A linear-work parallel algorithm for finding minimum spanning trees

to appear in Proceedings of SPAA '94

Richard Cole* New York University

Philip N. Klein† Brown University

Robert E. Tarjan‡ Princeton University and NEC Research Institute

Abstract

We give the first linear-work parallel algorithm for finding a minimum spanning tree. It is a randomized algorithm, and requires $O(2^{\log^* n} \log n)$ expected time. It is a modification of the sequential linear-time algorithm of Klein and Tarjan.

1 Introduction

The degree to which a parallel algorithm makes efficient use of its processors can be measured by its *work*, the number of processors multiplied by the time required. If the work done is within a constant factor of the running time of the best sequential algorithm for the same problem, the parallel algorithm exhibits linear speed-up.

We give the first such parallel algorithm for finding a minimum spanning tree. It is randomized, and runs in $O(2^{\log^* m} \log m)$ expected time and does $O(m)$ expected work on graphs with m edges. It is based on a randomized linear-time algorithm that was recently discovered by Klein and Tarjan [16].

The algorithm of Klein and Tarjan is a modification of one proposed by Karger [12]. It is a recursive algorithm in which two successive recursive calls are made; thus it appeared to be difficult to parallelize. In this paper we show that the algorithm can be modified so that with high probability the depth of recursion is reduced from $O(\log n)$ to $\log^* n$. This modification thus ensures that the size of the recursion tree is at most $2^{\log^* n}$. The sequential algorithm uses a linear-time algorithm due to Dixon, Rauch, and Tarjan for verifying a minimum spanning tree. A parallel version of this algorithm that runs in $O(\log n)$ time was discovered by Dixon and Tarjan [7]. A simpler such algorithm was recently discovered by King [14, 15]. By using one of these algorithms as a subroutine, we obtain an expected time bound of $O(2^{\log^* n} \log n)$ for the concurrent-read concurrent-write PRAM.

To reduce the depth of recursion from logarithmic to $\log^* n$, we use the following idea. Whereas each recursive call of the sequential algorithm executes one iteration of a contraction step (called a "Borůvka" step, after the discoverer of the algorithm that consists of such steps), the number of Borůvka steps executed by the parallel algorithm increases quickly with the recursion depth. In fact, since carrying out so many Borůvka steps would be too slow, the parallel algorithm instead uses an alternative, randomized approach.

1.1 Previous work

We review some of the work on obtaining *work-efficient* PRAM algorithms for the minimum spanning tree problem. Chin, Lam, and Chen gave a parallel algorithm that runs in $O(\log^2 n)$ time using $n^2/\log^2 n$ processors. Thus their algorithm achieves linear speed-up when the input graph is a complete graph. However, it is not very work-efficient for sparse input graphs. Awerbuch and Shiloach [1] proposed a parallel algorithm for finding a minimum spanning tree; their algorithm requires $O(\log n)$ time using $m + n$ processors, where n and m are, respectively, the number of nodes and edges of the graph. Thus their algorithm is within a logarithmic factor of optimal in its use of processors. However, their result assumes a model in which write-conflicts are de-

*This research was supported in part by NSF grants CCR-8906949 and CCR-9202900.

†This research was supported in part by NSF PYI award CCR-9157620, together with PYI matching funds from Thinking Machines Corporation, Xerox Corporation, and Honeywell Corporation. Additional support provided by ARPA contract N00014-91-J-4052 ARPA Order No. 8225.

‡Department of Computer Science, Princeton University, Princeton, NJ and the NEC Research Institute, Princeton, NJ. Research at Princeton University partially supported by the National Science Foundation, Grant No. CCR-8920505; the Office of Naval Research, Contract No. N0014-91-J-1463; and DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science and Technology Center, Grant No. NSF-STC88-09648.

termined by priority, where the priority of a processor is determined by the weight of the edge assigned to it. Our algorithm assumes a weaker model in which arbitrary processors succeed in writing.

Cole and Vishkin [5] have claimed an algorithm running on a CRCW PRAM that requires $O(\log n)$ time and $O((n+m) \log\log\log n / \log n)$ processors. Thus their algorithm is within a triply logarithmic factor of optimal. Their algorithm assumes the same strong model as the algorithm of Awerbuch and Shiloach.

