Simple Concurrent Labeling Algorithms for Connected Components

Sixue Liu
Princeton University
sixuel@cs.princeton.edu

Robert E. Tarjan
Princeton University and Intertrust Technologies
ret@cs.princeton.edu

Abstract

We present new concurrent labeling algorithms for finding connected components, and we study their theoretical efficiency. Even though many such algorithms have been proposed and many experiments with them have been done, our algorithms are simpler. We obtain an $O(\lg n)$ step bound for two of our algorithms using a novel multi-round analysis. We conjecture that our other algorithms also take $O(\lg n)$ steps but are only able to prove an $O(\lg^2 n)$ bound. We also point out some gaps in previous analyses of similar algorithms. Our results show that even a basic problem like connected components still has secrets to reveal.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Connected Components, Concurrent Algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.3

Funding Research at Princeton University partially supported by an innovation research grant from Princeton and a gift from Microsoft.

Acknowledgements We thank Dipen Rughwani, Kostas Tsioutsiouliklis, and Yunhong Zhou for telling us about [18], for extensive discussions about the problem and our algorithms, and for insightful comments on our results.

1 Introduction

The problem of finding the connected components of an undirected graph with n vertices and m edges is fundamental in algorithmic graph theory. Any kind of graph search, such as depth-first or breadth-first search, solves it in linear time sequentially, which is best possible. The problem becomes more interesting in a concurrent model of computation. In the heyday of the theoretical study of PRAM algorithms, many more-and-more efficient algorithms for the problem were discovered, culminating in 2001 with the $O(\lg n)$-time, $O((m+n)/\lg n)$-processor randomized EREW PRAM algorithm of Halperin and Zwick [8], where lg is the base-two logarithm. Most of this work was motivated by obtaining the best bounds, not the simplest algorithms.

With the growth of the internet, the world-wide web, and cloud computing, computing connected components on huge graphs has become commercially important, and practitioners have put versions of the PRAM algorithms into use. Many of these algorithms are quite complicated, and even some of the simple ones have been further simplified when implemented. There is evidence that such simple algorithms perform well in practice, but claims about their theoretical performance are unsound.

Given this situation, our goal here is to develop and analyze the simplest possible efficient algorithms for the problem and to rigorously analyze their efficiency. In exchange for algorithmic simplicity, we are willing to allow analytic complexity. Our computational framework is based on the MPC (Massive Parallel Computing) model [3], which is a variant of the BSP (Bulk Synchronous Parallel) model [20]. The MPC model is more powerful than PRAM models in that it allows programmable resolution of write conflicts, but it is a realistic model of cloud computing platforms. Our algorithms also work on a CRCW (concurrent read, concurrent write) PRAM in which write conflicts are resolved in favor of the smallest value written. This model is stronger than the more standard ARBITRARY CRCW PRAM, in which write conflicts are resolved arbitrarily (one of the writes succeeds, but the algorithm has no control over which), but is weaker than the MPC model.

We develop several algorithms that are all simpler than existing algorithms. We conjecture that all of them take $O(\lg n)$ concurrent steps and $O(m \lg n)$ messages. We prove this bound for two of them using a novel multi-round analysis. For the others we obtain an $O(\lg^2 n)$ step bound and $O(m \lg^2 n)$ message bound.

Our paper contains five sections, in addition to this introduction, §2 presents our algorithmic framework, §3 presents our algorithms, §4 discusses related work, §5 completely analyzes one of our algorithms and partially analyzes the others, and §6 closes with some remarks and open problems.

2 Algorithmic Framework

Given an undirected graph with vertex set $[n] = \{1, 2, \ldots, n\}$ and m edges, we wish to compute its connected components via a concurrent algorithm. More precisely, for each component we want to label all its vertices with a unique vertex in the component, so that two vertices are in the same component if and only if they have the same label. To state bounds simply, we assume $n > 1$ and $m > 0$. We denote the ends of an edge e by $e.v$ and $e.w$.

We use the MPC (Massive Parallel Computing) model [3], specialized to our problem. (We discuss the MPC model further in §4.) We consider algorithms that operate in synchronous concurrent steps, each of which can send messages between the edges and vertices. Each edge and each vertex has a local memory. In a concurrent step, each edge and vertex can update its local memory based on the messages sent to it in the previous step, and then send a message to each vertex and edge that it knows about. Initially a vertex knows only about itself, and an edge knows only about its two ends. Thus in the first concurrent step only edges can send messages, and only to their ends. A vertex or edge *knows about* another vertex or edge once it has received a message containing the vertex or edge. We ignore contention resulting from many messages being sent to the same vertex or edge during a step, and the need for a vertex or edge to send many messages during a step. We measure the efficiency of an algorithm primarily by the number of concurrent steps and secondarily by the number of messages sent.

Even in this unrealistically strong model, there is a non-constant lower bound on the number of steps needed to compute connected components:

▶ **Theorem 1.** *Computing connected components takes $\Omega(\lg d)$ steps, where d is the maximum of the diameters of the components.*

Proof. Let u be the vertex that eventually becomes the label of all vertices in the component. Some vertex v is at distance at least $d/2$ from u; otherwise, the diameter of the component is less than d, a contradiction. An induction on the number of steps shows that after k steps a vertex has only received messages containing vertices within distance 2^k. Since v must receive a message containing u, the computation takes at least $\lg d - 1$ steps. If $d = 1$, the computation takes at least one step. ◀

If messages can be arbitrarily large, it is easy to solve the problem in $O(\lg d)$ steps, by first sending each edge end to the other end, and then repeatedly sending from each vertex the entire set of vertices it knows about to all these vertices [15]. If there is a large component, however, such an algorithm is not practical, for at least two reasons: it requires huge memory at each vertex, and the last step can send a number of messages quadratic in n. Hence we restrict the local memory of a vertex or edge to hold only a small constant number of vertices and edges. We also restrict messages to hold only a small constant number of vertices and edges, along with an indication of the message type, such as a label request or a label update.

All our algorithms maintain a label for each vertex u, initially u itself. Labels are updated step-by-step until none changes, after which all vertices in a component have the same label, which is one of the vertices in the component. At any given time the current labels define a univalent digraph (directed graph) whose arcs lead from vertices to their labels. We call this the *label digraph*. If these arcs form no cycles other than loops (arcs of the form (u, u)), then this graph is a forest of trees rooted at the self-labeled vertices: the parent of u is its label unless the label of u is u, in which case u is a root. We call this the *label forest*. Each tree in the forest is a *label tree*.

All our algorithms maintain the label digraph as a forest; that is, they maintain acyclicity except for self-labels. (We know of one previous algorithm that does not maintain the label digraph as a forest: see §4.) Henceforth we call the label of a vertex u its *parent* and denote it by $u.p$, and we call a label tree just a *tree*. If v is a vertex such that $v.p \neq v$, v is a *child* of $v.p$. A tree is *flat* if the parent of every child is the root. (Some authors call such a tree a *star*.) A vertex is a *leaf* if it is not a root and it has no children. We call a one-vertex tree a *singleton*.

During the computation, all the vertices in a tree are in a single connected component, but there may be more than one tree per component. At the end of the computation, there is one flat tree per component, whose root is the component label.

3 Algorithms

A simple way to guarantee acyclicity is to maintain the parent of a vertex to be the minimum of the vertices it knows about. We call this *minimum labeling*. (An equivalent alternative is to maintain the label of a vertex to be the maximum of the vertices it knows about.) All our algorithms use minimum labeling. Each algorithm proceeds in rounds, each of which updates parents using vertices received from edges, reduces tree depths using *shortcutting*, and possibly alters edges.

The most obvious way to use edges to update parents is for each edge to send its minimum end to its maximum end:

CONNECT: *for each edge* e, *send* $\min\{e.v, e.w\}$ *to* $\max\{e.v, e.w\}$.

This method does not give a correct algorithm unless we combine it with some form of edge alteration or edge addition. In the absence of changes to the edge set, we use one of the following methods instead of CONNECT:

PARENT-CONNECT: *for each edge* e, *request* $e.v.p$ *from* $e.v$ *and* $e.w.p$ *from* $e.w$; *send the minimum of the received vertices to the maximum of the received vertices.*

EXTENDED-CONNECT: *for each edge* e, *request* $e.v.p$ *from* $e.v$ *and* $e.w.p$ *from* $e.w$; *let the received values be* x *and* y, *respectively*; *if* $y < x$ *then send* y *to* v *and to* x *else send* x *to* w *and to* y.

