EREW PRAM (exclusive-read-exclusive-write parallel random access machine) [6]. The virtue of an algorithm in the EREW model is that no arbitration of concurrent access requests need be provided in the machine on which it runs. Such arbitration is necessary in the more powerful CREW and CRCW models employed in all previous work on these problems.

We introduce a set of rules that make use of a few simple observations on MSTs as well as of the fact that it is preferable to break the cycles introduced by the new edges (because there is only a polynomial number of them) than to compute the tree from scratch. These rules are defined to apply locally on the nodes of the existing MST, so we get parallel algorithms (using tree-contraction) as well as sequential ones (using, for example, depth-first-search).

History. The vertex updating problem of a minimum spanning tree was first addressed by Spira and Pan in [1], where an $O(n)$ sequential algorithm was presented. Another solution using depth-first-search and having the same time complexity was later given by Chin and Houck in [2], while Pawagi and Ramakrishnan [3] gave a parallel solution to the problem. Their algorithm, which runs in $O(\lg n)$ time using n^2 CREW PRAMs, precomputes all maximum weight edges on paths between any pair of nodes in the tree, and then breaks the cycles simultaneously in constant time. Varman and Doshi [4] presented an efficient solution that works in the same parallel time, but uses n CREW PRAM processors. They use divide-and-conquer to split the problem into approximately $\sqrt{n}$ equally-sized subproblems which they solve recursively. Even though their idea is rather simple, the implementation details make the algorithm rather complex. More recently, Jung and Mehlhorn [5] have given an optimal solution for the more powerful CRCW PRAM model by reduction to an all sub-expression evaluation problem. They use an optimal tree contraction algorithm as a subroutine, as do we. However, their approach to breaking cycles is different from ours. In our case, we are able to restrict consideration of cycles to those with no more than four vertices and, as we will show, they can be treated without concurrent writing. Also, because of the all sub-expression evaluation reduction, they need to have an upwards and a downwards pass on the tree to complete the calculation; our approach simply computes an MST in the upward pass.

There is, of course, an obvious algorithm for solving the problem: compute from scratch an MST of the graph having as edges the old MST edges plus the added edges of z. This, however, requires $O(\log n \log\log n)$ time using $n+m$ EREW PRAM processors [7, 8] employing very elaborate techniques. The solution we present here is faster and significantly simpler.

The paper is organized as follows: The remainder of this section discusses a useful input representation. Section 2 has an outline of the solution and describes the rules and the invariant used. Section 3 presents the main theorem and some of the algorithms that can be derived using the rules. Finally, Section 4 shows how the vertex updating algorithms can be used to solve the multiple vertex updating problem in parallel. An earlier version of this paper was presented in [9]. However, the present version differs considerably and is simpler than the previous one.

Representation. As is well known, any MST T of a given graph G can be found by a sequence of deletions of an edge of maximum weight (MaxWE) from some cycle. Since the same sequence of deletions can be followed on the graph augmented with the new vertex z, there is an MST in the augmented graph in which none of these original non-tree edges appears. Therefore, it is sufficient to pose the MST problem in the augmented graph on a graph composed of the original MST and the edges to the new vertex z. This graph, which we call the *sufficient graph*, has at most $2n-1$ edges. We choose to represent the sufficient graph as a tree T with $n-1$ weighted edges (corresponding to the given MST) and with weights on each of its n nodes (corresponding to weights of the newly introduced edges to z). We will call this object a *weighted tree* (Figure 1). A path between any two weighted

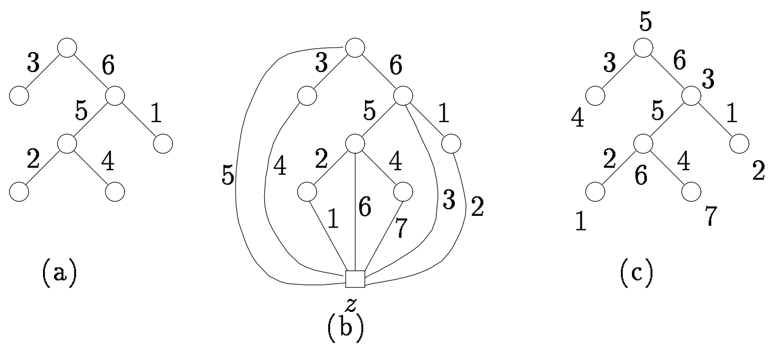

Figure 1: (a) The initial (given) MST, (b) the sufficient graph after introducing the new vertex z along with its weighted edges and (c) the corresponding weighted tree.

nodes in the weighted tree corresponds to a cycle in the graph augmented with z. Such a graph with $2n-1$ edges is shown in Figure 1b and is implied by the weighted tree shown in Figure 1c. In the discussion that follows, reference to the weight of a node will mean reference to the corresponding edge in the sufficient graph, unless noted otherwise.

2 Breaking the Cycles

Outline of the Algorithm. As we mentioned, we are given the input in the form of a weighted rooted tree. We assume that each vertex has a pointer to a circular linked list of its children, and the linked lists are stored in an array. The representation of the input is not crucial, since it can be derived in $O(\lg n)$ time using $n/\lg n$ processors from any reasonable representation.

We should mention at the outset that in the course of our description we treat every case where read or write conflicts might be expected to occur and we show in each case how these conflicts are avoided. The basic idea is that when only a constant number of vertices are involved in a computation, avoiding conflicts is always possible with a constant dilation in running time and is in fact straightforward to implement.

