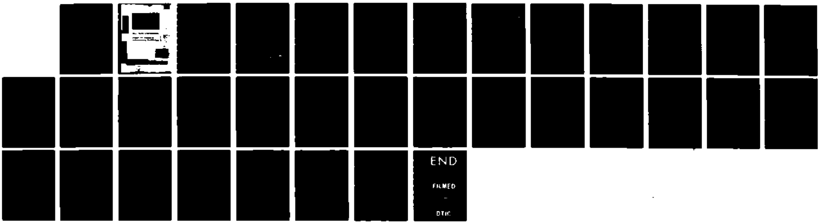
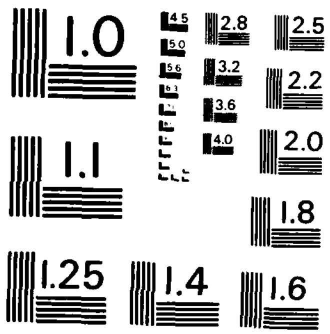
AD-A161 791 OPTIMAL PARALLEL ALGORITHMS FOR INTERGER SORTING AND GRAPH CONNECTIVITY(U) HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB J H REIF 1985 TR-08-85 1/1

UNCLASSIFIED N00014-80-C-0647 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A161 791

(12)

OPTIMAL PARALLEL ALGORITHMS FOR
INTEGER SORTING AND GRAPH CONNECTIVITY

John H. Reif

TR-08-85

Harvard University

Center for Research in Computing Technology

Aiken Computation Laboratory
33 Oxford Street
Cambridge, Massachusetts 02138

12

OPTIMAL PARALLEL ALGORITHMS FOR
INTEGER SORTING AND GRAPH CONNECTIVITY

John H. Reif

TR-08-85

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) unclassified

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

AN OPTIMAL PARALLEL ALGORITHM FOR INTEGER SORTING

5. TYPE OF REPORT & PERIOD COVERED

Technical Report

6. PERFORMING ORG. REPORT NUMBER

TR-08-85

7. AUTHOR(s)

John H. Reif

8. CONTRACT OR GRANT NUMBER(s)

N00014-80-C-0647

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Harvard University
Cambridge, MA 02138

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

12. REPORT DATE

1985

13. NUMBER OF PAGES

10

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Same as above

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

~~Same as above~~

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

unlimited

18. SUPPLEMENTARY NOTES

unlimited

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

randomized computation, parallel computation, optimal algorithms, sorting, P-RAM

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse side.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

0. ABSTRACT

We assume a parallel RAM model which allows both concurrent writes and concurrent reads of global memory. Our algorithms are *randomized:* each processor is allowed an independent random number generator. However our stated resource bounds hold for worst case input with overwhelming likelihood as the input size grows.

We give a new parallel algorithm for integer sorting where the integer keys are restricted to at most polynomial magnitude. Our algorithm costs only logarithmic time and is the first known where the product of the time and processor bounds are bounded by a linear function of the input size. These simultaneous resource bounds are asymptotically optimal. All previous known parallel sorting algorithms required at least a linear number of processors to achieve logarithmic time bounds, and hence were nonoptimal by at least a logarithmic factor.

A large literature exists on efficient sequential RAM algorithms with time bound linear in the input size. Many of these algorithms require sorts to be done on integers of at most polynomial magnitude. For example, the depth first search algorithms of [Tarjan, 72] and [Hopcroft and Tarjan, 73] require the edges (which may be considered integers) to be sorted into adjacency lists. A $\Omega(n \log n)$ comparison sort such as QUICK-SORT or HEAP-SORT would not be sufficiently efficient for these applications. Instead, the BUCKET-SORT (see [Aho, Hopcroft, and Ullman, 74]) is used to sort in linear time. The BUCKET-SORT algorithm is sufficiently simple and elegant so that it is widely used in practice.

The goal of this paper is to develop an efficient and possibly practical integer sorting algorithm for a parallel RAM model, but we will utilize quite different techniques--such as randomization.

Optimal Parallel Algorithms for Integer Sorting and Graph Connectivity

John H. Reif*

Aiken Computation Lab.
Harvard University
Cambridge, Massachusetts

March, 1985

0. ABSTRACT

~~We~~ give new parallel algorithms for integer sorting and undirected graph connectivity problems such as connected components and spanning forest. ~~Our~~ algorithms cost only logarithmic time and are the first known that are *optimal:* the product of their time and processor bounds are bounded by a linear function of the input size. All previous known parallel algorithms for these problems required at least a linear number of processors to achieve logarithmic time bounds, and hence were nonoptimal by at least a logarithmic factor.

We assume a parallel RAM model which allows both concurrent writes and concurrent reads of global memory. ~~Our~~ algorithms are *randomized*; each processor is allowed an independent random number generator; however our stated resource bounds hold for worst case input with overwhelming likelihood as the input size grows.

*This work was supported by Office of Naval Research Contract N00014-80-C-0647.

1. INTRODUCTION

1.1 Optimal Sequential RAM Algorithms

A large literature exists on efficient sequential algorithms with time bound linear in the input size. This literature generally assumes the sequential Random Access Machine Model (RAM); for an introduction to this literature see [Aho, Hopcroft, and Ullman, 74]. Perhaps the most influential works done in this area were the graph algorithms of [Tarjan, 72] and [Hopcroft and Tarjan, 73]. These efficient sequential algorithms relied on linear time algorithms for (1) *bucket sort*, and (2) *depth first search.*

This linear time bucket sort was essential to depth first search since the edges must be sorted into adjacency lists. By ingenious use of both (1) and (2), Hopcroft and Tarjan derived linear time algorithms for graph problems such as connected components, spanning forest, and biconnected components.