The straightforward way to update labels using messages sent from edges is to replace each parent by the minimum of the received vertices:

UPDATE: *for each vertex v, replace $v.p$ by the minimum of $v.p$ and vertices received from edges.*

Such updating can move subtrees between trees in the forest, producing behavior that is hard to analyze. A more conservative alternative is to update only the parents of roots:

ROOT-UPDATE: *for each root v, replace $v.p$ by the minimum of $v.p$ and vertices received from edges.*

Shortcutting is the key to obtaining a logarithmic step bound. Shortcutting replaces the parent of each vertex by its grandparent:

SHORTCUT: *for each vertex v request $v.p.p$ from $v.p$; replace $v.p$ by the received vertex.*

Edge alteration replaces each edge end by its parent:

ALTER: *for each edge e, request $e.v.p$ from $e.v$ and $e.w.p$ from $e.w$; let the returned vertices be x and y, respectively; if $x = y$ then delete e else replace $e.v$ and $e.w$ by x and y.*

Choosing one of the three connection methods, one of the two update methods, and whether or not to alter edges produces one of twelve algorithms. Not all these algorithms are correct: if we use CONNECT, we must choose ALTER to alter edges; if we do not, the partition of vertices defined by the trees does not change after the first connect. Furthermore not all the correct algorithms are as simple as possible: if we use ALTER, using CONNECT gives a correct algorithm, and we see no advantage in using PARENT-CONNECT or EXTENDED-CONNECT, which are more complicated. (We have no theoretical or practical evidence to justify this intuition, however.) Finally, if we use ROOT-UPDATE, PARENT-CONNECT and EXTENDED-CONNECT are equivalent. This leaves us with five algorithms:

- Algorithm P: *repeat* {PARENT-CONNECT; UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm E: *repeat* {EXTENDED-CONNECT; UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm A: *repeat* {CONNECT; UPDATE; SHORTCUT; ALTER} *until no parent changes.*
- Algorithm R: *repeat* {PARENT-CONNECT; ROOT-UPDATE; SHORTCUT} *until no parent changes.*
- Algorithm RA: *repeat* {CONNECT; ROOT-UPDATE; SHORTCUT; ALTER} *until no parent changes.*

We call an iteration of the repeat loop in each of these algorithms a *round*. Two of these algorithms, R and RA, are equivalent:

▶ **Theorem 2.** *Algorithms R and RA do the same parent updates in each step.*

Proof. Consider running algorithms R and RA concurrently. We prove by induction on the number of steps that (i) parents are the same in both algorithms and (ii) if edge e has original ends $e.v = v$ and $e.w = w$ and has ends $e.v = x$ and $e.w = y$ in RA, then $v.p = x$ and $w.p = y$ in both R and RA. Both (i) and (ii) hold initially. Suppose they hold at the beginning of a round. Then the CONNECT and ROOT-UPDATE in RA do the same parent changes as the PARENT-CONNECT and ROOT-UPDATE in R, so (i) holds after these steps. The SHORTCUT steps in R and RA also do the same parent changes, so (i) holds after this step. The ALTER at the end of RA re-establishes (ii). ◀

Henceforth we treat R and RA as different implementations of the same algorithm. Algorithms A and P are not equivalent in the same sense, as the example in Figure 1 shows.

To prove correctness, we need the following key result:

▶ **Lemma 3.** *All our algorithms maintain the invariant that any two tree roots in the same component are connected by a path of current edges.*

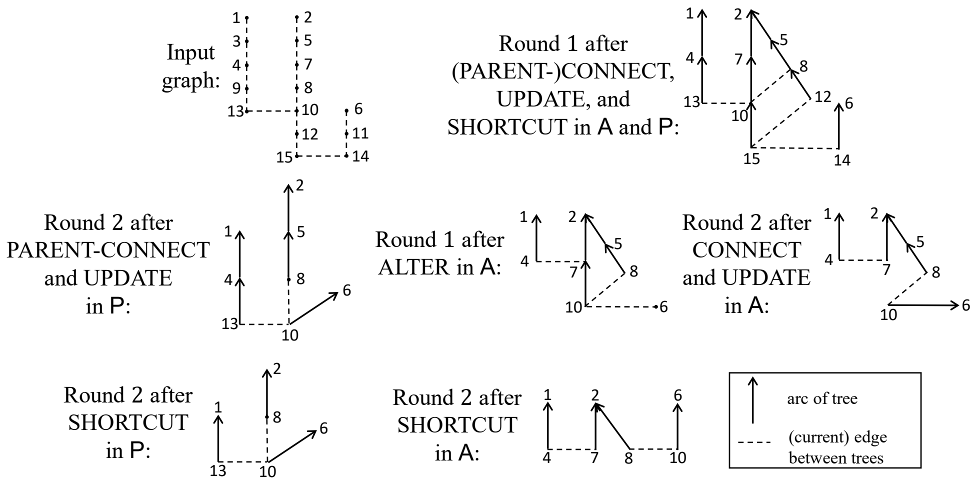

Figure 1 A graph on which A and P do different parent updates. Only necessary vertices, edges, and tree arcs are shown. In round 1 before the ALTER, all arcs and edges are the same in both algorithms. In round 2, the parent of 10 becomes 6 in both algorithms. In round 3 (not shown), P changes the parent of 6 to 1 using the original edge $(10, 13)$, but A changes the parent of 6 to 2 using the new edge $(2, 6)$ altered from $(8, 10)$.

Proof. This is a tautology for P, E, and R. We prove it for A by induction on the number of steps. (The same proof works for RA.) A SHORTCUT changes no edges nor roots; a CONNECT followed by an UPDATE changes no edges and only converts roots to non-roots. Thus both preserve the invariant, since both preserve every path of current edges, and vertices that are roots after an UPDATE are roots before the preceding CONNECT. Given a path between two roots, an ALTER preserves each root it contains, including the ends of the path, and replaces each edge end that is a child by its parent. Thus an ALTER also preserves the invariant. ◀

▶ **Theorem 4.** *All our algorithms are correct.*

Proof. For each algorithm, an induction on the number of steps proves that the vertex sets of the trees are subsets of the vertex sets of the connected components. We prove that each algorithm cannot stop until there is one flat tree per component, which implies correctness. Consider a component whose vertices are in two or more trees just before a round. By Lemma 3, some current edge connects a vertex in one of these trees with a vertex in another tree. If one of these trees is not flat, the SHORTCUT during the round will change some parent if no parents change before the SHORTCUT. If both are flat, some parent will change before the SHORTCUT. ◀

In §5 we prove the following step bounds: $O(d)$ for A and E; $O(\lg^2 n)$ for A, P, and E; and $O(\lg n)$ for R and RA. We conjecture that P, A, and E in fact have an $O(\lg n)$ step bound. We leave a tighter analysis of these algorithms as an open problem. We have included algorithm E because it is similar to an existing algorithm (see §4), and we can prove a step bound that depends only on the diameter.

We conclude this section with five observations.

First, our algorithms need very little memory: one cell per vertex to hold its parent, and two cells per edge to hold its current ends.

Second, although we have used a message-passing framework, it is straightforward to implement our algorithms on a CRCW PRAM in which write conflicts are resolved in favor of the smallest value written. We say more about the effect of write conflict resolution on connected components algorithms in §4.

Third, in algorithm RA, the condition for edge deletion can be strengthened, since when an edge causes a parental root update it connects a child with its parent, and it can never again cause a parent update. Thus the condition for deletion in the ALTER can be strengthened to "*if* $x = y$ *or* $e.v = y$ *or* $e.w = x$ *then delete* e". This optimization does *not* work in algorithm A.

Fourth, algorithms like R that use root updating are *monotone* in that the vertex set of each new tree in the label forest is a union of vertex sets of old trees. This is *not* true of algorithms such as A, P, and E, which *can* and in general *do* move subtrees between trees. Monotonicity seems to make analysis simpler. Many but not all previous algorithms are monotone.

Fifth, each of our algorithms does one SHORTCUT per round, but we could increase this to two or indeed to any number. Our $O(\lg^2 n)$ bound for P, E, and A extends to the variants of these algorithms that do any fixed positive number of SHORTCUT steps each round. Similarly, our $O(\lg n)$ bound for R and RA extends to the variants of these algorithms that do any fixed positive number of SHORTCUT steps each round. On the other hand, if we modify R (or RA) to do enough SHORTCUT steps each round to flatten all the trees, the worst-case number of steps becomes $\Theta(\lg^2 n)$, as we now show.