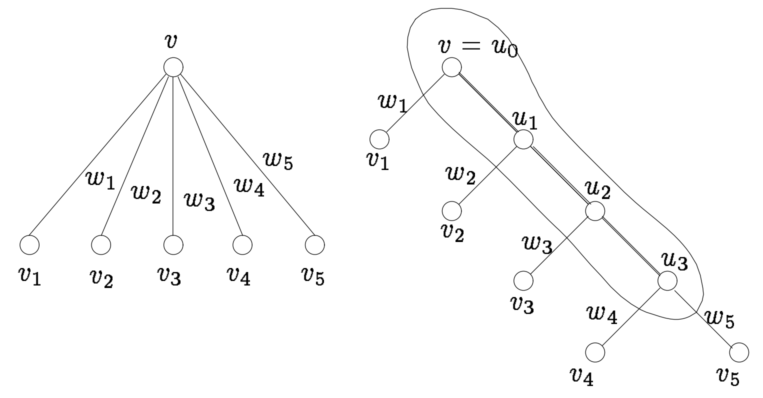

Figure 2: Binarization: A node with more than two children is represented by a right path of unremovable edges.

The algorithm consists of a number of phases. During each phase, leaves and nodes of degree 2 of the weighted tree (such as the root or internal nodes having one child) are *processed*. Each tree-node is processed once in the entire course of the algorithm. The order in which the nodes are processed in parallel is dictated by a tree-contraction schedule. Processing a node means examining the edges composing *small cycles* (cycles of length 3 or 4) that contain the node, and breaking these cycles by removing a MaxWE that appears in them, effectively computing an MST of the subgraph induced by the examined edges. This is done by a set of rules which also update neighboring nodes, so that the size of the unprocessed part of the tree decreases.

A sequential algorithm needs only to apply the appropriate rule while visiting the nodes of the tree. Thus a depth-first-search (or a breadth-first-search) visit of the nodes suffices. When working in parallel though, the rules can apply to many nodes at once, provided that no confusion arises from the simultaneous updating of neighboring nodes. A valid tree-contraction schedule (like the ones presented in Section 3) suffices to assure that neighboring nodes are not processed at the same time. After processing all the nodes, an MST of the sufficient graph has been computed and the algorithm terminates.

Binarization. The rules we present assume a binary tree as input, so some preprocessing is needed to transform the weighted tree to a *binary weighted tree*. This can be achieved in $O(\lg n)$ time using $n/\lg n$ EREW PRAM processors, and is needed only for ordering the processing of a node's children. Note that only the parallel algorithms need this transformation. This transformation is performed by the procedure *binarize*, which we describe here. Each node v having k children $v_1, \cdots, v_k$ is represented by a *right path* (Figure 2) composed of $v = u_0$ and $k-2$ fake nodes $u_1, \cdots, u_{k-2}$, so that node u_j is the right child of u_{j-1} for $j = 1, \ldots, k-2$. Node v_i, for $i = 1, \cdots, k-1$ becomes the left child of node u_{i-1}, and v_k the right child of u_{k-2}. We assign weights of $-\infty$ to the edges of the right path (which makes them unremovable by our rules). Since the fake nodes are introduced only to fix the processing order of v's children, they are assigned weight $+\infty$ making it impossible to retain them in the MST connecting z and v. Of course, the real nodes v and the v_i's keep their weights. At the end of the algorithm each right path is always included in the MST of the binarized problem, giving a unique obvious solution to the original problem.

Some tree-contraction schedules require a regular binary tree as input. For these cases, the binarization is extended to handle nodes v with only one child in the input tree. For each of them a second child v' is introduced, for which $w[v'] = +\infty$ and $w[(v, v')] = -\infty$.

The binary weighted tree has the same number of cycles, but may have height much greater than the input tree and may contain twice as many nodes. This fact, though, does not affect the asymptotic running time of the algorithm, which is logarithmic in the number of tree nodes.

Invariant and Rules. The rules are divided into two categories: Rules that are applied to leaves (pruning rules), and rules that are applied to nodes with only one child (shortcutting rules). Each node is examined exactly once for rule application. Then, its incident edges are identified as being either *included* in the new MST, or *excluded* from it.

We will make use of the following two well-known facts which we state here without proof:

Fact 1 *The edge with minimum weight incident to some node will always be included in the MST.*

□

Actually, Prim's and other sequential [1] and parallel [10] algorithms are based on this fact. Edge inclusion in our rules makes use of this observation.

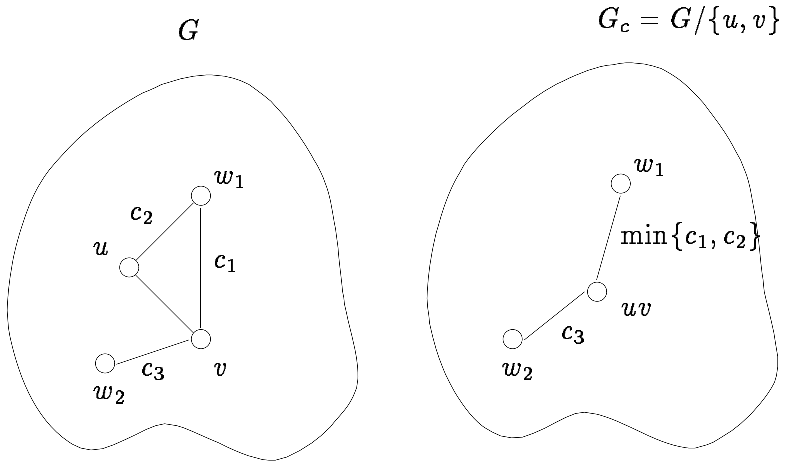

Figure 3: Contraction of edge $\{u, v\}$ in G produces $G_c = G/\{u, v\}$.

Fact 2 *Whenever some edge is found to correspond to the MaxWE of some cycle it can be removed from the graph without affecting the computation of the MST of the graph.*

$\square$

Kruskal's MST algorithm makes use of this fact. Edge exclusion in our rules is based on this observation.

A useful Lemma. Let $w : V \cup E \rightarrow R$ give the weights of the nodes and the edges. At the beginning of the algorithm, the weight of a node v is the weight of the edge connecting v to z. To resolve ties, we adopt the convention that the currently processed node has weight larger than its equally weighted neighbors.