The goal of this paper is to achieve similar results (i.e., optimal algorithms) for a parallel RAM model, but we will utilize quite different techniques (i.e., randomization).

1.2 Known Parallel RAM Algorithms

The performance of a parallel algorithm can be specified by bounds on its principal resources: processors and time. We generally let P denote the processor bound and T denote the time bound. For most nontrivial problems such as sorting and the above graph problems, the product $P \cdot T$ is lower bounded by at least a constant times the input size. Thus for these problems, we consider a parallel algorithm to be *optimal* if $P \cdot T = O(\text{input size})$. For example, given a graph of n vertices and m edges, a parallel graph connectivity algorithm is optimal if $P \cdot T = O(n+m)$. Of course, if we have an optimal algorithm with any processor bound P, then we also have (by the obvious processor simulation) an optimal algorithm for any processor bound P', where $P \geq P' \geq 1$. Hence an optimal algorithm may also be useful in practical situations where we have a limited number of processors.

We assume a *parallel RAM* model of [Shiloach and Viskin, 81]. The processors are synchronous, and each is a unit cost sequential RAM which in a single step may either read or write into a memory cell or register, or perform an arithmetic operation on an integer. Each memory cell and register may contain at most a logarithmic number of bits in the input size. This parallel RAM model allows multiple reads at a single memory cell and also allows multiple writes at a single memory cell, where multiple writes are allowed to be resolved arbitrarily. This model is known as the CRCW parallel RAM and is quite robust, see [Kucera, 82] for its relation to other parallel machine models. In addition we allow each processor an independent random number generator.

There are a number of known algorithms for sorting in logarithmic time using a linear number of processors; for example [Reischuk, 82] gives a randomized parallel RAM algorithm (which unfortunately requires memory cells of $n^{1/2}$ bits each). [Reif and Valiant, 83] give a randomized parallel algorithm (which has only moderate constant bounds and requires memory cells of $O(\log n)$ bits each), and [Ajtai, Komlós, and Szemerédi 83; and Leighton, 84] give a deterministic parallel algorithm. This last result of [Leighton, 84] appeared to finally settle the problem of parallel sorting since $PT = \Omega(n \log n)$ is a known lower bound in the case of comparison sorting. However, these lower bounds on PT need not hold for *integer* sorting: sorting n integers on the range $[n]$* (note that the restriction to the range $[n]$ is natural, since RAM memory cells can only contain numbers with at most a logarithmic number of bits.) Integer sorting is all that is required for most practical applications of interest, for example for putting a list of edges into adjacency list representative by sorting the edges by the vertices from which they depart. On the other hand, an optimal integer sort is essential in the derivation of any optimal parallel graph algorithm which requires the edges to be put in adjacency list representation.

---

*Note throughout this paper, we let $[n]$ denote $\{1,\ldots,n\}$.

Previously $T = O(\log n)$ time bounds and simultaneous $P = n+m$ processor bounds have been given for connected components [Shiloach and Vishkin,83] and spanning trees [Awerbuch and Shiloach, 83] of graphs with n vertices and m edges. All these previous algorithms had a $PT = \Omega((n+m)\log n)$ bound, which was a logarithmic factor more resources than optimal for logarithmic time bounds. [Tarjan and Vishkin, 83] pose as an open problem to find optimal parallel graph algorithms.

In fact no optimal graph searching method has been proposed for parallel RAM, for any sublinear time bounds, except in the special case where the graph is extremely dense (i.e., $m = \Omega(n^2)$). [Chin, Lam and Chen, 82] and [Vishkin, 81], both give $O(\log n)^2$ time connectivity algorithms requiring $(n^2+m)/(\log n)^2$ processors, which is optimal only if $m = \Omega(n^2)$.

Vishkin conjectured that randomized techniques would be needed to get optimal parallel graph connectivity algorithms. Indeed the literature contains some interesting attempts to use randomization to derive optimal parallel algorithms for graph problems. For example [Vishkin, 84] recently gave a randomized algorithm for finding the number of successors on a linear list which used an optimal number of processors with an almost logarithmic time bound. (However, Vishkin's algorithm assumed an oracle which provided a random permutation, but he provided no efficient method for parallel construction of random permutations.) Also [Reif, 84] gave a randomized parallel graph algorithm which had optimal processor bounds only for graphs with $m \geq n(\log n)^2$ edges.

1.3 Our optimal Parallel RAM Algorithms

Our main results are optimal randomized parallel RAM algorithms:

(1) $\tilde{O}(\log n)$ time, $n/\log n$ processor algorithms for integer sorting

(2) $\tilde{O}(\log n)$ time, $(m+n)/\log n$ processor algorithms for connected components and spanning forests for any graph of n vertices and m edges.

Here $\tilde{O}$ denotes that the upper bound holds within a constant factor with overwhelming likelihood, for the *worst case input*. In particular, we let $T(n) = \tilde{O}(f(n))$ denote $\exists c\ \forall\ \alpha \geq 1$, $\forall$ sufficiently large n, $T(n) \leq c\alpha f(n)$ holds with probability at least $1 - 1/n^{\alpha}$.

Our integer sorting algorithm is quite easy to implement and may be of some practical use, since it has very moderate constant factors.

1.4 Organization of This Paper

In Section 2, we give a known optimal algorithm for parallel prefix computation which will be of some use in devising our optimal parallel algorithms.