Algorithm S: *repeat* {PARENT-CONNECT; ROOT-UPDATE;
repeat SHORTCUT *until no parent changes*} *until no parent changes.*

Algorithm SA: *repeat* {CONNECT; ROOT-UPDATE;
repeat SHORTCUT *until no parent changes*; ALTER} *until no parent changes.*

In both of these algorithms, all trees are flat just before a CONNECT, which implies that CONNECT only changes the parents of roots. A proof like that of Theorem 2 shows that S and SA are equivalent in that they make the same parent changes. We thus treat them as alternative implementations of the same algorithm.

▶ **Theorem 5.** *Algorithm S (and SA) takes* $O(\lg^2 n)$ *steps.*

Proof. The inner repeat loop stops after at most $\lceil \lg n \rceil$ steps with all trees flat. We claim that if there are two or more trees, two rounds of the outer loop at least halve their number, from which the theorem follows. Call a root *minimal* if its tree has edges connecting it only with trees having greater roots. A CONNECT makes every non-minimal root into a non-root. Suppose there are k roots just before a CONNECT, of which j are minimal. If $j < k/2$, the CONNECT reduces the number of roots to at most $j < k/2$. Divide the minima into the i that get a new child as a result of the CONNECT and the $j - i$ that do not. If x is a minimal that does not get a new child, there must be a root $y > x$ such that the trees rooted at x and y are connected by an edge, and y has parent less than x after the CONNECT. The next CONNECT will make x a child. That is, the current CONNECT makes at least i roots into non-roots, and the next CONNECT makes at least $j - i$ into non-roots, for a total of at least $j \geq k/2$, giving the claim in this case as well. ◀

We show by example that the bound in Theorem 5 is tight. For convenience we consider algorithm SA. During the execution, at the beginning of each round it suffices to consider only the induced subgraph of vertices that are roots. This is because in each round before the ALTER, every tree is flat, so the ALTER makes all edge ends roots.

Observe that if there is a tree path of $2^k + 1$ vertices just before the SHORTCUT steps in round j, there will be $\Omega(k + 1)$ SHORTCUT steps in round j. If in every round there is a *new* path of $2^k + 1$ vertices, and the algorithm requires many rounds, the input graph will be a bad example. Our example is a disjoint union of certain graphs based on this observation. To produce a given graph G with m' edges and vertex set $[n']$ at the end of round j, it suffices to start with a graph $g(G)$ (for the *generator* of G) at the beginning of round j constructed as follows: For each vertex i in G, add two vertices, i and $i+n'$ to $g(G)$; for each vertex i in G, add an edge $(i, i+n')$ to $g(G)$; for each edge (i, i') in G, add an edge $(i+n', i'+n')$ to $g(G)$. $g(G)$ contains all the vertices of G but none of its edges; $g(G)$ contains $2n'$ vertices and $n'+m$ edges.

Consider the effect of a round of SA on $g(G)$. The CONNECT step makes i the parent of $i+n'$ for $i \in [n']$. The SHORTCUT steps do nothing. The ALTER converts each edge $(i+n', i'+n')$ for $i \in [n']$ into (i, i') and deletes each edge $(i, i+n')$ for $i \in [n']$. Thus after the round, G is the induced subgraph on the vertices that are roots. By induction, $g^r(G)$ is converted by r rounds of SA into G. $g^r(G)$ contains $n'2^r$ vertices and $n'(2^r-1)+m'$ edges.

For any positive integer $k > 1$, consider the disjoint union of the graphs P, $g(P)$, $g^2(P)$, $\ldots$, $g^{k-1}(P)$, where P is a path of 2^k+1 vertices from 1 to 2^k+1, with the vertices renumbered so that all vertices are distinct and the order within each connected component is preserved. If SA is run on this graph, round i for each i does $\Omega(k+1)$ SHORTCUT steps on the path produced by $i-1$ rounds on $g^{i-1}(P)$, so the total number of steps is $\Omega(k^2)$. The total number of vertices in this graph is $n = (2^k+1)(2^k-1) = 2^{2k}-1$. (The number of edges is $2^{2k}-k+1$: the input graph is a set of $k-1$ trees.) Thus the number of steps is $\Omega(\lg^2 n)$, making the bound in Theorem 5 tight.

4 Related Work

Previous work on concurrent algorithms for connected components was done by two different communities in two overlapping eras. First, theoretical computer scientists developed provably efficient algorithms for various versions of the PRAM (parallel random-access machine). This work began in the late 1970's and reached a natural conclusion in the work of Halperin and Zwick [8, 9], who gave $O(\lg n)$-time, $O((m+n)/\lg n)$-processor randomized algorithms for the EREW (exclusive read, exclusive write) PRAM. The EREW PRAM is the weakest PRAM model, and computing connected components in this model requires $\Omega(\lg n)$ time [5]. To solve the problem sequentially takes $O(n+m)$ time, so the Halperin-Zwick algorithms minimize both the time and the total work (number of processors times time). One of their algorithms not only finds the connected components but also a spanning tree of each. The main theoretical question remaining open is whether a deterministic algorithm can achieve the same bounds.

Halperin and Zwick's paper contains a table listing results preceding theirs, and we refer the reader to their paper for these results. Our interest is in simple algorithms for a more powerful computational model, so we content ourselves here with discussing simple labeling algorithms related to ours. (The Halperin-Zwick algorithms and many of the preceding ones are *not* simple.) First we review variants of the PRAM model and how they relate to our algorithmic framework.

The three main variants of the PRAM model, in increasing order of strength, are EREW, CREW (concurrent read, exclusive write), and CRCW (concurrent read, concurrent write). The CRCW PRAM has four standard versions that differ in how they handle write conflicts: (i) COMMON: all writes to the same location at the same time must be the of the same value; (ii) ARBITRARY: among concurrent writes to the same location, an arbitrary one succeeds; (iii) PRIORITY: among concurrent writes to the same location, the one done by the highest-priority processor succeeds; (iv) COMBINING: values written concurrently to a given location are combined using some symmetric function. As mentioned in §3, our algorithms can be implemented on a COMBINING CRCW PRAM, with minimization as the combining function.

The first $O(\lg n)$-time PRAM algorithm was that of Shiloach and Vishkin [17]. It runs on an ARBITRARY CRCW PRAM, as do the other algorithms we discuss, except as noted. The following is a version of their algorithm in our framework:

Algorithm SV: *repeat* {SHORTCUT; PARENT-CONNECT; ARBITRARY-ROOT-UPDATE; MAX-PARENT-CONNECT; PASSIVE-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

This algorithm uses the notion of a passive tree. (Shiloach and Vishkin used the term "stagnant".) A tree is *passive* if it did not change in the previous round. A passive tree is necessarily flat, but a flat tree need not be passive. Our description of their algorithm omits the extra computation needed to keep track of which trees are passive.

The algorithm uses variants of some of the methods in §3. Method ARBITRARY-ROOT-UPDATE replaces each parent of a root by *any* of the most-recently received vertices other than itself, if there are any, instead of the minimum; MAX-PARENT-CONNECT sends for each edge the maximum of the parents of the edge ends to the minimum instead of vice-versa; PASSIVE-ROOT-UPDATE replaces each root of a passive tree by any just-received vertex other than itself, if there are any:

ARBITRARY-ROOT-UPDATE: *for each root v replace $v.p$ by any just-received vertex other than v, if there is one.*

MAX-PARENT-CONNECT: *for each edge e, request $e.v.p$ from $e.v$ and $e.w.p$ from $e.w$; send the maximum of the received vertices to the minimum of the received vertices.*

PASSIVE-ROOT-UPDATE: *for each root v of a passive tree, replace $v.p$ by any just-received vertex other than v, if there is one.*

Algorithm SV does neither minimum nor maximum labeling: the first update in a round connects roots to smaller vertices, the second connects roots to larger vertices. The proof that the algorithm creates no cycles is non-trivial, as is the efficiency analysis. Most interesting for us, the first three steps of the algorithm are algorithm R, but with arbitrary parent updates rather than minimum ones. Shiloach and Vishkin claimed that the second SHORTCUT can be omitted, but they showed by example that omission of PASSIVE-ROOT-UPDATE results in an algorithm that can take $\Omega(n)$ rounds. Their example also shows that the efficiency of algorithm R depends on resolving concurrent parent updates by minimum value rather than arbitrarily. That is, our stronger model of computation is critical in obtaining a simpler algorithm.