For the purposes of this lemma, we assume that the edges of G can be assigned labels based on their endpoints, *e.g.* edge $\{u, v\}$ receives the label $lab\{u, v\}$, and that these labels persist under the change of endvertices in the contraction operation we now define. Given a weighted graph G and an edge $\{u, v\}$, we define the *contracted graph* $G/\{u, v\}$ as follows: Delete edge u, v and then replace vertices u and v with a new vertex uv, retaining the original edges and edge labels with only the indicated endpoint renamed.

Thus, any edge incident on vertiex u, for example, is now incident on vertex uv. (Figure 3.) In the case where there originally was an edge from a vertex w to both u and v, delete the now parallel edge with the greater weight (or, in the case of a tie, one of the parallel edges).

Lemma 1 *For $\{u, v\} \in MST(G)$,*

$$MST(G) = \{u, v\} \cup MST(G/\{u, v\})$$

where the MST is defined by its edges, and equality is over edge labels that persist under the contraction operation $G/\{u, v\}$.

Proof. Let us denote the contracted graph $G/\{u, v\}$ with G_c. The lemma states that $MST(G) = \{u, v\} \cup MST(G_c)$.

Consider an $MST(G_c)$. Then, for each edge $e \in G_c - MST(G_c)$ there is a cycle composed of edges in G_c, among which e is the MaxWE.

Observe that from the way G_c is constructed, each edge in G_c has a corresponding edge in G. (The opposite is not true.) Now consider a subgraph $S \subseteq G$ induced by $\{u, v\}$ and the edges of G corresponding to $MST(G_c)$. Note that S is a spanning tree of G. We will show that for each edge $e' \in G - S$ there is a cycle in G, in which e' is the MaxWE; this will prove the lemma.

To see that we consider two cases:

1. e' corresponds to edge e in G_c. Since $e \notin MST(G_c)$, there is a cycle in G_c in which e is the MaxWE. Each one of the edges in this cycle corresponds to an edge in G, and e' is the MaxWE in this cycle.

2. e' does not correspond to an edge in G_c. Then e' was one of the edges deleted by the contraction, therefore it has one of its endpoints at u or v. Without loss of generality we assume that $e' = \{w, u\}$. Then, there is another edge $\{w, v\} \in G$ with weight less than the weight of e' which has a corresponding edge in G_c, namely $\{w, uv\}$. Thus, there is a cycle involving edges e', $\{w, v\}$, $\{u, v\}$, in which e' is the MaxWE.

$\square$

We are ready now to present our rules. Each application of a rule will preserve the following invariant.

Invariant *Whenever the rules include an edge e, then there is some $MST(G)$ containing all edges already included for which $e \in MST(G)$. Whenever the*

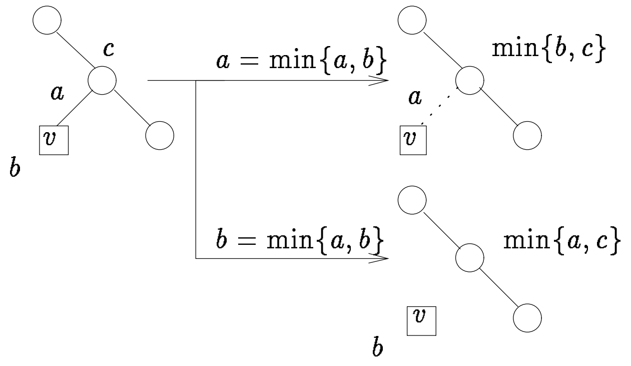

Figure 4: Pruning Rules for a Leaf. An included edge is dotted and we keep its weight letter. We erase an excluded edge along with its weight letter.

rules exclude an edge e', then there is some $MST(G)$ containing all edges already included and $e' \notin MST(G)$. Updates in the weights of the vertices reflect the effect of contracting the included edge.

2.1 Pruning Rules

Consider a small cycle involving leaf v, $p(v)$, the parent of v in the weighted tree of the sufficient graph, and z. Let $w[(v, p(v))] = a$, $w[v] = b$ and $w[p(v)] = c$ (Figure 4). The cycle of length 3 they form can be broken in such a way that the invariant is preserved. We consider the following cases:

$a = \min\{a, b\}$: Then, a should be included and the edge corresponding to $\max\{b, c\}$ should be excluded. We update $w[p(v)] \leftarrow \min\{b, c\}$.

$b = \min\{a, b\}$: Now, b should be included while, the edge corresponding to $\max\{a, c\}$ should be excluded. Similiarly with the previous case, we update $w[p(v)] \leftarrow \min\{a, c\}$.

Some special care to avoid concurrent writing must be taken in the case that v's sibling, say u, is also a leaf (Figure 8) and is processed at the same time. Actually, the ACD scheduling method [11] which we describe in a later section schedules v and u to be processed at the same time. Section 3.2 discusses how we avoid concurrent accesses.

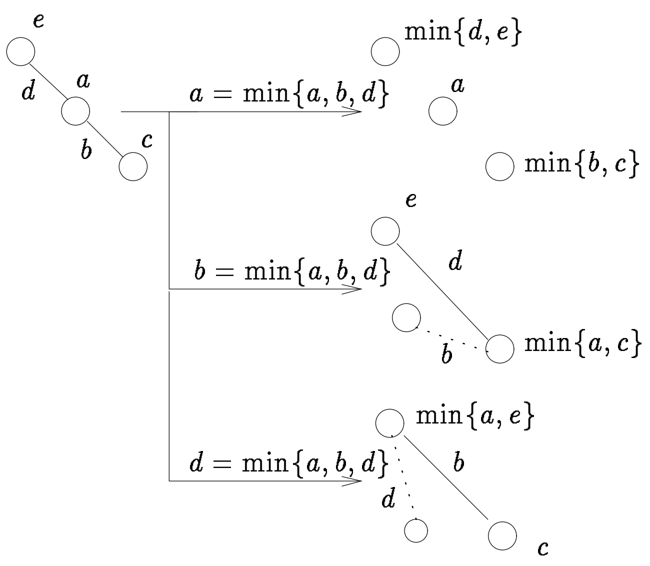

Figure 5: Shortcutting Rules.