In Section 3, we give our optimal parallel algorithm for integer sorting, which achieves its efficiency by some interesting new randomization techniques. As an immediate consequence, (see Appendix A3) we get an optimal parallel algorithm for computing a random permutation.

In Section 4 (and Appendix A4) we give our algorithm for graph connectivity. It is derived in stages where we consider graphs of decreasing density. We first give a simple logarithmic time algorithm called RANDOM-MATE, which is nonoptimal, but utilizes randomization in an essential and new way. We next modify this algorithm so that it is optimal for graphs of n vertices with at least $m \geq n(\log n)^2$ edges. Then we give efficient parallel reductions from various cases of sparse graphs to the case $m \geq n(\log n)^2$.

In the Appendix A1 we give some useful upper bounds for the tails of various probability distributions which arise in the analysis of our algorithms.

In a separate paper we give applications of our optimal parallel graph connectivity algorithm to finding Euler cycles, biconnected components, and minimum spanning trees.

2. PARALLEL PREFIX COMPUTATION

2.1 Prefix Circuits

Let D be a domain and let $\circ$ be an associative operation which takes $O(1)$ sequential time over this domain. The *prefix computation problem* is defined as follows:

<u>input</u> $X(1),\ldots,X(n) \in D$

<u>output</u> $X(1), X(1) \circ X(2), \ldots, X(1) \circ \ldots \circ X(n)$.

[Ladner and Fischer, 80] show prefix computation can be done by a circuit of size n and depth $O(\log n)$.

Known techniques attributed to Brent, give the following processor improvement:

LEMMA 2.1. *Prefix computation can be done in time* $O(\log n)$ *using* $n/\log n$ *P-RAM processors.*

The *prefix sum computation problem* is defined as follows: Given input integers $X(1),\ldots,X(n) \in [n]$, output the vector $\text{PREFIX-SUM}(X) = (Y(0), Y(1),\ldots,Y(n))$ where $Y(0) = 0$ and $Y(i) = \Sigma_{j \leq i} X(j)$ for $i \in [n]$. By Lemma 2.1, we can do this computation in time $O(\log n)$ using $n/\log n$ processors.

3. AN OPTIMAL PARALLEL SORTING ALGORITHM

3.1 Known Sorting Algorithms

The *integer sorting problem* of size n is defined:

input keys $k_1,\ldots,k_n \in [n]$

output permutation $\sigma = (\sigma(1),\ldots,\sigma(n))$ such that $k_{\sigma(1)} \leq \ldots \leq k_{\sigma(n)}$.

The input keys $k_1,\ldots,k_n$ are not necessarily distinct. By use of the well known and quite practical BUCKET-SORT algorithm [Aho, Hopcroft, and Ullman, 74],

LEMMA 3.1. *Integer sorting can be done in time* $O(n)$ *by a deterministic sequential RAM.*

Any comparison based sort requires $PT = \Omega(n \log n)$, and the best known parallel sorts actually achieve these bounds. In particular, [Reif and Valiant, 83] show

LEMMA 3.2. n *keys can be sorted in time* $\tilde{O}(\log n)$ *using* n *processors in a constant degree network.*

This algorithm uses memory cells of $O(\log n)$ bits. It can also be implemented by the randomized P-RAM model. In addition, [Ajtai, Komlós, and Szemerédi,83], [Leighton, 84] give a deterministic sorting network which takes $O(\log n)$ time with $O(n)$ processors. In the following, we prove:

THEOREM 3.1. *Integer sort can be done in time* $\tilde{O}(\log n)$ *using* $n/\log n$ *P-RAM processors.*

We will achieve $PT = \tilde{O}(n)$ for integer sorting, making essential use of the fact that the input keys $k_1, \ldots, k_n$ are integers in $[n]$ as in the case of all our graph applications. We would be quite surprised if any purely deterministic methods yield $PT = O(n)$ for parallel integer sort in the case of time bounds $T = O(\log n)$. Although we will use deterministic methods to solve some restricted integer sorting problems, (see Lemmas 3.4 and 3.5 below) our optimal parallel algorithm for the general integer sorting problem requires some interesting, new use of randomization techniques (see Lemmas 3.6 and 3.7).

3.2 Easy Integer Sorting Problems

Given a sequence of keys $k_1, \ldots, k_n \in [n]$ let the *key index sets* be $I(k) = \{i | k_i = k\}$ for each key value $k \in [n]$. We will assume $\log n$ divides n.

LEMMA 3.3. *Given* $I(1), \ldots, I(n)$, *we can sort* $k_1, \ldots, k_n$ *in* $O(\log n)$ *time using* $P = n/\log n$ *processors.*

Proof. See Appendix A3.

A sorting algorithm is *stable* if given $k_1, \ldots, k_n$, the algorithm outputs a permuation σ of $(1, \ldots, n)$ where $\forall i, j \in [n]$ if $k_i = k_j$ and $i < j$ then $\sigma(i) < \sigma(j)$.

LEMMA 3.4. *A stable sort of* n *keys* $k_1, \ldots, k_n \in [\log n]$ *can be computed in* $O(\log n)$ *time using* $P = n/\log n$ *processors.*

Proof. See Appendix A3.

LEMMA 3.5. n *keys* $k_1, \ldots, k_n \in [(\log n)^2]$ *can be sorted in* $O(\log n)$ *time using* $P = n/\log n$ *processors.*

Proof. See Appendix A3.

Note: We can similarly extend Lemma 3.5 to apply to key values in $[(\log n)^{O(1)}]$.