Karger [11] has claimed an algorithm running on an EREW PRAM that requires $O(\log n)$ time and $m/\log n + n^{1+\epsilon}$ processors for any constant $\epsilon > 0$. Thus his algorithm is within a constant factor of optimal for sufficiently dense graphs, and is within a fractional polynomial factor for very sparse graphs.

2 Overview of the algorithm

We start by reviewing the sequential algorithm for finding a minimum spanning forest. We then provide an overview of the parallel algorithm. We assume throughout that edge-weights are distinct; this assumption ensures that the minimum spanning forest is unique. If weights are not distinct, ties can be broken by considering edge-ID's.

2.1 The sequential algorithm

Given an n-node, m-edge graph with no isolated nodes, the algorithm first executes a *Borůvka step*, which contracts at least $n/2$ edges, halving the number of nodes. The Borůvka step ensures that the contracted edges belong to the minimum spanning forest of the given graph. Let us refer to the resulting graph as the *contracted graph*.

If the given graph is sufficiently dense (in particular, if $m/n \geq 5$), the algorithm randomly selects a sample subgraph of the contracted graph by including each edge in the sample independently with probability $p = 1/2$. The algorithm recursively finds a minimum spanning forest F of the sample subgraph. Then the algorithm uses F to discard edges of the contracted graph that could not possibly be in the minimum spanning forest. (This step is discussed in greater detail later.) Finally the algorithm recurs on the subgraph of the contracted graph consisting of edges not discarded.

We now provide greater detail for the discarding step. For any forest F, we say an edge e not in the forest is F-heavy if the endpoints of e are connected by a path in F and every edge on that path has weight less than that of e. An F-heavy edge is not in the minimum spanning forest (see, e.g., [21]), and hence can be discarded. Dixon, Rauch, and Tarjan [6] give a linear-time algorithm that can be used to determine the set of F-heavy edges. King [14] has recently given a simpler linear-time algorithm for this problem. Klein and Tarjan [16] show that the number of edges that are not F-heavy is probably not much more than n/p, where p is the sampling probability (in the algorithm above,$p = 1/2$). It follows that the expected sum of sizes of all graphs being recurred on decreases geometrically with the depth of the recursion. Since each call requires linear time (aside from the time spent in recursive invocations), it follows that the algorithm runs in linear time.

2.2 The parallel algorithm

The central difficulty in obtaining a fast parallel algorithm lies in the recursive structure of the algorithm in the case when random sampling takes place; the second recursive call relies for its input on the output of the first recursive call. This data dependency imposes sequentiality on the process. In order to nevertheless obtain a fast parallel algorithm, we modify the Borůvka step and the sampling probability p so as to reduce the recursion depth from $O(\log m)$ to $\log^* m$. Consequently, the recursion tree has size at most $2^{\log^* m}$, so the sequentiality forces only $2^{\log^* m}$ computations to take place sequentially. We can bound the expected time per computation by $O(\log n)$, thereby obtaining an expected running time of $O(2^{\log^* m} \log n)$.

The key to reducing the recursion depth is ensuring that the sum of sizes of all graphs being recurred on decreases much more quickly than geometrically. Rather than carry out a single Borůvka step, which reduces the number of nodes by a factor of two, we carry out a number of such contraction steps, reducing the number of nodes by a larger factor. The number of steps executed depends on the depth of recursion; the greater the depth, the greater the number of steps. Since at greater recursion depth the sum of sizes of graphs being recurred on is significantly smaller than the initial problem size, the contraction steps are less costly. Hence we can afford to carry out more of them while maintaining the linear bound on the total work done.

Carrying out many Borůvka steps would take too much time, so we use a slightly different approach. In Section 3 we give a procedure $\textsc{Contract}(G, k)$ that, for a graph G, contracts minimum-spanning-forest edges. The procedure is guaranteed to contract enough edges so that the resulting graph has at most $1/k$ times as many nonisolated nodes as G. The procedure takes expected time $O(\log m)$ and work $O(m \log k)$, where m is the number of edges of G.