2.2 Shortcutting Rules

Now we consider a situation where v has only one child, say u. There are two possible small cycles involving v: z, v, u, z and $z, v, p(v), z$. We will describe how to break these cycles in a way that the invariant is preserved. Let $w[v] = a$, $w[(v,u)] = b$, $w[u] = c$, $w[(v,p(v))] = d$ and $w[p(v)] = e$ (Figure 5).

$a = \min\{a,b,d\}$: Then, a should be included, and the edges corresponding to $\max\{e,d\}$ and $\max\{c,b\}$ should be excluded. We update $w[p(v)] \leftarrow \min\{e,d\}$ and $w[u] \leftarrow \min\{c,b\}$.

$d = \min\{a,b,d\}$: Then, d should included and the edge corresponding to $\max\{a,e\}$ should be excluded. We update $w[p(v)] \leftarrow \min\{a,e\}$.

$b = \min\{a,b,d\}$: Similiarly, b should be included and the edge corresponding to $\max\{a,c\}$ should be excluded. We update $w[u] \leftarrow \min\{a,c\}$.

The fact that shortcutting a node may update both parent and child nodes creates a possibility of a write conflict. This can be easily avoided, since at most three processors may try to access the same memory cell simultaneously. Section 3.2 describes how this conflict is avoided.

Updating weights should be implemented in such a way that the original edge, whose weight appears after the update on the node, is remembered. One pointer per node pointing to this original edge suffices to accomplice this.

Correctness Lemma. We have described a set of rules that define a *prune* operation which removes the leaves of a tree, and a *shortcut* operation which removes nodes of degree two from the tree. Note that individual prunings and shortcuttings take $O(1)$ time to be performed.

We are now ready to prove the following:

Lemma 2 *Application of a pruning or a shortcutting rule on some node v of the weighted graph preserves the invariant.*

Proof. Without loss of generality we may assume that G has a unique MST. This can be easily accompliced by assigning unique weights on the edges of the sufficient graph. The invariant states three things:

1. Whenever the rules include an edge e, it is because e is the minimum weight edge incident to a node, and by Fact 1, $e \in MST(G)$.

2. Whenever the rules exclude an edge e', it is because there is a small cycle in which e' is the MaxWE, and by Fact 2, $e' \notin MST(G)$.

3. In every application of a rule, if we were to contract the included edge, then we would have to introduce an edge with weight equal to the updated weight.

□

We define a *valid tree-contraction schedule* to be one which schedules the nodes of the binary tree for pruning and shortcutting in such a way that (i) when a node is operated upon it has degree one or two, and (ii) neighboring nodes are not operated upon simultaneously.

Lemma 3 *When the rules are applied on the nodes of a binary weighted tree at times given by any valid tree-contraction scheduling, they correctly produce the updated minimum spanning tree.*

Proof. As we said, a valid contraction schedule defines processing times on nodes having degree 1 or 2. So, only the prune and shortcut operations are needed, and they are provided by the rules. Moreover any node may be accessed simultaneously by at most three processors. This conflict can be resolved as described in Section 3.2. Therefore the shortcutting and pruning operations can be done without confusion, and their application, according to Lemmas 1 and 2, preserves the invariant. Thus, at the end of the schedule, the MST of the sufficient graph, has been computed. □

3 The Algorithms

We say that a *sequential* algorithm for some problem of size n is *optimal* if it runs in time that matches a lower bound for the problem to within a constant factor.

Let $t(n)$ denote the parallel running time for some parallel algorithm, and $p(n)$ denote the number of processors employed by the algorithm. Then, $w(n) = t(n)p(n)$ denotes the work performed by the algorithm. A parallel algorithm for some problem is said to be *optimal* [6] if it has polylogarithmic parallel running time and the work $w(n)$ performed by the algorithm is $O(T(n))$, where $T(n)$ is the running time of the best sequential algorithm for the same problem.

The vertex updating problem has a lower bound of $\Omega(n)$ sequential time. Also, any parallel algorithm for the problem must have running time of $\Omega(\log n)$. To see that, consider the following argument: Consider a tree all of whose nodes are on a path of length $n-1$, and therefore having only two leaf-nodes. If we add two new weighted edges connecting z to each of the two leaves, the updating MST problem is equivalent to the problem of computing the maximum of $n+1$ weights on the cycle created. This problem has a sequential lower bound of $\Omega(n)$ and a parallel lower bound of $\Omega(\log n)$ [18].

We next present the algorithms.

Theorem 1 *There are optimal parallel and sequential algorithms that solve the MST vertex updating problem based on the rules presented above. The parallel algorithms run on a binary weighted tree in $O(\log n)$ time using $n/\lg n$ EREW PRAM processors and the sequential algorithms run in $O(n)$.*

Proof. A. THE OPTIMAL PARALLEL ALGORITHMS. There are, actually, several valid tree contraction schedules that produce optimal behavior in our algorithm. First, [12] proposed such a schedule which was constructed on the fly by an optimal randomized algorithm. The problem and its applications drew the attention of researchers, and soon several optimal deterministic algorithms were presented [13, 14, 11, 15, 16].

Shunting. We will briefly describe here the simplest of these schedules (called *Shunting*) which was proposed independently by [13] and [14]. The algorithm is composed of a number of phases, each containing the following steps: (Figure 6)

1. Number the leaves of the tree from left-to-right. Here, the input is supposed to be a *regular* binary tree, i.e. a binary tree in which every internal node has exactly two children. The numbering can be done within the desired bounds using the eulerian tour technique [17].

2. Prune the odd-numbered leaves that are the left children of their parent. Then, shortcut their parent. This is the *shunt* operation.

3. Shunt the odd-numbered leaves that are the right children of their parent.

4. Shift out the last bit of the numbers of the remaining leaves and repeat steps 2 to 4 until the whole tree has been contracted.