Awerbuch and Shiloach presented a simpler $O(\lg n)$-time algorithm and gave a simpler efficiency analysis [2]. Our analysis of algorithm RA in §5 uses a variant of their potential function. Their algorithm does only one SHORTCUT per round and in updates only changes the parents of roots of flat trees, using the following method:

FLAT-ROOT-UPDATE: *for each root v of a flat tree, replace $v.p$ by any just-received vertex other than v, if there is one.*

Algorithm AS: *repeat* {PARENT-CONNECT; FLAT-ROOT-UPDATE; MAX-PARENT-CONNECT; FLAT-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

The algorithm needs to do additional computation to keep track of flat trees. Although Awerbuch and Shiloach do not mention it, their analysis shows that replacing the first update in their algorithm with ARBITRARY-ROOT-UPDATE produces a correct $O(\lg n)$-time algorithm. The resulting algorithm is algorithm SV with the first SHORTCUT deleted and PASSIVE-TREE-CONNECT replaced by FLAT-TREE-CONNECT.

An even simpler but randomized $O(\lg n)$-time algorithm was proposed by Reif [16]:

Algorithm Reif: *repeat* {*for each vertex flip a coin*; RANDOM-CONNECT; ARBITRARY-ROOT-UPDATE; SHORTCUT} *until no parent changes.*

RANDOM-CONNECT: *for each edge* (v, w) *if* $v.p$ *flipped heads and* $w.p$ *flipped tails then send* $v.p$ *to* $w.p$; *if* $w.p$ *flipped heads and* $v.p$ *flipped tails then send* $w.p$ *to* $v.p$.

Reif's algorithm keeps the label trees flat. As a result, RANDOM-CONNECT only sends vertices to roots. This allows replacement of ARBITRARY-ROOT-UPDATE by a method that for each vertex sets its parent equal to any just-received vertex if there is one. Reif's algorithm is simpler than both SV and AS, but our algorithm RA is even simpler, and it is deterministic.

We know of only one algorithm, that of Johnson and Metaxis [12], that does not maintain acyclicity. Their algorithm runs in $O((\lg n)^{3/2})$ time on an EREW PRAM. It does a form of SHORTCUT to eliminate any non-trivial cycles that it creates.

Algorithms that run on a more restricted form of PRAM, or use fewer processors (and thereby do less work) use various kinds of edge alteration, along with simulation and other techniques to resolve read and write conflicts. Such algorithms are much more complicated than those above. Again we refer the reader to [8] for results and references.

The second era of concurrent connected components algorithms was that of the experimentalists. It began in the 1990's and continues to the present. Experimentation has expanded greatly with the growing importance of huge graphs (the internet, the world-wide web, relationship graphs, and others) and the development of cloud computing frameworks. These trends make concurrent algorithms for connected components both practical and useful. The general approach of experimentalists has been to take one or more algorithms in the literature, possibly simplify or modify them, implement the resulting suite of algorithms on one or more computing platforms, and report the results of experiments done on some collection of graphs. Examples of such studies include [6, 7, 10, 21, 14, 18].

Our interest is in the theoretical efficiency of such simple algorithms, and in the theoretical power of the new concurrent computing platforms as compared to the classical PRAM model. One theoretical model used to study algorithms on such platforms is the MPC (Massive Parallel Computing) model. A large number of virtual processes run concurrently in supersteps. In each superstep, a process can do arbitrary computation based on messages it received during the previous step, and then send messages to any or all other processes. The next superstep begins once all messages are received. Our framework for connected components algorithms is the MPC model, specialized for the kind of algorithms we consider.

Our work started with a study of the following algorithm of Stergio, Rughwani, and Tsioutsiouliklis [18]:

Algorithm SRT: *repeat* {
for each edge (v, w) *if* $v.p < w.p$ *then send* $v.p$ *to* w *else send* $w.p$ *to* v;
for each vertex v *let* $new(v)$ *be the minimum of* $u.p$ *and the vertices received by* v;
for each vertex v *if* $new(v) < v.p$ *then send* $new(v)$ *to* $v.p$;
for each vertex v *send* $new(v).p$ *to* v;
for each vertex v *let* $v.p$ *be the minimum of* $v.p$ *and the vertices received by* v}
until no parent changes.

They implemented this algorithm on the Hronos computing platform and solved problems with trillions of edges. They claimed a bound of $O(\lg n)$ steps for their algorithm. But we are unable to make sense of their proof. This algorithm moves subtrees between trees, which makes it hard to analyze. We conjecture, however, that our proof of $O(\lg^2 n)$ steps for algorithms P, E, and A extends to their algorithm.

Our algorithm E is a variant of algorithm SRT that is bit simpler. Algorithm P is a further simplification. Algorithm R is algorithm P with updates restricted to roots. We are able to obtain an $O(\lg n)$ step bound for algorithms R and RA, but not for E, P, nor A. Our diameter-based analysis of algorithm E is also valid for algorithm SRT and gives a bound of $O(d)$ steps. We think that algorithms SRT, E, P, and A all take $O(\lg n)$ steps but have no proof.

A second paper with an analysis gap is that of Yan et al. [21]. They consider algorithms in the PREGEL framework [13], which is a graph-processing platform designed on top of the MPC model. All the algorithms they consider can be expressed in our framework. They give a version of algorithm SV with the first SHORTCUT deleted, and modify it further by replacing "MAX-PARENT-CONNECT; PASSIVE-ROOT-UPDATE" with code equivalent to "PARENT-CONNECT; FLAT-ROOT-UPDATE". These two steps do nothing, since any connections they might make are done by the previous steps "PARENT-CONNECT; ARBITRARY-ROOT-UPDATE". Their termination condition, that all trees are flat, is also incorrect. They claim an $O(\lg n)$ step bound for their algorithm, but since they use arbitrary updating of roots, their algorithm, once the termination condition is corrected, takes $\Omega(n)$ steps on the example of Siloach and Vishkin.

A third, more recent paper with an analysis gap is that of Burkhardt [4]. He proposes an algorithm that does a novel form of edge alteration: it converts each original edge into two oppositely directed arcs and alters these arcs independently. The algorithm does not do SHORTCUT explicitly, but it does do a variant of SHORTCUT implicitly. The algorithm maintains both old and new vertex labels, which increases its complexity. Burkhardt claims a step bound of $O(\lg d)$, but a counterexample in [1] disproves this claim. He also claims a linear space bound, but we are unable to verify his proof of this. We conjecture that Burkhardt's algorithm and simpler variants take $O(\lg n)$ steps but have no proof. We think that our $O(\lg^2 n)$ analysis should apply to his algorithm but have not verified this.

Very recently, Andoni et al. have used the power of the MPC model to obtain a randomized algorithm running in $O(\log d \log \log_{m/n} n)$ steps [1]. Their algorithm uses the distance-doubling technique of [15] (discussed in §2) but controlled to keep message sizes sufficiently small. The algorithm relies heavily on the ability to sort in $O(1)$ steps on the MPC model. We think an appropriate version of their algorithm can be implemented on a CRCW PRAM, possibly with the same asymptotic bound as theirs, and we are working to achieve this. Any such algorithm will be much more complicated than those we present here, however.

5 Analysis

In this section we prove an $O(d)$ step bound for algorithms E and A, an $O(\lg^2 n)$ bound for algorithms P, E, and A, and an $O(\lg n)$ bound for algorithm R (and RA). We begin with some assumptions and preliminary results. We assume the graph is connected and contains at least two vertices, which is without loss of generality since each algorithm operates concurrently and independently on each component. For brevity, we use CONNECT to denote the steps preceding SHORTCUT in a round, no matter which algorithm we are considering. We denote an edge e with $e.v = v$ and $e.w = w$ by (v, w). For the analysis only, we assume that ALTER does not delete an edge (v, w) when $v.p = w.p$ but instead converts it into a loop (an edge having both ends the same) $(v.p, w.p)$. Once an edge becomes a loop, it remains a loop. Loops do not affect the parent changes done by A (or RA); they merely allow us to treat a vertex that is both ends of a loop as having an incident edge.

We partition the vertices into two colors, as follows: in algorithm A, a vertex is *green* if it is a root or has an incident edge, and *red* otherwise; in all other algorithms a vertex is *red* if it is a leaf (a non-root with no children) and *green* otherwise.