The other nontrivial step of the minimum-spanning-forest algorithm involves discarding F-heavy edges. The discarding step can be executed in parallel by an algorithm of Dixon and Tarjan [7] that runs in $O(\log n)$ time using $(n+m)/\log n$ processors.King [15] has a parallel algorithm that is relatively simple, especially if it is adapted for use in the algorithm presented in this paper.[1]

Before giving our parallel algorithm, we define some recurrence relations expressing parameters in terms of the recursion depth d. The first parameter $r(d)$ controls the sampling probability The second parameter $s(d)$ reflects the sum of sizes of all graphs being recurred on at depth d. We shall show below that at depth d, the expected sum of sizes of all graphs is at most $m_0/s(d)$, where m_0 is the size of the initial input graph.

$$r(0) = 2^2, \qquad r(d+1) = 2^{r(d)}$$
$$s(0) = 1, \qquad s(d+1) = s(d)\cdot r(d)/4$$

Finally, let $w(d) = 32s(d)2^{-d}$. This parameter controls how much work is to be done by the procedure Contract.

The algorithm MST(G,d) follows. It assumes that G has no isolated nodes. To find a minimum spanning forest of an input graph G_0, one calls MST$(G_0, 0)$. Let m_0 denote the number of edges in the input graph, and let c be a constant to be determined.

MST(G,d)

1. If $|E(G)| < cm_0/\lg m_0 \lg\lg m_0$ then call Contract$(G, |V(G)|)$ and return the edges contracted. Otherwise, continue.
2. Let G' be the result of Contract$(G, 2^{w(d)})$.
3. Let H be a graph obtained from G by sampling edges with probability $p = 1/r(d)$ and then deleting isolated nodes.
4. Recursively call MST$(H, d+1)$, obtaining a forest F.
5. Let G'' be the graph obtained from G' by discarding the F-heavy edges, and then deleting isolated nodes.
6. Recursively call MST$(G'', d+1)$.
7. Return the edges in the tree found by the recursive call in Step 6 together with the edges contracted in Step 2.

The correctness of the algorithm follows from the correctness of the Contract procedure and the fact that F-heavy edges are not in the minimum spanning forest. We proceed to the analysis.

The following weak bound on $s(d)$ can be proved by a simple induction.

$$s(d) \geq 2^{d-1} \tag{1}$$

[1] One step in King's parallel verification algorithm can be omitted because the computation it performs is already done by our algorithm.

Let m_0 be the number of edges in the original input graph. We define the *size* of an invocation MST(G,d) as the number of edges in the graph G to which it is applied. The *children* of an invocation are the recursive invocations that it directly invokes. We say an invocation is *small* if its size is less than $cm_0/\lg m_0 \lg\lg m_0$ and is *large* otherwise. Note that because of Step 1 a small invocation does not result in further recursive invocations. We say a depth-d invocation of size m *fails* if the sum of the sizes of its large children is greater than $4m/r(d)$.

Claim 1 *The probability that a depth-d invocation fails is $e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$.*

Proof: Consider the invocation MST(G,d), and let m be its size. It invokes two children. We prove a bound of $e^{-\Omega(m_0/\lg m_0 \lg\lg m_0)}$ on the probability that the first child is large and has size greater than $1.5m/r(d)$. We prove a bound of $e^{-\Omega(m/r(d)\lg m_0 \lg m_0)}$ on the probability that the second child is large and has size greater than $2.5m/r(d)$. Hence the probability is $e^{-\Omega(m_0/\lg m_0 \lg\lg m_0)} + e^{-\Omega(m/r(d)\lg m_0 \lg m_0)}$ that the sum of sizes of large children exceeds $4m/r(d)$. This latter probability is itself $e^{-\Omega(m/r(d)\lg m_0 \lg\lg m_0)}$.

In Step 3, we obtain H by sampling from G' with probability $p = 1/r(d)$. Let X be the number of edges in H. Since G' has fewer than m edges, X is binomially distributed with mean less than $pm = m/r(d)$. By a Chernoff bound, the probability is $e^{-\Omega(m_0/\lg m_0 \lg\lg m_0)}$ that $X > m_0/\lg^2 m_0$ and $X > 1.5m/r(d)$.