Lemma 4 ([13]) *The shunting algorithm computes a valid contraction schedule which has length $O(\lg n)$.*

If we had a processor assigned to each node, we could contract the tree using n processors in the desired time. But the time at which processing occurs can be computed beforehand for each leaf and placed in an array of length $\lceil n/2 \rceil + 1$. The array is filled with pointers to leaves having numbers 1,3,5,7,..., then to leaves having numbers 2,6,10,14,..., etc. In general there are $O(\lg n)$ phases numbered $0 \leq i \leq \lceil \lg(n-2) \rceil$, and in each of them, leaves numbered $2^i, 3 \cdot 2^i, 5 \cdot 2^i, 7 \cdot 2^i, \ldots$ are shunted. Having the array, it takes time $O(n/p)$ using $p \leq n/\lg n$ processors to do the contraction. Optimality is achieved for $p = n/\lg n$. (Example of the algorithm using the shunting schedule is shown in Figure 7.)

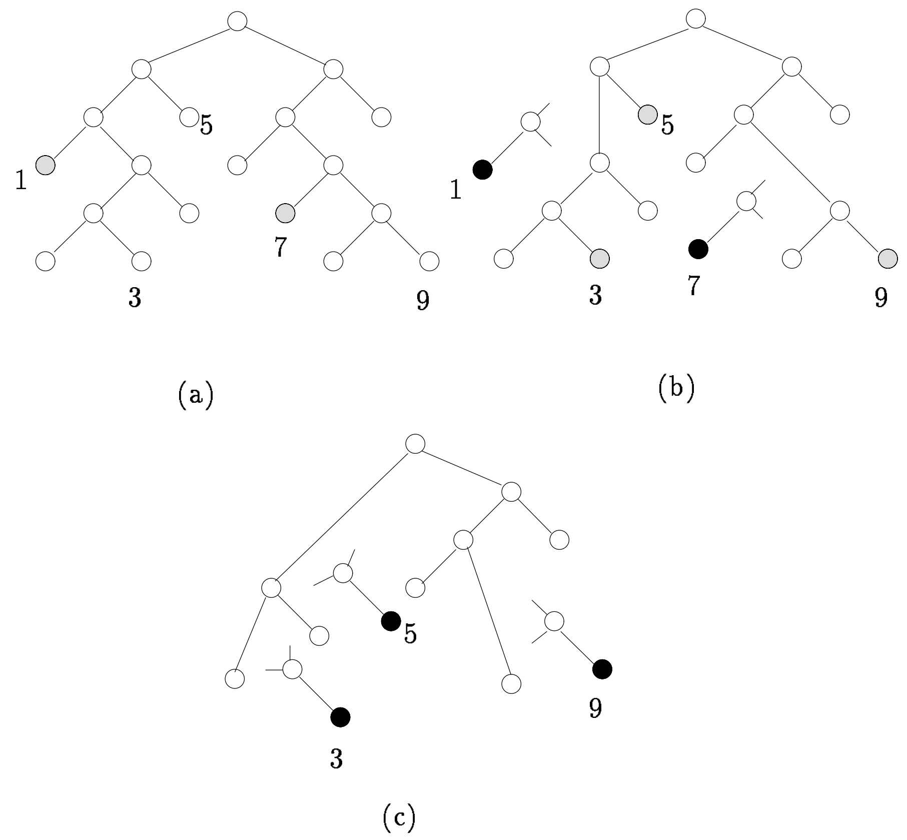

Figure 6: The Shunting Schedule: The first phase. (a) Numbering of the leaves. (b) Step 2: Shunting of odd numbered left children. (c) Step 3: Shunting of odd numbered right children.

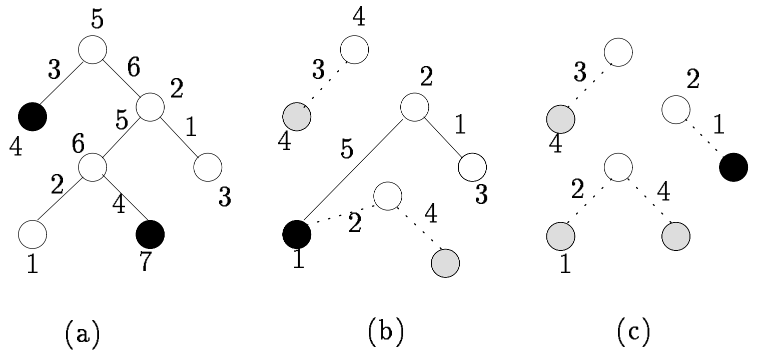

Figure 7: Run of the Parallel Algorithm using SHUNTING on the tree of Figure 1. Dotted edges are included in the MST, deleted edges are excluded from it. (a) In black are the first two processed vertices. (b) Third step. (c) Fourth and final step.

ACD. The accelerated centroid decomposition (ACD) technique was proposed by [11] and also provides an optimal valid scheduling. Using their technique another optimal algorithm for the vertex updating problem is acquired. To conserve space we will not describe this method here; we refer the interested reader to [11].

B. THE OPTIMAL SEQUENTIAL ALGORITHMS. The rules we present do not depend on the particular order in which the nodes of the tree are removed. Therefore, different removal sequences of the nodes yield different algorithms and we can derive sequential algorithms from the parallel ones. In particular, the Shunting and the ACD numberings give two such algorithms. Their running times differ only by a constant. In the next paragraphs we present two more sequential algorithms.

Remove on the fly. Use depth-first-search (or breadth-first-search to visit the nodes of the tree. Every time a node of degree 1 or 2 is encountered, process it using pruning or shortcutting rule, respectively. Each node will be visited at most twice (on the way down the tree and on the way up the tree), so its running time is $O(n)$.

Postorder. Probably the simplest to implement sequential algorithm is the following: Visit the nodes of the weighted tree in postorder (using, for example, depth-first-search). A node is processed after all its children have been processed so, in this case, only the pruning rules are needed. Since each node is processed at most once, we have an $O(n)$ sequential algorithm.