▶ **Lemma 6.** *If a* CONNECT *changes the parent of a vertex v to w, then both v and w are green.*

Proof. In algorithm A, v and w must be the ends of the edge causing the edge, so both are green. In the other algorithms, there must be an edge (x, y) such that $x.p = v$ and $y.p = w$. Hence both v and w are green. ◀

▶ **Lemma 7.** *A* SHORTCUT *in an algorithm other than A changes all green non-roots with no green children to red, and changes no other vertex colors.*

Proof. Let v be any vertex just before a SHORTCUT. If v is red, it is a non-root with no child. The SHORTCUT cannot give it a child, so it stays red. If v is a green root, it stays a root, so it stays green. If v is a green non-root, v is a non-root after the SHORTCUT, and it has a child after the SHORTCUT if and only if it has a grandchild before the SHORTCUT, which is true if and only if it has a green child before the SHORTCUT. ◀

▶ **Lemma 8.** *In algorithm A, (i) each green vertex is a root or has a green parent; (ii) each red vertex has a green grandparent, and has a green parent just after a* SHORTCUT*; and (iii) a* SHORTCUT *followed by an* ALTER *changes all green non-roots with no green children to red, and changes no other vertex colors.*

Proof. By induction of the number of steps. The lemma is true initially since all vertices are roots and hence green. By Lemma 6, a CONNECT preserves the lemma, since it does not change any vertex colors nor the parent of any red vertex. A SHORTCUT preserves (i) and (ii), since before the SHORTCUT any vertex has a green grandparent or parent or is itself green, so after the SHORTCUT it has either a green parent or is green. Given that all vertices have green parents after a SHORTCUT, the subsequent ALTER changes each edge end that is a green non-root to its parent, and does not affect ends that are roots. (Here we use the non-deletion of loops.) This makes no red vertex green by (i), makes a green vertex red if and only if it is a non-root with no green children, giving (iii), and leaves each red vertex with a green parent or grandparent by (i) and (ii). ◀

▶ **Lemma 9.** *The parent of a vertex never increases. Once a vertex is a non-root, it stays a non-root.*

Proof. The first part of the lemma follows by induction on the number of parent changes: each such change decreases the parent. By the first part, once $v.p < v$ this inequality continues to hold, which gives the second part of the lemma. ◀

5.1 A Diameter Bound

▶ **Theorem 10.** *Algorithm E takes $O(d)$ steps.*

Proof. Let u be the minimum vertex. We prove by induction on i that after i rounds all vertices at distance i or less from u have parent u. This is true for $i = 0$. Let w be a vertex at distance $i > 0$ from u. Then there is an edge (w, v) with v at distance $i - 1$ from u. By the induction hypothesis, v has parent u after round $i - 1$, which it will send to w in round i, as a result of which w will have parent u after round i. After at most d rounds, all vertices will have parent u, so the algorithm stops after at most $d + 1$ rounds. ◀

Theorem 10 also holds for algorithm A, but the proof is more elaborate. We need a strengthening of the result of Lemma 3, as well as Lemma 8.

▶ **Lemma 11.** *Let u be the minimum vertex. After $d-i$ rounds of algorithm* A*, each green vertex has a path of at most $d-i$ current edges connecting it with u.*

Proof. By induction on i. The lemma is true initially, since very vertex has a path to u containing at most d edges. Suppose it is true just before round i. Let v be a green vertex at the end of round i. If v is a root at the end of round i, then it was a root at the start of round i, and by the induction hypothesis there was a path P of at most $d-i$ edges connecting v with u at the start of round i. If v is not a root at the end of round i, then it was not a root at the beginning of the SHORTCUT in round i, and by Lemma 8 it had a green child, say w, at this time. By the induction hypothesis, there was a path P of at most $d-i$ edges connecting w with u at the start of round i. In either case we can assume P is loop-free, since deleting loops leaves it a path with the same ends. The ALTER in round i converts P into a path P' connecting v with u containing at most $d-i$ edges. Delete all loops from P'. Let (x,u) be the last edge on P. After the CONNECT in round i, x must be a child of u. Hence the ALTER in round i converts this edge to a loop. Deleting this loop from P' gives a path satisfying the lemma for v after round i. ◀

▶ **Theorem 12.** *Algorithm A takes $O(d)$ steps.*

Proof. Consider the state after d steps. By Lemma 11, the only green vertex is the minimum vertex, and by Lemma 8 all red vertices are children or grandchildren of this minimum vertex. The algorithm stops after at most two more rounds. ◀

5.2 A Log-Squared Bound

We conjecture that all our algorithms take $O(\lg n)$ steps, but for P, E, and A we are only able to prove something weaker:

▶ **Theorem 13.** *Algorithms P, E, and A take $O(\lg^2 n)$ steps.*

To prove Theorem 13, we shall show that $O(\lg n)$ rounds reduce the number of green vertices by at least a factor of two. Given Lemma 6, it is convenient to consider the situation just before a SHORTCUT. By Lemma 7 or 8 depending on the algorithm, a SHORTCUT converts all green non-roots with no green children to red. We shall prove that a CONNECT converts all green roots with no green children into non-roots. This allows us to bound the number of rounds in which there are many green vertices with no green children. To bound the number of rounds in which there are many green vertices but few that have no green children, we prove that in such a situation the green vertices form many long vertex-disjoint tree paths, on which we can quantify the effect of a SHORTCUT. We proceed with the details. We assume throughout this section that the algorithm is P, E, or A.

▶ **Lemma 14.** *Let v be a root with no green children just before the* SHORTCUT *in round i. Either v is the only root, or the* CONNECT *in round $i+1$ makes v a non-root.*

Proof. Suppose v is not the only root just before the SHORTCUT in round i. In algorithm A, just before this SHORTCUT there must be an edge (v,w) with w not in the tree rooted at v. If $w<v$, then the edge formed from (v,w) by the ALTER in round i will cause the CONNECT in round $i+1$ to make v a non-root. If $w>v$, then $w.p<v$ after the CONNECT in round i; otherwise, this CONNECT would have made w, a green vertex, a child of v. The ALTER in round i converts (v,w) into an edge (v,x) with $x<v$, which causes the CONNECT in round $i+1$ to make v a non-root in this as well.

A similar argument applies to algorithms P and E. Just before the SHORTCUT in round i, there must be an edge (x,y) with $x.p=v$ and with $y.p$ not in the tree rooted at v. If

$y.p < v$, (x, y) will cause the CONNECT in round $i+1$ to make v a non-root. If $y.p > v$, then $y.p.p < v$ after the CONNECT in round i; otherwise, this CONNECT would have made $y.p$, a green vertex, a child of v. After the SHORTCUT in round i, $y.p < v$, so in this case also (v, w) causes the CONNECT in round $i+1$ to make v a non-root. ◀

▶ Remark. Lemma 14 does *not* hold for algorithm R: a root with no green children can remain a root for many rounds.

The *depth* of a vertex v is the number of proper ancestors in its tree; that is, the number of tree arcs on the path from v to the root.

▶ **Lemma 15.** *If there are at least n' green vertices, with less than n'/k having no green child, then there is a green vertex of depth at least k.*

Proof. Every green vertex has a green descendant with no green children (possibly itself). Any two green vertices of the same depth are unrelated and hence must have distinct green descendants with no green children. Hence the number of green vertices of any given depth is less than n'/k. Since there are at least n' green vertices, and less than n'/k of each depth from 0 to $k-1$, there must be at least one of depth k. ◀

▶ **Lemma 16.** *If there are n' green vertices, with less than $n'/(2k)$ having no green child, then there is a set of vertex-disjoint tree paths of green vertices, each containing at least $k+1$ vertices, that together contain at least $n'/2$ green vertices.*

Proof. Find a green vertex of maximum depth and delete the tree path from it to the root of its tree. This deletion creates no new green vertex with no green child. Repeat this step until there are fewer than $n'/2$ vertices. By Lemma 15, each path deleted contains at least $k+1$ vertices. ◀

To quantify the effect of a SHORTCUT on long paths, we borrow an idea from the analysis of compressed-tree algorithms for disjoint set union [19]. Suppose there are n' green vertices. For the purpose of the analysis only, we renumber the green vertices from 1 to n' in the order of their original numbers and identify each green vertex by its new number. We define the *level* $v.l$ of a green child v to be $v.l = \lfloor \lg(v - v.p) \rfloor$. The level of a green child is between 0 and $\lg n'$. By Lemma 9, which holds for the new vertex numbers as well as the original ones, the level of a child never decreases. We show that if k is big enough a SHORTCUT increases by $\Omega(k)$ the sum of the levels of the green vertices on a tree path of k green vertices.