3.3 Randomized Sampling and Sorting in Key Domain $[n/(\log n)^2]$

In the following subsection, we fix a key domain $[D]$ where $D = n/(\log n)^2$. (We assume $(\log n)^2$ divides n). Let the input keys be $k_1, \ldots, k_n \in [D]$ and their index sets be $I(k) = \{i | k_i = k\}$ for each key value $k \in [D]$.

LEMMA 3.6. *Given as input* $k_1,\ldots,k_n \in [D]$, *we can compute* $N(1),\ldots,N(D)$ *in* $\tilde{O}(\log n)$ *time using* $P = n/\log n$ *processors, such that* $\Sigma_{k\in[D]} N(k) \leq O(n)$ *and furthermore with high likelihood (in fact with probability* $\geq 1 - 1/n^{\alpha}$ *for any given* $\alpha \geq 1$) $N(k) \geq |I(k)|$ *for each* $k \in [D]$.

As proof, we execute the following randomized sampling algorithm

<u>Step 1</u> <u>for</u> each processor $\pi \in [P]$ in parallel <u>do</u>

<u>do</u> choose a random $s_\pi \in [n]$ <u>od</u>

$S \leftarrow \{s_1,\ldots,s_P\}$

<u>Comment</u>. Here we randomly choose a set $S \subseteq [n]$ of P key indices.

<u>Step 2</u> Sort $k_{s_1},\ldots,k_{s_P}$ and compute index set $I_S(k) = \{i \in S | k_i = k\}$ for each key value $k \in [D]$.

<u>Comment</u>. Applying Lemma 3.2, this sorting can be done by known parallel algorithms in $\tilde{O}(\log n)$ time using P processors.

<u>Step 3</u> <u>for</u> each $k \in [D]$ <u>do</u>

$N(k) \leftarrow d_0(\log n)(|I_S(k)| + \log n)$

Comment. d_0 is a constant to be determined in the probabilistic analysis.

<u>output</u> $N(1),\ldots,N(D)$.

See Appendix A3 for a proof of the probabilistic bounds given in Lemma 3.6.

<u>Lemma 3.7</u>. n *keys* $k_1,\ldots,k_n \in [D]$, (*where* $D = n/(\log n)^2$) *can be sorted in* $\tilde{O}(\log n)$ *time using* $P = n/\log n$ *processors*.

<u>Proof</u>. (We will actually use $O(P)$ processors, but we observe that we can then slow the computations down by a constant factor to reduce the processor bound to P.) Our randomized algorithm is given below.

<u>Step 1</u> Compute $N(1),\ldots,N(D)$ as defined in Lemma 3.6.

<u>Comment</u>. Here we use the random sampling algorithm of Lemma 3.6.

<u>Step 2</u> $(\bar{N}(0),\ldots,\bar{N}(D)) \leftarrow \text{PREFIX-SUM}(N(1),\ldots,N(D))$

<u>Comment</u>. This prefix-sum computation is done by Lemma 2.1 in $O(\log n)$ time and $O(P)$ processors.

Step 3 for each key value $k \in [D]$

do $P_k \leftarrow \{\pi \mid \pi \in [D] \text{ or } \bar{N}(k-1) + D < \pi \leq \bar{N}(k) + D\}$. Using these P_k processors, construct a table $A_k = (A_k(1), A_k(2), \ldots, A_k(N(k)), A_k(N(k)+1)))$ and initialize each element of the table to be an empty list.

od

Step 4 for each $\pi \in [P]$ in parallel do

for each $t = 1, \ldots, \log n$ sequentially do

$i_\pi \leftarrow (\pi-1)\log n+t$

choose a random number $r_\pi \in [N(k)]$

attempt to add i_π to front of list $A_{k_{i_\pi}}(r_\pi)$

if successful (i.e., i_π is now in front of list $A_{k_{i_\pi}}(r_\pi)$)

then CONFLICT $(i_\pi) \leftarrow 0$ else CONFLICT $(i_\pi) \leftarrow 1$ if

od od

Comment. Each processor $\pi \in [P]$ is responsible for keys $k_{(\pi-1)\log n+1}, \ldots, k_{\pi \log n}$. The inner loop for $t = 1, \ldots, \log n$ is executed sequentially so as to minimize conflicts. In the t-th iteration of the inner loop, processor π attempts to add the index $i_\pi = (\pi-1)\log n+t$ of the key k_{i_π} to the front of list $A_{k_{i_\pi}}(r_{i_\pi})$ where r_π is a randomly chosen integer in $[N(k)]$. This may not be successful if some other processor π' simultaneously attempts to add some other index $i_{\pi'}$ to the front of list $A_{k_{i_\pi}}(r_{i_\pi})$. Only one addition to this list will succeed. But this conflict will only happen in the case $k_{i_{\pi'}} = k_{i_\pi}$ and π' makes the same unlucky choice of $r_{\pi'} = r_\pi$.

Claim 3.1. Let $n' = \Sigma_{i=1}^{n}$ CONFLICT(i). Then $n' \leq \tilde{O}(P)$. In particular, $\exists c \; \forall \alpha \geq 1$ $\text{Prob}(n' \leq \alpha\, cn/\log n) \geq 1 - 1/n^\alpha$.

Proof. See Appendix A3.