□

Remarks 1. Several researchers who have given solutions to other tree contraction problems, have used a variety of names to denote the "removal of a leaf" and "removal of a node with degree two" operations. *Rake* has been used as a synonym for prune, *compress* and *by-pass* as synonyms for shortcut. Finally, *shunt* and *rake* have been used to denote the application of a prune followed by a shortcut.

2. The algorithms presented do not exhaust the possible sequential and parallel algorithms that can be derived based on the rules, but include only the simpler ones. Other tree-contraction techniques [15, 16] lead to different algorithms with the same bounds.

3. The shunting method described in [13] requires that the root of the tree not be shunted until the end. This is needed mainly for the expression tree evaluation algorithm. In our case, shunting the root is permitted since there is no top-to-bottom information to be preserved.

3.1 Binarization

Theorem 2 *There are logarithmic-time optimal parallel algorithms solving the MST vertex updating problem on a rooted tree.*

Proof: The binarized graph (described in Section 2) has exactly the same number of cycles as the given graph, and at the end of the algorithm the edges composing the right path are always included into the new MST. Therefore, the solution of the binarized problem shows a corresponding unique and unambiguous solution to the general problem. □

Similar binarization techniques to the one described have been used in [4], [11] and [13]. Another technique [5] "plants" a balanced binary tree over the v_i's with v as the root. The internal nodes and the internal edges have weights as those in the right path in the previously described technique. Both constructions require the *list ranking* algorithm [19, 20] which runs within the desired bounds. (Actually, in [4] the Eulerian tour technique is used which has the list ranking procedure as a subroutine.)

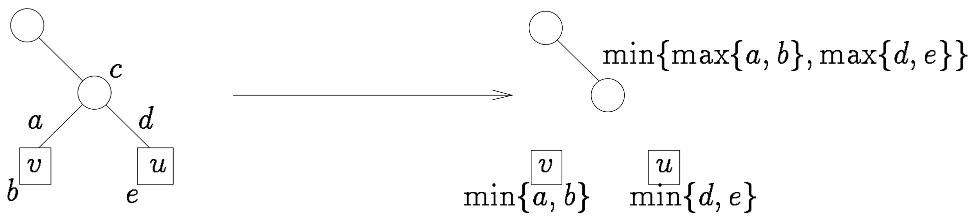

Figure 8: When v's sibling is a leaf.

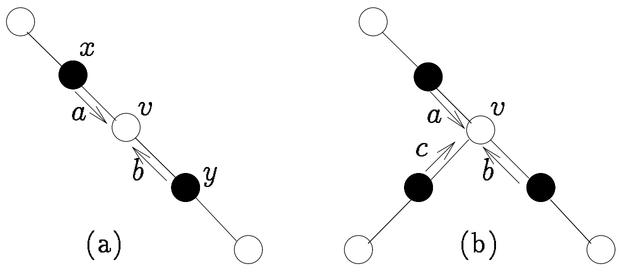

Figure 9: Memory Access Conflicts and how to resolve them. Two (a) and three (b) processors attempt to update v.

3.2 Memory Access Conflicts.

We first describe how to avoid concurrent writing when prunning simultaneously two leaves (Figure 8). Let $w[u] = e$ and $w[(u, p(v))] = d$. Prunning both v and u will result in an attempt by the processors assigned to them to update $w[p(v)]$ at the same time. The correct execution of the algorithm requires that a cycle of length 4 (namely $z, v, p(v), u, z$) be broken by excluding its MaxWE. So $w[p(v)]$ must be updated to either $\max\{a, b\}$ or $\max\{d, e\}$, depending on which child was connected to the excluded edge. This is done by the processor assigned to, say, u as follows: If $\max\{a, b\} > \max\{d, e\}$ then $\max\{a, b\}$ is excluded and $w[p(v)] \leftarrow \max\{d, e\}$. Else (if $\max\{d, e\} > \max\{a, b\}$) $\max\{d, e\}$ is excluded and $w[p(v)] \leftarrow \max\{a, b\}$.

This is the only conflict that can occur during prunnings. The fact that shortcutting may update both parent and child nodes creates a possibility of a write conflict. Consider the following situation: Let nodes x, v, y satisfy $p(y) = v$ and $p(v) = x$, and assume that nodes x and y are to be processed simultaneously (Figure 9a). Moreover, let's assume that shortcutting node x calls for updating child node v with $w[v] \leftarrow a$, and shortcutting node y calls for updating parent node v with $w[v] \leftarrow b$.

The write conflict actually represents a cycle involving nodes z, x, v, y, and possibly x's parent and/or y's child. Since two processors may try to write on the same memory cell, the conflict can be avoided and the cycle behind it should be broken by removing the edge $\max\{a, b\}$ while updating $w[v] \leftarrow \min\{a, b\}$.

This is the only kind of access conflict that can be created by the shunting schedule we described. There are other schedules though, which create a slightly more complicated conflict when updating a node with three values (Figure 9b), say a, b, and c. Again, this can be resolved by breaking the MaxWEs of the three cycles behind the conflict. Therefore, in this case the algorithm should update $w[v] \leftarrow \min\{a, b, c\}$ and should exclude the edges that correspond to the other two values. Note that the write conflict we described in the beginning of this section (when two sibling leaves are pruned simultaneously) can be viewed as a special case of this conflict.

4 On the Multiple Vertex Updates Problem

We define the problem of *multiple vertex updates* of an MST as follows: Let $G = (V_G, E_G)$ be a weighted graph on n vertices and m weighted edges and $T = (V_G, E_T)$ be its MST. Suppose G is augmented with $k = |V_k|$ new vertices that are connected to V_G's vertices by $kn = |E_k|$ new weighted edges, but they are not connected among themselves. We are asked to compute the new MST T'.

The problem of multiple vertex updates was considered by Pawagi [21] and a parallel algorithm was presented running in $O(\lg n \lg k)$ time using nk CREW PRAM processors. The problem was also addressed in [22] and [23].