▶ **Lemma 17.** *Let n' be the number of green vertices. Consider a tree path of at least $k+2$ green vertices, where $k \geq 2 \lg n'$. A* SHORTCUT *increases the sum of the levels of the children on the path by at least $k/4$.*

Proof. Let u be a green vertex on the path other than the last two. Let v be the parent and w the grandparent of u, and let i and j be the levels of u and v, respectively. A SHORTCUT increases the level of u from $i = \lfloor \lg(u-v) \rfloor$ to $\lfloor \lg(u-w) \rfloor = \lfloor \lg(u-v+v-w) \rfloor \geq \lfloor \lg(2^i+2^j) \rfloor$. If $i < j$, this increases the level of u to at least j; if $i = j$, it increases the level of u to $i+1$.

Let $x_1, x_2, \ldots, x_{k+1}$ be the vertices on the path, excluding the last one. For each i from 1 to k, let $\Delta_i = x_{i+1}.l - x_i.l$. The sum of Δ_i's is $\Sigma = x_{k+1}.l - x_1.l > 0 - \lg n'$ since they form a telescoping series. Let k_+, k_0, and k_-, respectively be the number of positive, zero, and negative Δ_i's, and let Σ_+ and Σ_- be the sum of the positive Δ_i's and the sum of the negative Δ_i's, respectively. Then $\Sigma = \Sigma_+ + \Sigma_-$, which implies $\Sigma_+ > -\Sigma_- - \lg n'$. Since the Δ_i's are integers, $\Sigma_+ \geq k_+$ and $-\Sigma_- \geq k_-$. Combining inequalities, we obtain $2\Sigma_+ > k_+ + k_- - \lg n'$. Adding k_0 to both sides gives $2\Sigma_+ + k_0 > k - \lg n' \geq k/2$. Hence $\Sigma_+ + k_0 \geq \Sigma_+ + k_0/2 > k/4$. The lemma follows from the argument in the previous paragraph. ◀

Now we have all the pieces. It remains to put them together.

▶ **Lemma 18.** *Suppose there are $n' \geq 16$ green vertices just before a round. After $O(\lg n')$ rounds, there are at most $n'/2$ green vertices.*

Proof. At the beginning of the round, renumber the green vertices from 1 to n' in the order of their original numbers and identify each green vertex by its new number. Consider a round in which there are still at least $n'/2$ green vertices just before the SHORTCUT. We consider three cases:

(i) There are at least $n'/(16 \lg n' + 8)$ green roots with no green children. By Lemma 14, all such roots become non-roots during the next CONNECT. This can happen at most $16 \lg n' + 7$ times, since there is always a green root.

(ii) There are at least $n'/(16 \lg n' + 8)$ green non-roots with no green children. By Lemma 7 or 8 depending on the algorithm, the SHORTCUT makes all such vertices red. This can happen at most $16 \lg n' + 8$ times.

(iii) There are fewer than $n'/(8 \lg n' + 4)$ green vertices with no green children. By Lemma 16, there is a set of vertex-disjoint green paths, each containing at least $2 \lg n' + 2$ vertices, that together contain at least $n'/4$ vertices. By Lemma 17, for any such path containing $k + 2$ vertices, the SHORTCUT increases the levels of the vertices on the path by at least $k/4 \geq k/5 + 2/5$, since $k \geq 2 \lg n' \geq 8$. Summing over all the paths, the SHORTCUT increases the sum of levels by at least $n'/20$. This can happen at most $20 \lg n'$ times.

We conclude that after $O(\lg n')$ rounds, the number of green vertices is at most $n'/2$. ◀

Theorem 13 is immediate from Lemma 18.

5.3 A Logarithmic Bound

The proof of Theorem 13 fails for algorithm R, because Lemma 14 is false for this algorithm. But we can get an even better bound by using a different technique, that of Awerbuch and Shiloach [2] extended to cover a constant number of rounds rather than just one.

▶ **Theorem 19.** *Algorithms R (and hence RA) takes $O(\lg n)$ steps.*

To prove Theorem 19, we begin with some preliminary results and some terminology.

▶ **Lemma 20.** *After two rounds of algorithm R, each tree contains at least two vertices.*

Proof. Let v be a vertex and (v, w) an incident edge. If $w < v$, then v becomes a non-root in the first CONNECT. If $w > v$, then the first CONNECT either makes w a child of v or gives w a parent less than v. In the latter case, if v is still a root before the second CONNECT then this CONNECT will make v a non-root. We conclude that after the first two CONNECT steps v is in a tree with at least two vertices. ◀

By Lemma 20, after round two, all trees have height at least one. When we speak of a tree *in round k*, we mean a tree existing at the end of the round. We say a CONNECT *link* trees T_1 and T_2 if it makes the root of one of the trees a child of a vertex in the other. If the CONNECT makes the root of T_1 (respectively T_2) a child of a vertex in T_2 (respectively T_1), we say the CONNECT *links* T_1 to T_2 (respectively T_2 to T_1).

▶ **Definition 21.** A tree in round $k > 2$ is *passive* in round k if the tree existed at the beginning of the round (round k does not change it), and *active* otherwise. All trees in round 2 are *active* in round 2.

▶ **Lemma 22.** *For any integer $k > 2$, if trees T_1 and T_2 are passive in round $k-1$, the* CONNECT *in round k does not link T_1 and T_2, and there is no edge with one end in T_1 and the other in T_2.*

Proof. If T_1 and T_2 were linked in round k, there would be an edge connecting them that caused the link. Since T_1 and T_2 did not change in round $k-1$, such an edge would have caused them to link in round $k-1$, and hence they would not be passive. ◀

We measure progress in the algorithm using a *potential function*. It is like that of Awerbuch and Shiloach, but modified to guarantee that the total potential never increases, and to give passive trees, which can linger indefinitely, a potential of zero.

▶ **Definition 23.** The *potential* $\phi_k(T)$ of a tree T in round $k \geq 2$ is

$$\phi_k(T) = \begin{cases} 0 & \text{if } T \text{ is passive in round } k \\ 3 & \text{if } T \text{ is active and flat in round } k \\ h+1 & \text{if } T \text{ has height } h \geq 2 \text{ in round } k \end{cases}$$

The *total potential* in round $k \geq 2$ is the sum of the potentials of all the trees in the round.

We shall obtain a bound on the total potential that decreases by a constant factor each round (other than the first two). Theorem 19 is immediate from such a bound. It suffices to consider each active tree individually, since algorithm R is monotone and the total potential in a round is the sum of the potentials of the trees in the round. We shall track an active tree T backward through the rounds in order to see what earlier trees were combined to form it. We show that these earlier trees had enough potential to give the desired potential decrease.

▶ **Definition 24.** Let T be a tree that is active in round $k > 2$. For i such that $2 \leq i < k$, the *constituent trees* of T in round i are those in round i whose vertices are in T. The *potential* $\Phi_i(T)$ of T in round i is the sum of the potentials of the constituent trees of T in round i. In particular, $\Phi_k(T) = \phi_k(T)$.

▶ **Lemma 25.** *Let T be active in round $k > 2$. For i such that $2 \leq i < k$, the constituent trees of T in round i include at least one active tree.*

Proof. By induction on i for decreasing i. The lemma holds for $i = k$ by assumption and for $i = 2$ since all trees in round 2 are active. Suppose it holds for $i > 3$. If the constituent trees of T in round $i-1$ are all passive, the CONNECT in round i changes none of these trees by Lemma 22. Since all these trees are flat, the SHORTCUT in round i also changes none of them. Thus all these trees are passive in round i, contradicting the induction hypothesis. ◀

▶ **Lemma 26.** *Let T be active in round $k > 2$. Then $\Phi_{k-1}(T) \geq \Phi_k(T)$, and if $\Phi_k(T) \geq 5$ then $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$.*

Proof. Let h be the sum of the heights of the constituent trees of T in round $k-1$, let t be the number of these constituent trees that are active in round $k-1$, and let f be the number of these trees that are active and flat. Then $\Phi_{k-1}(T) = h + t + f$. Consider the tree T' formed from the constituent trees of T by the CONNECT in round k. The SHORTCUT in round k transforms T' into T. By Lemma 22, along any path in T', there cannot be consecutive vertices from two different passive trees. By Lemma 6, all leaves of constituent trees are leaves of T'. It follows that T' has height at most $h + t + 1$: the active constituent trees contribute at most h vertices to a longest path, at most $t + 1$ roots of passive trees are on the path, at most one leaf of some constituent tree is on the path, and the path has length one less than its number of vertices. Thus $\Phi_k(T) \leq \lceil (h+t+1)/2 \rceil + 1$ if $h+t > 2$, $\Phi_k(T) = 3$ if $h+t = 2$.