Step 5 $(u(0),\dots,u(n)) \leftarrow \text{PREFIX-SUM}(\text{CONFLICT}(1),\dots,\text{CONFLICT}(n))$

$n' \leftarrow u(n)$

for each $\pi \in [P]$ in parallel

do for each $t = 1,\dots,\log n$ sequentially

do $i_\pi \leftarrow (\pi-1)\log n + t$

if $\text{CONFLICT}(i_\pi)$ then $j_{u(i_\pi)} \leftarrow i_\pi$ fi

od od

Comment. $(j_1,\dots,j_{n'})$ is the list of indices j such that $\text{CONFLICT}(j) = 1$. Again, the prefix computations can be done by applying Lemma 2.1.

Step 6. Sort $k_{j_1},\dots,k_{j_{n'}}$ and for each key value $k \in [D]$ assign

$A_k(N(k)+1)) \leftarrow \{j_\ell \mid k = k_{j_\ell}\}$.

Comment. In $A_k(N(k)+1)$ we place the list $\{j_\ell \mid k = k_{j_\ell}\}$ of conflicted indices with key value k. Assuming $n' \leqslant O(P)$, this step can be done by known parallel sorting algorighms in time $\tilde{O}(\log n)$ using P processors.

Step 7. for each key value $k \in [D]$

do Construct table A'_k consisting of a list of all the elements of the lists $A_k(1)$, $A_k(2),\dots,A_k(N(k)),A_k(N(k)+1)$.

od

Comment. This is done in $O(\log n)$ time by careful use of the processor set P_k. In particular, we first compute $(a_k(0),\dots,a_k(N(k)+1)) \leftarrow \text{PREFIX-SUM}(|A_k(1)|,|A_k(2)|,\dots,|A_k(k))|,|A_k(N(k)+1)|)$. Note that $|A_k(i)| \leqslant d_0 \log n$ for each i. Hence for each $i = 1,\dots,N(k)+1$ in parallel we can place the elements of $A_k(i)$ into locations $A'_k(a_k(i-1)+1)),\dots,A'(a_k(i))$ using a single processor $\pi \in P_k$ with time $O(\log n)$.

Step 8. Compute a permutation σ of $(1,\dots,n)$ such that the elements of $A'_1,\dots,A'_D$ appear in order.

Comment. We apply here Lemma 3.3.

output. $\sigma = (\sigma(1),\dots,\sigma(n))$.

The total time for steps 1-8 is $\tilde{O}(\log n)$ using P processors. □

3.4. Summary of Our Parallel Sorting Algorithm

Finally, we prove Theorem 3.1, by combining the above techniques. (We again assume $(\log n)^2$ divides n.)

Input keys $k_1, \ldots, k_n \in [n]$

Step 1 Assign $k_i' = \lceil k_i/(\log n)^2 \rceil + 1$ and $k_i'' = k_i - (k_i' - 1)(\log n)^2 + 1$ for each $i \in [n]$

Comment. $k_1', \ldots, k_n' \in [D]$ where $D = n/(\log n)^2$ and $k_1'', \ldots, k_n'' \in [(\log n)^2]$

Step 2 Sort $k_1', \ldots, k_n' \in [D]$ resulting in index sets $I'(k) = \{i \mid k_i' = k\}$ for each key value $k \in [D]$

Comment. This is done by applying Lemma 3.7.

Step 3 Sort $\{k_i'' \mid i \in I'(k)\} \subseteq [(\log n)^2]$ yielding ordered list $L(k)$ of indices in $I'(k)$ for each key value $k \in [D]$

Comment. This is done by applying the stable sort of Lemma 3.5 to the ordered list of keys $I'(1) \ldots I'(D)$.

Step 4 Compute the permutation σ which orders the indices as $L(1), \ldots, L(D)$

Comment. Here we apply Lemma 3.3, σ satisfies $k_{\sigma(1)} \leq \ldots \leq k_{\sigma(n)}$

output σ

The Lemmas 3.2-3.7 and the appropriate use of prefix-sum computation (Lemma 2.1) imply that each step can be done in $\tilde{O}(\log n)$ using $P = n/\log n$ processors. □

3.5 Optimal Parallel Generation of a Random Permutation

COROLLARY 3.1. *A random permutation* σ *of* $(1, \ldots, n)$ *can be constructed in* $\tilde{O}(\log n)$ *time using* $P = n/\log n$ *P-RAM processors.*

Proof. See Appendix A3.

4. OPTIMAL PARALLEL GRAPH ALGORITHMS

Given a graph G, let CC(G) be the connected components of G. We prove in this section:

THEOREM 4.1. *For any graph* G *with* n *vertices and* m *edges we can compute* CC(G) *in* $\tilde{O}(\log n)$ *time using* $(m+n)/\log n$ *parallel RAM processors.*

(Note: Simple modifications of our algorithms also give a spanning forest of G within the same resource bounds.)

The proof of Theorem 4.1 will be separated into three cases of decreasing density of edges. In each case, we efficiently reduce the connected components problem to one for a denser graph. The density reductions use various randomized sampling techniques (see details in Appendix A4).

4.1 A New, But Nonoptimal Randomized Algorithm

We begin by describing a new randomized algorithm RANDOM-MATE for computing CC(G) of $G=(V,E)$ with n vertices $V=\{1,\ldots,n\}$ and m edges E. We will associate a distinct processor with each vertex of V and each edge of E. This algorithm will be *nonoptimal* since it runs in $\tilde{O}(\log n)$ time using $n+m$ processors as did previous parallel graph connectivity algorithms [Shiloach and Vishkin, 83]. However, RANDOM-MATE has the advantage (not shared by the previous deterministic algorithms) that it can be modified to an optimal algorithm, as we prove in the Appendix A4.