We will show how our solution for the (single) vertex update problem can be used to achieve a better solution for the multiple updates problem on a weaker model of parallel computation. Note that this is optimal for graphs having $\Omega(kn)$ edges.

Theorem 3 *The multiple updates MST problem can be solved in parallel in time $O(\lg n \lg k)$ using $nk/(\lg n \lg k)$ EREW PRAM processors.*

Proof. The algorithm follows in general the one presented in [21] but in certain parts uses different implementation techniques to achieve the tighter time and processor bounds. The algorithm consists essentially of three parts.

1. Make k copies of T and solve k update MST problems in parallel.
2. Combine the MSTs of the k solutions into a new graph G_z. This graph may contain cycles. Transform it to an equivalent bipartite graph G_b.
3. Solve the bipartite MST problem on the graph G_b.

We will show that each of these parts can be implemented within the desired bounds.

1. Solving k Updating Problems. Making k copies of T requires $O(kn)$ operations and it can be done in constant time if kn processors are available. Therefore, it can be done in $O(\lg n \lg k)$ time using $kn/(\lg n \lg k)$ processors following Brent's technique [6].

According to Theorem 1, a single updating of an MST can be done in $O(\lg n)$ time when $n/\lg n$ processors are available. Here we have k problems to solve, each of size n. Allocating $n/(\lg n \lg k)$ processors per problem, it takes $O(\lg n \lg k)$ time to solve each one in parallel.

2. Creating the Bipartite Graph. Next, we have to combine the k solutions found in the first part, into a new graph G_z which in turn is transformed to an equivalent bipartite graph G_b. By *equivalent* here, we mean that there is a cycle in G_b if and only if there is a cycle in G_z. Graph G_z will never be explicitly created; it is only defined for the sake of description.

If some edge $(v, w) \in E_T$ did not appear in at least one of the k solutions, it was the MaxWE of some cycle and thus must be excluded. So, $G_z = (V \cup \{z_1, \cdots, z_k\}, E_z)$ is composed of edges $(v, w) \in E_T$ that appear in *all* k solutions, along with edges of the form $(z_i, v) \in E_k$, $\forall i \in \{1, \cdots, k\}$. It is easy to see that the formation of G_z can be done within the desired bounds, because there are at most $n - 1$ such edges per solution to examine.

We can view G_z as a collection of subtrees C_j of the old MST that are held together by the z_i's (Figure 10). Every cycle in G_z can be viewed as starting at some z_i, then entering subtree C_j at a node v_e and visiting some of

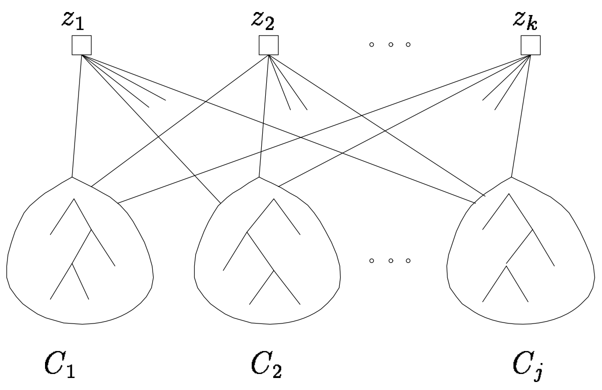

Figure 10: The graph G_z results from putting together the k solutions. The figure points out G_z's bipartite nature.

its nodes, then exiting through a node v'_e and visiting z_l, *etc.*, until returning back to z_i (Figure 11). The nodes v_e that are adjacent to some z_i are called *e-nodes.* The new graph $G_b = (V_b, \{z_1, \cdots z_k\}, E_b)$ has a set of vertices V_b which contains one vertex v for each e-node v_e of G_z. Consider a path from z_i to some e-node v_e and let x be the MaxWE on this path. Then edge $(z_i, v) \in E_b$ corresponds to this path and has cost equal to x's cost. The algorithm needs therefore to compute all MaxWE on all paths from z_i to e-nodes v_e.

Note that solving the MST on G_b effectively solves the MST problem on G_z: There is a one-to-one correspondence between cycles in G_b and G_z. Each cycle in G_b is broken by identifying and deleting the MaxWE in it. Such an edge is also the MaxWE in the corresponding cycle in G_z and should be deleted to compute the MST of G_z.

This is done as follows. First, all e-nodes v_e are recognized. Then we root each of the T_i's at z_i. Both of these operations can be done within the desired bounds. Now we have to compute the MaxWEs on the paths between z_i's and e-nodes by solving k instances of the following problem: Given a (regular binary rooted) tree with n nodes having weights associated with its edges, find the MaxWE for each path from a node to the root in $O(\lg n + n/p)$ time using $p \leq n/\lg n$ EREW PRAM processors. Each solution is a

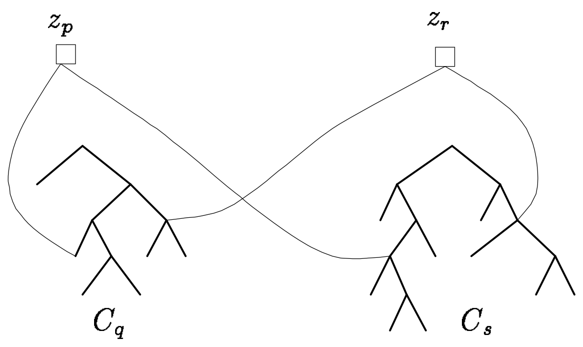

Figure 11: Cycles in G_z. They are composed of alternative visits to z_i's and to tree components. Vertices that are connected to z_i's are called *e-vertices*. In this picture a cycle of length 4 is shown.

simple application of the tree-contraction problem. We allocate $n/\lg n \lg k$ processors per tree and compute the problem in time $O(\lg n \lg k)$.