We prove the lemma by induction on $h+t$, which is at least 2, since at least one of the constituent trees must be active by Lemma 25. If $h+t=2$, there is one active constituent tree, and it is flat, so $f=1$. If $h+t=3$, there is one active constituent tree, and it is not flat. In both these cases, $\Phi_{k-1}(T) = 3$ and $\Phi_k(T) = 3$, so the lemma holds. Suppose $h+t > 3$. Then $\Phi_k(T) \leq \lceil (h+t+1)/2 \rceil + 1$. If $h+t=4$, $\Phi_{k-1}(T) \geq 4$ and $\Phi_k(T) \leq 4$; if $h+t=5$, $\Phi_{k-1}(T) \geq 5$ and $\Phi_k(T) \leq 4$; if $h+t=6$, $\Phi_{k-1}(T) \geq 6$ and $\Phi_k(T) \leq 5$. Thus the lemma holds in these cases. Each increase of $h+t$ by two increases the lower bound on $\Phi_{k-1}(T)$ by two and increases the upper bound on $\Phi_k(T)$ by one, which preserves the inequality $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$, so the lemma holds for all $h+t$ by induction: if $\Phi_k(T) \geq 5$, it must be the case that $h+t \geq 6$. ◀

▶ **Corollary 27.** *Let T be an active tree of height at least four in round $k \geq 3$. Then $\Phi_{k-1}(T) \geq (6/5)\Phi_k(T)$.*

Corollary 27 gives us the desired potential drop for any active tree of height at least four. It remains to consider active trees of heights one, two, and three. Since active trees of height one and two have the same potential, namely three, we shall handle these cases together. This gives us two cases: height at most two, and height three. In order to obtain the desired potential drop, we need to look backward up to two rounds in the case of height three, up to five in the case of height at most two.

▶ **Lemma 28.** *Let T be an active tree of height 3 in round $k \geq 3$. Let $j = \max\{2, k-2\}$. Then $\Phi_j(T) \geq (5/4)\Phi_k(T)$.*

Proof. If the constituent trees of T in round $k-1$, or in round $k-2$ if $k > 3$, include at least two active trees (of total potential at least six) or one active tree of height at least four (of potential at least five), then the lemma holds by Lemma 26, since $\Phi_k(T) = 4$. If not, by Lemma 25 the constituent trees of T in round $k-1$, and in round $k-2$ if $k > 3$, include exactly one active tree, of height exactly three. In this case there must be at least two constituent trees of T in round $k-1$, and in round $k-2$ if $k > 3$, since the tree formed from these constituent trees by the CONNECT in the next round must have height at least five, in order for the SHORTCUT in this round to produce a tree of height three. But this implies $k > 4$, since all trees in round 2 are active.

We conclude that the lemma holds except in the following case: $k > 4$ and the constituent trees of T in rounds $k-2$ and $k-1$ each include exactly one active tree, of height exactly three. Let T_2 and T_1, respectively, be the active constituent trees of T in rounds $k-2$ and $k-1$. Let T_1' and T', respectively, be the trees formed from the constituent trees of T by the CONNECT steps in rounds $k-1$ and k. The SHORTCUT in round $k-1$ transforms T_1' into T_1, and the SHORTCUT in round k transforms T' into T. Both T_1' and T' have height exactly five. The passive constituent trees of T in round $k-1$ are a proper subset of those in round $k-2$; specifically, those that are not combined with T_2 to form T_1' in the CONNECT of round $k-1$. By Lemma 22, no edge connects two passive constituent trees of T in round $k-2$.

Call a passive constituent tree of T in round $k-2$ *primary* if it has an edge connecting it to the root of T_2 or to a child of the root of T_2. Since T_1' has height five, its root must be different from that of T_2: if not, T_1' would have height at most four by the argument in the proof of Lemma 26. If the root of T_1' is different from that of T_2, its root must be the minimum of the roots of the primary trees of T in round $k-2$. The root of T' must also be different from the root of T_1 (which is the same as the root of T'_1). But the root of T' cannot be the root of one of the primary passive constituent trees of T in round $k-2$, since none of these roots are smaller than the root of T'_1. Nor can it be the root of one of the non-primary passive constituent trees of T in round $k-2$, since such a tree has no edge connecting it to the root or a child of the root of T_2, implying that it has no edge connecting it to the root, a child of the root, or a grandchild of the root of T'_1, further implying that it has no edge connecting it to the root or a child of the root of T_1, making it impossible for the CONNECT in round k to link T_1 to it. Thus this case is impossible. ◀

The analysis of an active tree of height at most two is like that in Lemma 28 but more complicated: we must consider *all* the passive constituent trees in the first relevant round, not just the primary ones. At a high level the argument is the same: if in one of the four rounds preceding round k there are two active constituent trees, or one of height at least three, we obtain the desired potential drop; if not, the algorithm eventually runs out of passive trees to link to the one active tree, which is impossible. We proceed with the details.

▶ **Lemma 29.** *Let T be an active tree of height at most two in round $k \geq 4$. Let $j = \max\{2, k-5\}$. Then $\Phi_j(T) \geq (4/3)\Phi_k(T)$.*

Proof. If for some i such that $j \leq i < k$ the constituent trees of T in round i include at least two active trees (of total potential at least six) or one active tree of height at least three (of potential at least four), then the lemma holds by Lemma 26, since $\Phi_k(T) = 3$. If not, by Lemma 25 the constituent trees of T in each round from j to $k-1$ include exactly one active tree, of height at most two. In this case the passive constituent trees of T in round i are a (not necessarily proper) subset of those in round i', for $j \leq i < i' < k$. Furthermore there must be at least two constituent trees of T in round $k-2$, and hence in round j, since otherwise the active constituent tree of T in round $k-1$ would be flat (because the CONNECT in round $k-1$ does nothing), and this tree would be equal to T, making T passive, a contradiction. Since all trees in round 2 are active, this makes the lemma true if $k \leq 7$.

We conclude that the lemma holds except in the following case: $k > 7$ and the constituent trees of T in each round from j to $k-1$ inclusive each include exactly one active tree, of height at most two. For each round i from j to k inclusive, let T_i be the active constituent tree of T in round i (so $T_k = T$), and for each round i from $j+1$ to k inclusive, let T'_i be the tree formed from the constituent trees of T by the CONNECT in round i. For $j < i \leq k$, the SHORTCUT in round i transforms T'_i into T_i.

By Lemma 22, no edge connects two passive constituent trees of T in round j, nor in any later round. Call a passive constituent tree of T in round j *primary* if it has an edge connecting it to the root of T_j or to a child of the root of T_j, *secondary* otherwise. Every passive constituent tree of T in round j must have an edge connecting it to T_j, since it does not have an edge connecting it to another passive constituent tree, and some CONNECT must link it with an active tree by the end of round k. Since T_j has height at most two, each secondary constituent tree of T in round j has an edge connecting it with a grandchild of the root of T_j.

We consider two cases: the roots of T_j and T'_{j+1} are the same, or they are different. (See Figure 2.) In the former case, the roots of all primary constituent trees of T in round j are greater than the root of T_j, and the CONNECT in round $j+1$ makes all of them children of the root of T_j. In the latter case, the root of T'_{j+1} is the minimum of the roots of the primary constituent trees of T in round j, and each such tree other than the one of minimum root is linked to T_j in round $j+1$ or to T_{j+1} in round $j+2$.