Our randomized connectivity algorithm will be motivated by the following

LEMMA 4.1. (The Random Mating Lemma) *Let* $G=(V,E)$ *be any graph. Suppose for each vertex* $v \in V$, *we randomly, independently assign* $\mathrm{SEX}(v) \in \{\underline{\text{male}}, \underline{\text{female}}\}$. *Let vertex* v *be <u>active</u> if there exists at least one departing edge* $\{v,u\} \in E$ *where* $u \neq v$, *and let vertex* v *be <u>mated</u> if* $\mathrm{SEX}(v) = \underline{\text{male}}$ *and* $\mathrm{SEX}(u) = \underline{\text{female}}$ *for at least one edge* $\{v,u\} \in E$. *Then with probability* $1/2$ *the number of mated vertices is at least* $1/8$ *of all active vertices.*

<u>Proof</u>. See Appendix A4.

To represent collapsed subgraphs, we use an array R which we view as pointers mapping $V \to V$. Let the graph *collapsed by* R be defined $R(G) = (R(V),R(E))$ where $R(V) = \{R(v) \mid v \in V\}$ and $R(E) = \{(R(v),R(u)) \mid \{v,u\} \in E,\ R(v) \neq R(u)\}$. Each vertex $r \in R(V)$ is named a R-*root*. Our algorithm below (and the ones to follow) will always satisfy $R(R(v)) = R(v)$ for each $v \in V$. Hence the R pointers define a directed forest $(V,\{(v,R(v)) \mid v \in V - R(V)\})$. Each tree in this forest will be called a R-*tree*; it will have height ≤ 1 and will consist of a maximal set of vertices of V mapped to the same R-root.

Initially we set $R(v) = v$ for all $v \in V$. We will prove that at the end of the algorithm the vertices of R-trees are the connected components CC(G).

We execute the main loop $c_0 \log n$ times, where c_0 is a constant defined in the proof below. On each execution of <u>male</u>, we merge together connected subgraphs by a randomly assigning R-roots male or female with equal probability, and then letting each R-root assigned male to be merged into a R-root assigned female, if there is an edge between those corresponding subgraphs. Note that we can view this a mating process where each male may be mated and merged into at most one female but many males may merge into the same female.

It will be useful to define $D(E) = \{(v,u) \mid \{v,u\} \in E\} \cup \{(u,v) \mid \{v,u\} \in E\}$ to be the directed edges derived from E.

```
algorithm  RANDOM-MATE
    input graph  G = (V,E)  with  n = |V|  and  m = |E|.
    initialize for each  v ∈ V  in parallel do  R(v) ← v  od
    main loop:  for  t = 1,...,c_0 log n
                do
       assign sex:  for each  v ∈ V  in parallel do
                        if R(v) = v  then
                        comment  v  is currently a R-root
                        randomly assign  SEX(v) ∈ {male,female}
                        fi od
          merge:  for each  (v,u) ∈ D(E)  in parallel do MATE(v,u)
          collapse:  For each  v ∈ V  in parallel
                        do  R(v) ← R(R(v))
                        comment collapse the R-trees to depth
                  od  od
    output  R(1),...,R(n)
    Also we define
procedure MATE(v,u)
    if  SEX(R(v)) = male and  SEX(R(u)) = female
    then  R(R(v)) ← R(u) fi
    comment  attempt to mate male R-root  R(v)  with  female R-root  R(u).
```

Claim 4.1. The vertex set of each R-tree is always within a single connected component of $CC(G)$.

Proof. See Appendix A4.

Note RANDOM-MATE may have incorrect output if after $c_0 \log n$ iterations, there still exists an active R-root. But the main body can easily be altered to test if $\exists \{v,u\} \in E$ such that $R(v) \neq R(u)$ and if so, go back to the main loop. RANDOM-MATE then yields the following (nonoptimal) result:

LEMMA 4.2. *For any graph* G *with* n *vertices and* m *edges, we can compute* $CC(G)$ *in time* $\tilde{O}(\log n)$ *using* $m+n$ *processes.*

Proof. See Appendix A4.

4.2-4.4 Optimal Parallel Algorithms for Various Edge Densities

We hope our careful description of RANDOM-MATE has interested the reader enough to read the proof of Theorem 4.1 given in the Appendix. The proof is broken into three cases:

(1) $m \geqslant n(\log n)^2$

(2) $m \geqslant n(\log n)^{1/3}$

(3) $m \leqslant n(\log n)^{1/3}$

Cases (1) and (2) apply random sampling techniques and various modified and improved forms of RANDOM-MATE which use $(m+n)/\log n$ processors. Case (3) uses a variant of RANDOM-MATE with a randomized conflict resolution technique similar to the conflict resolution techniques used in our integer sorting algorithm. The details are found in Appendix A4.

ACKNOWLEDGEMENTS

The author thanks S. Rajasekaran and Paul Spirakis for a careful reading of this manuscript.

REFERENCES

Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

Angluin, D. and L.G. Valiant, "Fast Probabilistic Algorithms for Hamiltonian Paths and Matchings," *J. Comp. Syst. Sci.* 18 (1979), pp. 155-193.

Ajtai M., J. Komlós, and E. Szemerédi, "An O(n log n) Sorting Network," Proc. 15th Annual Symposium on the Theory of Computing, 1983, pp. 1-9.

Awerbuch, B. and Y. Shiloach, "New Connectivity and MSF Algorithms for Ultracomputer and PRAM," IEEE Conf. on Parallel Comput., 1983.