It remains to consider the size of the graph G'' in the recursive call in Step 6. Let n be the number of nodes of G, and let n' be the number of nodes of G'. We first need to prove the inequality

$$2^{w(d)} \geq r(d)^2 \tag{2}$$

The inequality is easy to verify for $d = 0$. For $d > 0$,

$$\begin{aligned}
2^{w(d)} &= 2^{32s(d)2^{-d}} \\
&= 2^{32[s(d-1)\cdot r(d-1)/4]2^{-d}} \\
&= \left(2^{r(d-1)}\right)^{2s(d-1)2^{-(d-2)}} \\
&\geq \left(2^{r(d-1)}\right)^2 \\
&= r(d)^2
\end{aligned}$$

where in the penultimate line we use (1).

The graph G'' on which we recur in Step 6 consists of the edges of G that are F-light. Let p be the sampling probability $1/r(d)$. We use the main lemma of [16] to bound the probability that the number of F-light edges exceeds k, where k is a given integer. Namely, this probability is at most the probability of fewer than n' successes in k trials, where the probability of success is p. Since G has no isolated nodes, $n \leq 2m$. The procedure CONTRACT invoked in Step 2 ensures that $n' \leq n/2^{w(d)}$. Combining these inequalities with (2), we obtain $n' \leq 2m/r(d)^2$. Let $k = \max\{m_0/\lg m_0 \lg\lg m_0, 2.5m/r(d)\}$. The expected number of successes is pk, which is at least $2.5m/r(d)^2$. Hence $n' < \frac{2}{2.5}pk$. We may apply the Chernoff bound

$$\Pr[\text{number of successes} \leq (1-\delta)pk] < e^{-\Omega(\delta^2 pk)}. \quad (3)$$

Hence the probability that the number of F-light edges exceeds k is $e^{-\Omega(pk)}$, which is $e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$. □

Suppose the algorithm is applied to a graph with m_0 edges, and consider all the resulting invocations of the procedure MST. Let m_d be the sum of sizes of all large depth-d invocations. (We account for the sizes of small invocations separately.) We show that m_d decreases very quickly as the depth d increases. Let

$$d^* = \min\{d : r(d-1) \geq \frac{1}{2}\lg m_0 - 2\lg\lg m_0\}. \quad (4)$$

Note that $d^* \leq \lg^* m_0$. Note also that

$$r(d-1) \leq 2^{\frac{1}{2}\lg m_0 - 2\lg\lg m_0} \leq \sqrt{m_0}/\lg^2 m_0.$$

Lemma 1 *For $d \leq d^*$, m_d is at most $m_0/s(d)$ with probability $1 - e^{-\Omega(m_0/r(d-1)\lg m_0 \lg\lg m_0)}$.*

Proof: The proof is by induction on d. The basis $d = 0$ is trivial. For the inductive step, we show below that if (i) $m_d \leq m_0/s(d)$ and (ii) no depth-d invocation fails then $m_{d+1} \leq m_0/s(d+1)$. The probability that condition (i) fails to hold is $e^{-\Omega(m_0/r(d-1)\lg m_0 \lg\lg m_0)}$ by the inductive hypothesis. Given that condition (i) holds, the number of large depth-d invocations is certainly at most m_d, which by the assumption is no more than $m_0/s(d)$. The probability that any of these large depth-d invocations fails is at most $(m_0/s(d))e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$ by Claim 1. This probability, which is the probability that condition (ii) fails to hold, is $e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$ for $d+1 \leq d^*$. Thus the probability that both conditions (i) and (ii) hold is $1 - e^{-\Omega(m_0/r(d-1)\lg m_0 \lg\lg m_0)} - e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$, which is in turn $1 - e^{-\Omega(m_0/r(d)\lg m_0 \lg\lg m_0)}$.