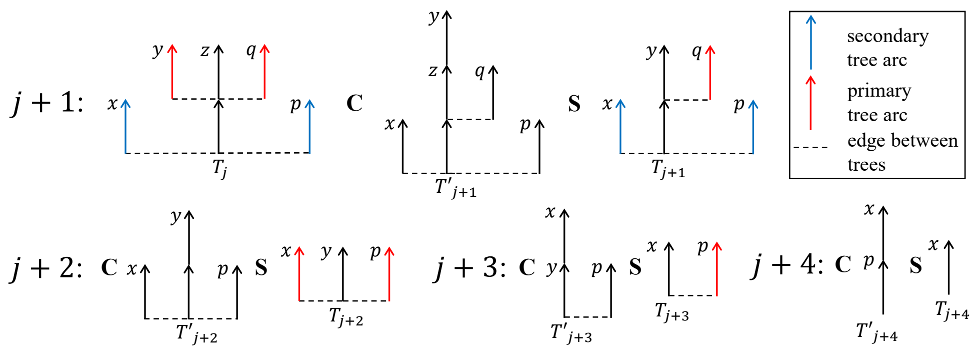

Figure 2 Worst-case illustration for the case of height at most two from round $j+1$ to round $j+4$. Only necessary vertices, arcs, and edges are shown. Vacant before **C** (CONNECT) or **S** (SHORTCUT) denotes the same forest as in the previous step. The vertices (roots) satisfy $x < p < y < q < z$.

Now consider the secondary constituent trees of T in round j. If the roots of T_j and T'_{j+1} are the same, then after the SHORTCUT in round $j+1$ each secondary constituent tree whose vertices are not in T_{j+1} has an edge connecting it with a child of the root of T_{j+1}. By the argument in the preceding paragraph, each such tree will be linked with T_{j+1} in round $j+2$ or with T_{j+2} in round $j+3$. If the roots of T_j and T'_{j+1} are different, none of the secondary constituent trees of T in round j has an edge connecting it with the root or a child of the root of T_{j+1} at the end of round $j+1$. In this case, the roots of T_{j+1} and T_{j+2} must be the same, so after the SHORTCUT in round $j+2$ each secondary constituent tree whose vertices are not in T_{j+2} has an edge connecting it with a child of the root of T_{j+2}. Each such tree will be linked with T_{j+2} in round $j+3$ or with T_{j+3} in round $j+4$. Furthermore the roots of T_{j+3} and T_{j+4} must be the same.

It follows that there is only one constituent tree of T in round $j+4$, and this tree is flat. But this tree must be T, making T passive in round $j+5=k$, a contradiction. Thus this case is impossible. ◀

Having covered all cases, we are ready to put them together into a proof of Theorem 19. Let $a = (4/3)^{1/5} \simeq 1.06$. We denote the number of vertices in a tree T by $|T|$.

▶ **Lemma 30.** *Let T be an active tree in round $k \geq 2$. Then $\Phi_k(T) \leq 2|T|/a^{k-2}$.*

Proof. By induction on k. The lemma holds for $k=2$ and $k=3$ since each active tree T after round two has at least two vertices and potential at most $|T|+1$, which is at most $2|T|$ in round two and at most $2|T|/a$ in round three since $a < 4/3$.

To prove the the lemma for $k \geq 4$, suppose the lemma holds for all k' such that $2 \leq k' < k$. We consider three cases. If the height of T exceeds three, then $\Phi_k(T) \leq (5/6)\Phi_{k-1}(T) \leq (5/6)2|T|/a^{k-3}$ by Corollary 27, the induction hypothesis, and the linearity of the potential function. The lemma holds for T since $a < 6/5$. If the height of T equals three, then $\Phi_k(T) \leq (4/5)\Phi_{k-2}(T) \leq (4/5)2|T|/a^{k-4}$ by Lemma 28, the induction hypothesis, and the linearity of the potential function. The lemma holds for T since $a^2 < 5/4$. If the height of T is at most two, then $\Phi_k(T) \leq (3/4)\Phi_j(T)$ by Lemma 26 and Lemma 29, where $j = \max\{2, k-5\}$. By the induction hypothesis and the linearity of the potential function, $\Phi_k(T) \leq (3/4)2|T|/a^{j-2}$. The lemma holds for T since $k-j \leq 5$ and $a = (4/3)^{1/5}$. ◀

Proof of Theorem 19. By Lemma 30, if k is such that $2n/a^{k-2} \leq 2$, then no tree can be active. This inequality is equivalent to $k \geq \lg n/\lg a + 2$. Every round except the last one has at least one active tree. Thus the number of rounds is at most $\lceil \lg n/\lg a \rceil + 2$. ◀

6 Remarks

We have presented several very simple label-update algorithms to compute connected components concurrently. We have shown that two of our algorithms, algorithms R and RA, take $O(\lg n)$ steps and $O(m \lg n)$ total messages, and the others take $O(\lg^2 n)$ steps and $O(m \lg^2 n)$ total messages. Crucial to our algorithms is the use of minimum-value resolution of write conflicts. We do not have tight efficiency analyses for our algorithms other than R and RA, and we leave obtaining such analyses as an open problem. We think our algorithms are simple enough to merit experimental study, but we leave this for future work.

Our analysis of algorithm R is novel in that it studies what happens over several rounds. Previous algorithms were designed so that they could be analyzed one round at a time. We have sacrificed simplicity in the analysis for simplicity in the algorithm.

We have ignored message contention. We think it is most fruitful to handle such contention separately from the underlying algorithm. Dealing with contention is a topic for future work, as is reducing the amount of synchronization required and developing incremental and batch-update algorithms. We think that concurrent algorithms for disjoint set union [11], the incremental version of the connected components problem, may be adaptable to give good asynchronous concurrent algorithms for the connected components problem with batch edge additions.

Another interesting extension of the connected components problem is to construct a forest of spanning trees of the components. It is easy to modify algorithm R to do this: when an edge causes a root to become a child, add the corresponding original edge to the spanning forest. It is not so obvious how to extend algorithms such as A, P, and E that move subtrees. This is perhaps another reason to favor algorithm R in practice.

References

1. Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685, 2018.
2. Baruch Awerbuch and Yossi Shiloach. New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM. *IEEE Trans. Computers*, 36(10):1258–1263, 1987.
3. Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017.
4. Paul Burkhardt. Graph connectivity in log-diameter steps using label propagation. *CoRR*, 2018. `arXiv:1808.06705`.
5. Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes. *SIAM J. Comput.*, 15(1):87–97, 1986.
6. Steve Goddard, Subodh Kumar, and Jan F. Prins. Connected components algorithms for mesh-connected parallel computers. In *Parallel Algorithms, Proceedings of a DIMACS Workshop, Brunswick, New Jersey, USA, October 17-18, 1994*, pages 43–58, 1994.
7. John Greiner. A Comparison of Parallel Algorithms for Connected Components. In *SPAA*, pages 16–25, 1994.
8. Shay Halperin and Uri Zwick. An Optimal Randomised Logarithmic Time Connectivity Algorithm for the EREW PRAM. *J. Comput. Syst. Sci.*, 53(3):395–416, 1996.
9. Shay Halperin and Uri Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests. *Journal of Algorithms*, 39(1):1–46, 2001.

10 Tsan-Sheng Hsu, Vijaya Ramachandran, and Nathaniel Dean. Parallel implementation of algorithms for finding connected components in graphs. *Parallel Algorithms: Third DIMACS Implementation Challenge, October 17-19, 1994*, 30:20, 1997.

11 Siddhartha V. Jayanti and Robert E. Tarjan. A Randomized Concurrent Algorithm for Disjoint Set Union. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 75–82, 2016.

12 Donald B. Johnson and Panagiotis Takis Metaxas. A Parallel Algorithm for Computing Minimum Spanning Trees. *J. Algorithms*, 19(3):383–401, 1995.

13 Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 135–146, 2010.

14 Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! But at what COST? In *15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015*, 2015.

15 Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 50–61, 2013.

16 John H Reif. Optimal Parallel Algorithms for Graph Connectivity. Technical report, Harvard University Cambridge, MA, Aiken Computation Lab, 1984.

17 Yossi Shiloach and Uzi Vishkin. An O(log n) Parallel Connectivity Algorithm. *J. Algorithms*, 3(1):57–67, 1982.

18 Stergios Stergiou, Dipen Rughwani, and Kostas Tsioutsiouliklis. Shortcutting Label Propagation for Distributed Connected Components. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 540–546, 2018.

19 Robert Endre Tarjan and Jan van Leeuwen. Worst-case Analysis of Set Union Algorithms. *J. ACM*, 31(2):245–281, 1984.

20 Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, 1990.

21 Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel Algorithms for Graph Connectivity Problems with Performance Guarantees. *PVLDB*, 7(14):1821–1832, 2014.