Batcher, K., "Sorting Networks and Their Applications," Spring Joint Computer Conf. 32, AFIPS Press, Montrale, N.J., pp. 307-314.

Chin, F.Y., J. Lam, and I. Chen, "Efficient Parallel Algorithms for Some Graph Problems," CACM, vol. 25, No. 9 (Sept. 1982), p. 659.

Chernoff, H., "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations," *Annals of Math. Statistics*, Vol. 23, 1952.

Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, New York, 1950.

Fitch, F.E., "Two Problems in Concrete Complexity: Cycle Detection and Parallel Prefix Computation," Ph.D. Thesis, Univ. of California, Berkeley, 1982.

Hirschberg, D.S., A.K. Chandra, and D.V. Sawata, "Computing Connected Components on Parallel Computers," CACM, Vol. 22 (1979), p. 461.

Hoeffding, W., "On the Distribution of the Number of Successes in Independent Trials," *Ann. of Math. Stat.* 27, (1956), 713-721.

Hopcroft, J.E. and R.E. Tarjan, "Efficient Algorithms for Graph Manipulation," *Comm. ACM* 16(6), (1973), 372-378.

Johnson, N.J. and S. Katz, *Discrete Distributions*, Houghton Mifflin Comp., Boston, MA, 1969.

Kucera, L., "Parallel Computation and Conflicts in Memory Access," *Information Processing Letters*, Vol. 14, No. 2, April 1982.

Kwan, S.C. and W.L. Ruzzo, "Adaptive Parallel Algorithms for Finding Minimum Spanning Trees," International Conference on Parallel Programming, 1984.

Leighton, T., "Tight Bounds on the Complexity of Parallel Sorting," 16th Symp. on Theory of Computing, Washington, D.C., 1984, pp. 71-80.

Ladner, R.E. and M.J. Fischer, "Parallel Prefix Computation," *J. Assoc. Computing Mech.*, Vol. 27, No. 4, Oct. 1980, pp. 831-838.

Nath, D. and S.N. Maheshwari, "Parallel Algorithms for the Connected Components and Minimal Spanning Tree Problems," *Inform. Proc. Letts.*, Vol. 14, No. 2, April 1982.

Rabin, M.O., "Probabilistic Algorithms," in: *Algorithms and Complexity*. J.F. Traub (ed.), Academic Press, New York, 1976.

Reif, J., "Symmetric Complementation," *J. of the ACM*, Vol. 31, No. 2, (1984), pp. 401-421

Reif, J., "On the Power of Probabilistic Choice in Synchronous Parallel Computations," *SIAM J. Computing*, Vol. 13, No. 1, (1984), pp. 46-56.

Reif, J.H. and J.D. Tyger, "Efficient Parallel Pseudo-Random Number Generation," Technical Report TR-07-84, Harvard University, 1984.

Reif, J.H., "Optimal Parallel Algorithms for Graph Connectivity," Tech. Rept. TR-08-84, Harvard University, Center for Computing Research, 1984.

Reif, J.H. and L.G. Valiant, "A Logarithmic Time Sort for Linear Size Networks," Proc. 15th Annual ACM Symp. on the Theory of Computing, pp. 10-16 (1983).

R. Reischuk, "A Fast Probabilistic Parallel Sorting Algorithm," Proc. of 22nd IEEE Symp. on Foundations of Computer Science (1981), 212-219.

Savage, C. and J. Ja'Ja', "Fast Efficient Parallel Algorithms for Some Graph Problems," *SIAM J. on Computing*, Vol. 10, N. 4 (Nov. 1981), p. 682.

Shiloach, Y. and U. Vishkin, "Finding the Maximum Merging and Sorting in a Parallel Computation Model," *J. of Algorithms*, Vol. 2, (1981), p. 88.

Shiloach, Y. and U. Vishkin, "An O(log n) Parallel Connectivity Algorithm," *J. of Algorithms*, Vol. 3 (1983), p. 57.

Tarjan, R.E., "Depth Forest Search and Linear Graph Algorithms," *SIAM J. Computing* 1(2), pp. 146-160 (1972).

Tarjan, R.E. and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithms," Technical Report, Courant Institute, New York University, New York, 1983.

Vishkin, U., "An Optimal Parallel Connectivity Algorithm," Tech. Report RC9149, IBM Watson Research Center, Yorktown Heights, New York, 1981, to appear in *Discrete Mathematics*.

Vishkin, U., "Randomized Speed-Ups in Parallel Computation," Proc. of the 16th Annual ACM Symp. on Theory of Computing, Washington, D.C., April 1984, pp. 230-239.

APPENDIX A1: Probabilistic Bounds

The randomized algorithms in the preceding sections are analyzed by applying the following probabilistic bounds on the tails of binomial and hypergeometric distributions (see also [Feller, 80]).

Let random variable X *upper bound* random variable Y (and Y *lower bound* X) if for all x such that $0 \leq x \leq 1$, $\mathrm{Prob}(X \leq x) \leq \mathrm{Prob}(Y \leq x)$.

A1.1 Binomial Distributions

A *binomial variable* X with parameters n,p is the sum of n independent Bernoulli trials, each chosen to be 1 with probability p and 0 with probability $1-p$. The *binomial distribution function* is $\mathrm{Prob}(X \leq x) = \Sigma_{k=0}^{x} \binom{n}{k} p^n (1-p)^{n-k}$.