Assume that $m_d \leq m_0/s(d)$ and no depth-d invocation fails. Consider one depth-d invocation $\mathrm{MST}(G,d)$, and suppose its size is m. By the definition of not failing, the sum of sizes of its large children invocations is at most $4m/r(d)$. Summing this bound over all large depth-d invocations, we infer that $m_{d+1} \leq 4m_d/r(d)$. Since $m_d \leq m_0/s(d)$, we obtain $m_{d+1} \leq m_0/s(d+1)$. □

Using the fact that the recursion tree is binary, we obtain the following corollary.

Corollary 1 *With probability $1 - e^{-\Omega(\sqrt{m_0})}$, the recursion depth is at most d^* and there are at most 2^{d^*+1} invocations.*

Proof: It is easy to prove by induction that $s(d+1) = \Omega(r(d)\lg\lg r(d))$. In particular, $r(d^*-1) = \Omega(\lg m_0)$, so $s(d^*) = \Omega(\lg m_0 \lg\lg m_0)$. Thus, by Lemma 1, the probability is $1 - e^{-\Omega(m_0/r(d^*-1)\lg m_0 \lg\lg m_0)} = 1 - e^{-\Omega(\sqrt{m_0})}$ that m_d is $O(m_0/\lg m_0 \lg\lg m_0)$. Let the constant c in Step 1 be the constant in this bound, so if the above likely event takes place, every level-d^* invocation stops in Step 1. Hence in this case there are no further recursions. □

We now compute the expected time required by the algorithm. The time per invocation (not including recursive calls) is dominated by Steps 1, 2, and 5. Step 5 can be carried out in $O(\log m)$ time using the linear-work algorithm of Dixon and Tarjan [7]. The expected time required by the procedure CONTRACT is at most logarithmic in the size of the graph. Hence the total expected time per invocation is $O(\log m)$. Since with high probability there are at most 2^{d^*+1} invocations, the total expected time for the algorithm is $O(2^{\log^* m_0}\log m_0)$.

Next we address the work done by the algorithm. First we bound the work done in Step 1 for small invocations. For a graph with m edges the work required by CONTRACT is $O(m\log m)$. Since in Step 1 the procedure CONTRACT is only called if $m \leq m_0/\lg m_0 \lg\lg m_0$, we can bound the work done in Step 1 by $O(m_0/\lg\lg m_0)$ per invocation. With high probability there are at most 2^{d^*+1} invocations, so the total work done in Step 1 overall is $O(m_0)$.

Next we bound work done by the algorithm during large depth-d invocations (not including that done in the resulting recursive calls). Consider a large depth-d invocation $\mathrm{MST}(G,d)$, and let m be the number of edges of G. For each nonrecursive step of the algorithm except Step 2, the work done is linear in m. The expected work done in Step 2 is $O(mw(d))$. Thus the total expected work done (not including recursive calls or Step 1) is $O(mw(d))$.

Summing over all large depth-d invocations, we infer that the expected work done (not including recursive calls) is $O(m_d w(d))$. To find the total expected work, we sum over d.

$$\sum_d m_d w(d) \quad \leq \quad \sum_d \frac{m_0}{s(d)} w(d)$$

$$= \sum_d 32m_0 2^{-d}$$
$$= O(m_0)$$

Thus the total expected work is linear.

3 The Contract procedure

The procedure Contract(G, k) contracts enough edges so that, in the resulting graph, the number of nonisolated nodes is at most $1/k$ times the number of nodes in G. The algorithm consists of a sequence of iterations. In each iteration, some nodes identify their lowest-weight incident edges, and contract them. We use randomization in both identifying the lowest-weight incident edges, and in ensuring that no cycles are formed by the contraction. It would take too long to fully carry out the contractions in each iteration, so we use an idea of Shiloach and Vishkin ([20]; see also [1]) and represent the sets of merged nodes using *parent pointers*.

A processor is associated with each edge and each node. For each node v, the algorithm maintains a *parent pointer* $p(v)$. The parent pointers form a structure consisting of trees where each root points to itself. A tree of nodes that all have the same parent is called a *star*. Each tree of parent pointers consists of the nodes in a component of the set of edges selected so far. Unlike the algorithm of Shiloach and Vishkin, our algorithm maintains the invariant that, at the beginning of each iteration, every pointer-structure tree is a star. An *eligible star* is one containing a node with an incident edge whose other endpoint belongs to a different star. After the last iteration, each star is coalesced into a single node by contracting edges. The number of nonisolated nodes after contraction is the number of eligible stars. Thus the algorithm need only iterate until the number of eligible stars is guaranteed to be at most $1/k$ times the number of nodes in G.