3. The Bipartite MST Problem. For the third part of the algorithm we need the following definition of the bipartite MST problem: Let $G = (V_k, V_n, E)$ be a weighted bipartite graph, where $|V_k| = k$ and $|V_n| = n$, $k \leq n$. We want to compute its MST. The following Lemma concludes the description of the algorithm along with the proof of Theorem 3.

Lemma 5 *The bipartite MST problem can be solved in $O(\lg n \lg k)$ parallel time using $kn/(\lg n \lg k)$ EREW PRAM processors.*

Proof. The algorithm that we use is a well-known algorithm whose main idea is attributed to Borůvka [24] and was described in its parallel form in [10]. The analysis, however, and the time-processors bounds for the bipartite-MST problem are new.

First, let us give some definitions. A *pseudotree* is a directed graph in which each vertex has outdegree one. A pseudotree has at most one simple cycle. A *pseudoforest* is a graph whose components are pseudotrees. The algorithm consists of a number of stages. In each stage, each vertex v selects the minimum weight edge (v, w) incident to it. This creates a pseudoforest of vertices connected via the selected edges. In this case the cycle of each pseudotree involves only two vertices, so the pseudotree can be easily transformed into a tree. Next, each tree is contracted to a star using pointer-doubling, and vertices in the same component are identified with the root of the star.

The crucial observation here is that, since every pseudotree must contain at least one z_i, after the first stage there will be no more than k vertices in the resulting graph and the problem can be solved in $O(\lg^2 k)$ time using $k^2/\lg^2 k$ processors. So, we only have to show that the first stage can be performed within the desired bounds.

As we said, the first stage consists of finding the minima of $O(n)$ sets of vertices, each with cardinality $O(k)$ and then to reduce the $O(k)$ resulting pseudotrees of height $O(n)$ to stars. For the first part Brent's technique applies. For the second part we use the optimal list-ranking technique of [20]. $\square$

Acknowledgement. The authors would like to thank the anonymous referee for the careful reading of a previous draft, and for pointing out Lemma 1 that simplified the presentation of this paper.

References

[1] P.M. Spira and A. Pan. On finding and updating spanning trees and shortest paths. *SIAM Journal on Computing*, 4(3):375–380, September 1975.

[2] F. Chin and D. Houck. Algorithms for updating minimal spanning trees. *Journal of Computer and System Sciences*, 16(12):333–344, 1978.

[3] S. Pawagi and I.V. Ramakrishnan. An $O(\log n)$ algorithm for parallel update of minimum spanning trees. *Information Processing Letters*, 22(5):223–229, April, 28 1986.

[4] P. Varman and K. Doshi. A parallel vertex insertion algorithm for minimum spanning trees. In *13th ICALP, Lecture Notes*, volume 226, pages 424–433, 1986.

[5] H. Jung and K. Mehlhorn. Parallel algorithms for computing maximal independent sets in trees and for updating minimum spanning trees. *Information Processing Letters*, 27(5):227–236, April, 28 1988.

[6] R. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. *Handbook for Theoretical Computer Science*, 1:869–941, 1990.

[7] D.B. Johnson and P. Metaxas. A parallel algorithm for computing minimum spanning trees. In *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '92)*, pages 363–372, June 1992.

[8] K.W. Chong and T.W. Lam. Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM. In *Proc. of the 4th ACM-SIAM Symposium on Discrete Algorithms*, January 1993.

[9] D.B. Johnson and P. Metaxas. Optimal algorithms for the vertex updating problem of a minimum spanning tree. In *Proc. of the 6th Intl Parallel Proccessing Symposium (IPPS '92)*, pages 306–314, March 1992.

[10] F.Y. Chin, J. Lam, and I-N. Chen. Efficient parallel algorithms for some graph problems. *Communications of ACM*, 25(9):659–665, September 1982.

[11] R. Cole and U. Vishkin. The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time. *Algorithmica*, 3:329–346, 1988.

[12] G. L. Miller and J. H. Reif. Parallel tree contraction. Part 1: Fundamentals. *Advances in Computing Research*, 5:47–72, 1989. Part in 26th FOCS 1985.

[13] K. Abrahamson, N. Dadoun, D.G. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

[14] S. Rao Kosaraju and Arthur L. Delcher. Optimal parallel evaluation of tree-structured computations by raking. *Aegean Workshop on Computing*, pages 101–110, 1988.

[15] A. Gibbons and W. Rytter. An optimal parallel algorithm for dynamic expression evaluation and its applications. In *Proc. of Symposium on Foundations of Software Technology and Theoretical Computer Science*, volume 6, pages 453–469. Springer Verlag, 1986.

[16] H. Gazit, G. L. Miller, and S. H. Teng. Optimal tree contraction in the EREW model. In *Concurrent Computations: Algorithms, Architecture, and Technology*, New York, 1988. Tewksbury and Dickinson and Schwartz (editors), Plenum Press.

[17] R.E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal of Computing*, 14(4):862–874, 1985.

[18] S. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM Journal of Computing*, 15(1):87–97, February 1986.

[19] R. Cole and U. Vishkin. Approximate parallel scheduling. Part 1: The basic technique with applications to optimal parallel list ranking in logarithmic time. *SIAM Journal of Computing*, 17(1):128–142, February 1988.

[20] R.J. Anderson and G.L. Miller. Deterministic parallel list ranking. *Algorithmica*, 6:859–868, 1991. Also: Proc. 3rd AWOC 1988.

[21] S. Pawagi. A parallel algorithm for multiple updates of minimum spanning trees. In *International Conference on Parallel Processing*, volume III, pages 9–15, 1989.

[22] S. Pawagi and O. Kaser. Optimal parallel algorithms for multiple updates of minimum spanning trees. Technical report, Dept of Comp. Sc., SUNY at Stony Brook, 1991. Unpublished Manuscript.

[23] A. Schäffer and P. Varman. Parallel batch update of minimum spanning trees. Technical report, Dept of Elec. and Comp. Engin., Rice University, May 1991. Unpublished Manuscript.

[24] R.E. Tarjan. *Data Structures and Network algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania 19103, 1983.