Contract(G, k)
Let n be the number of nodes in G.
Initialize by setting $p(v) := v$ for each node v.
Repeat $f(k)$ times:

- For each star, randomly choose heads or tails.
- For many stars S,
  - select the cheapest edge uv connecting a node u in S to a node v belonging to a different star, and
  - if S is heads and the other star is tails, set $p(p(u)) := p(v)$, and record uv as a contracted edge.
- For each node v, $p(v) := p(p(v))$.

If the number of eligible stars is more than n/k, go to the repeat-loop. Otherwise, coalesce each star into a single node, and return.

We use a randomized method, outlined in Subsection 3.1, for selecting the cheapest edge incident to an eligible star. This method does not select an edge for every star, but for a sufficiently large proportion of the eligible stars (with high probability), as shown by Lemma 2. Using this lemma and a Chernoff bound, it is not hard to show by induction on the number of iterations that the algorithm maintains the following invariant:

> After i iterations, the number of eligible stars is at most n/c^i with probability $1-e^{-\Omega(n/c^i)}$.

Here c is a constant determined by the analysis.

We choose the number of iterations $f(k) = \log_c k$. This ensures that at the end of the first phase the number of eligible stars remaining is at most n/k with probability $1 - e^{-\Omega(1)}$. If the first phase was successful in reducing the number of eligible stars to at most n/k, the Contract algorithm terminates. Otherwise, it repeats. Since the probability of success is $1-e^{-\Omega(1)}$, the expected number of repetitions is constant.

3.1 The edge-selection method

In this subsection, we outline the method used to determine the cheapest incident edge for many of the eligible stars. Suppose that i iterations have already taken place. The invariant implies that probably the number of eligible stars is at most n/c^i. Let $s = n/c^i$, and let m be the number of edges.

The method consists of four steps. First, many of the stars are allocated blocks of memory of size $c_1 m/s$. (More on this step later.) The entries of the blocks are initialized to infinity. Second, for each star allocated a block of memory, each of the edges incident to that star attempts to write its cost into a randomly chosen entry in that block. Third, Megiddo's parallel algorithm [17] for computing the minimum is applied to the entries of each block. Finally, for each star the incident edges verify that the minimum computed for that star is indeed the minimum of the costs of the incident edges; if not, it is marked as invalid.

In the first step we allocate a total of at most $c_2 s$ blocks of memory. To do the allocation, we must assign distinct integers from 1 to $c_2 s$ to the stars. We start by allocating an auxiliary array of size $c_2 s$. Each star randomly selects an integer between 1 and $c_2 s$, and attempts to write its root's node-number into the corresponding entry of the array. Then each star reads the same entry to determine if it succeeded in writing. Each successful star interprets its randomly selected integer as the index of an allocated block of memory. The unsuccessful stars remain idle for the remainder of this procedure.

The first step can be executed in constant time with one processor per star. The second and fourth step can be executed in constant time with one processor per edge.

Megiddo's algorithm takes constant expected time. We allow it to run for constant time; by choice of this constant, we can ensure that it terminates successfully with probability at least a constant c_3. Thus we execute the third step in constant time with $c_1 m/s$ processors per block; since there are at most $c_2 s$ blocks, the total number of processors required is $O(m)$.

Lemma 2 *Suppose that the number of eligible stars is at most s. Then, with probability at least $1 - e^{-\Omega(s)}$, there are at most $(4/5)s$ eligible stars whose minimum incident edge is not determined.*

Proof: We can assume that the number of stars is more than $(4/5)s$, for otherwise the lemma holds trivially. Say the first step *fails* if more than $s/5$ eligible stars fail to get blocks allocated. Say an eligible star has *low degree* if it has no more than $10m/s$ incident edges. At most $s/5$ eligible stars have high degree (else the number of edges exceeds m, a contradiction). Say a low-degree eligible star that was allocated a block *fails in the second step* if the minimum incident edge attempts to write its cost to the same entry of the block as another incident edge. Say the second step fails if more than $s/5$ stars fail in the second step. Say the third step fails if more than $s/5$ invocations of Megiddo's algorithm terminate unsuccessfully. By choice of constants c_1, c_2, and c_3 one can ensure that the probability of any step failing is $e^{-\Omega(s)}$. If no step fails, at most $(4/5)s$ eligible stars fail to determine their minimum incident edges. □

Remarks

Our linear-work parallel algorithm for finding a minimum spanning forest clearly serves as a linear-work parallel algorithm for computing connected components. Indeed, in this case the verification step becomes trivial, as does the edge-selection method in *Contract*. The resulting randomized connected-components algorithm, while not as fast as the $O(\log n)$-time algorithm of Gazit [9], is conceptually quite simple.

References

[1] B. Awerbuch and Y. Shiloach, "New connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM," *IEEE Transactions on Computers,* C-36 (10), 1987, pp. 1258-1263.

[2] O. Borůvka, "O jistém problému minimálním, *Práca Moravské Přírodovědecké Společnosti* 3, 1926, pp. 37-58. (In Czech.)

[3] F. Y. Chin, J. Lam, and I.-N. Chen, "Efficient parallel algorithms for some graph problems, *CACM 25*, 1982, pp. 659–665.

[4] K. W. Chong and T. W. Lam, "Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM, *Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, 1993, pp. 11–20.

[5] R. Cole and U. Vishkin, "Approximate and exact parallel scheduling with applications to list, tree, and graph problems," *Proc. 27th Annual IEEE Symp. on Foundations of Computer Science*, 1986, pp. 478-491.

[6] B. Dixon, M. Rauch, and R. E. Tarjan, "Verification and sensitivity analysis of minimum spanning trees in linear time," *SIAM J. on Computing* 21, 1992, pp. 1184-1192.

[7] B. Dixon and R. E. Tarjan, "Optimal parallel verification of minimum spanning trees in logarithmic time," preprint, 1993.

[8] M. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1986, pp. 478-491.

[9] H. Gazit, "An optimal randomized parallel algorithm for finding connected components in a graph," *Proc. 27th Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1986, pp. 492-501.

[10] D. B. Johnson and Metaxas, "A parallel algorithm for computing minimum spanning trees, *Proc. 4th Annual ACM Symp. on Parallel Algorithms and Architectures*, 1992, pp. 3630–372.

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

[12] D. R. Karger "Random sampling in matroids, with applications to graph connectivity and minimum spanning trees," *Proc. 34st Annual IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 84-93.

[13] R. M. Karp and V. Ramachandran, "A survey of parallel algorithms for shared-memory machines," Chapter 17 in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity* J. van Leeuwen, ed., MIT Press, Cambridge, Mass., 1990, pp. 869–941.

[14] V. King, " A simpler minimum spanning tree verification algorithm," manuscript, 1993.

[15] V. King, personal communication, 1994.

[16] P. N. Klein and R. E. Tarjan, "A randomized linear-time algorithm for finding minimum spanning trees, to appear in *Proc. 26th Annual ACM Symp. on Theory of Computing*, to appear, 1994.

[17] N. Megiddo, "Parallel algorithms for finding the maximum and the median almost surely in constant time," preprint, 1982.

[18] P. Raghavan, "Lecture Notes on Randomized Algorithms," Research Report RC 15340 (#68237), Computer Science/Mathematics IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, 1990, p. 54.

[19] C. Savage and J. Ja'Ja, "Fast efficient parallel algorithms for some graph problems, *SIAM J. Comput 10* 1981, pp. 682.

[20] Y. Shiloach and U. Vishkin, "An $O(\log n)$ parallel connectivity algorithm," *J. Algorithms 3*, 1982, pp. 57-67.

[21] R. E. Tarjan, *Data Structures and Network Algorithms*, Chapter 6, Society for Industrial and Applied Mathematics, Philadelphia, 1983.