The bounds of [Chernoff, 52] and [Angluin and Valiant, 79] imply

LEMMA A1.1. $\forall \varepsilon,p,n$ *where* $0 \leq p \leq 1$ *and* $0 < \varepsilon < 1$,

$$\mathrm{Prob}(X \leq \lfloor (1-\varepsilon)pn \rfloor) \leq \exp(-\varepsilon^2 np/2)$$

$$\mathrm{Prob}(X \geq \lceil (1+\varepsilon)np \rceil) \leq \exp(-\varepsilon^2 np/3) \; .$$

LEMMA A1.2. [Hoeffding, 56]. *Let* $X_1,\ldots,X_n$ *be independent binomial variables. Then* $\Sigma_{i=1}^{n} X_i$ *is upper bound by a binomial variable with parameters* n,p *with mean* $np = \Sigma_{i=1}^{n} \mathrm{mean}(X_i)$.

A1.2 Hypergeometric Distributions

Fix p,s where $0 \leq p \leq 1$ and $0 \leq s \leq n$. Let A be a subset of $\{1,\ldots,n\}$ of size np. A *hypergeometric variable* Y with parameters s,np,n is defined as $Y = |S \cap A|$ where S is a random sample of s elements of $\{1,\ldots,n\}$ chosen without replacement.

Suppose we independently choose $s \leq n$ random integers $r_1,\ldots,r_s \in \{1,\ldots,n\}$. Let index i be the *conflicted* if $\exists$ distinct a,b such that $r_a = r_b = i$. Let Z be the total number of conflicted indices $i \in \{1,\ldots,n\}$.

LEMMA A1.3. Z *is upper bounded by a hypergeometric variable with parameters* s,s,n.

[Johnson and Katz, 69] attribute the following bound to Uhlmann.

LEMMA A1.4. *If* X *is binomial with parameters* s,p *and* Y *is hypergeometric with parameters* s,np,n *then*

$$\text{Prob}(X \leq x) > \text{Prob}(Y \leq x) \qquad \text{for} \qquad 0 < p \leq \frac{nx}{(s-1)(n+1)}$$

and

$$\text{Prob}(X \leq x) > \text{Prob}(Y \leq x) \qquad \text{for} \qquad \frac{(1+nx/(s-1))}{(n+1)} \leq p \leq 1 \quad .$$

APPENDIX A3: Proof of Parallel Sorting Algorithms

<u>Proof of Lemma 3.3</u>. Compute $(h_0,\ldots,h_k) = \text{PREFIX-SUM}(|I(1)|,\ldots,|I(n)|)$ in $O(\log n)$ using P processors by Lemma 2.1. We then set $\sigma(h_{k-1}+1),\ldots,\sigma(n_k)$ to consecutive elements in $I(k)$ using a total of $O(\log n)$ time and P processors (the required processor assignment can easily be done by using the prefix sum computation.) Then $k_{\sigma(1)} \leq \ldots \leq k_{\sigma(n)}$ is a sort. □

<u>Proof of Lemma 3.4</u>. To each processor $\pi \in [P]$, we assign key indices $J(\pi) = \{j \mid (\pi-1)\log n < j \leq \min(n,\pi \log n)\}$. Let each processor π sequentially sort the keys $\{k_j \mid j \in J(\pi)\}$ by BUCKET-SORT in time $O(\log n)$, and so compute each list $J_{\pi,k} = (j \in J(\pi) \mid k_j = k)$ in increasing order of indices for each key value $k \in [\log n]$. Then for each key value $k \in [\log n]$ we compose the lists $J_{1,k}\ldots J_{P,k}$ to form the list $I(k)$ of indices with key value k. Finally, we apply Lemma 3.3 to compute the required permutation σ ordering the indices as they appear in $I(1),\ldots,I(P)$. The total time is $O(\log n)$ using P processors. □

<u>Proof of Lemma 3.5.</u> Let $k_i' = \lceil k_i/\log n \rceil + 1$ and let $k_i'' = k_i - (k_i'-1)\log n+1$ for each $i \in [P]$. We first apply Lemma 3.4 to get a sort of $k_1',\ldots,k_n'$, yielding a permutation σ. Then we apply Lemma 3.4 again to get stable sort of $k''_{\sigma(1)},\ldots,k''_{\sigma(n)}$, yielding a permutation σ'. Then $k_{\sigma'(1)} \leq \ldots \leq k_{\sigma'(n)}$, and hence σ' is a sort of $k_1,\ldots,k_n$. □

<u>Proof of Lemma 3.6.</u> If $d_0(\log n)^2 \geq |I(k)|$ then always $N(k) \geq d_0(\log n)^2 \geq |I(k)|$. Else suppose $d_0(\log n)^2 < |I(k)|$. $|I_S(k)|$ is upper bounded by a binomial variable with parameters $n/\log n$, $|I(k)|/n$. The Chernoff bounds given in Appendix A.1, Lemma A1.1, imply $\exists c\ \forall \alpha \geq 1$ if $c_0 = (c\alpha)^{-1}$ then

$\text{Prob}(|I_S(k)| \geq d_0^{-1}|I(k)|/\log n) \geq 1-1/n^{\alpha}$. Since $N(k) \geq d_0|I_S(k)|\log n$, the probability bounds hold as claimed. □

Proof of Claim 3.1. By Lemma 3.6, with likelihood $\geq 1-1/n^{\alpha}$, we can assume $N(k) \geq |I(k)|$. Let $n_k = \Sigma_{i\in N(k)}$ CONFLICT(i). The key observation is that on each stage t, $1/\log n$ of the key indices of $I(k)$ are assigned to random positions of the table A_k. Let $n_{k,t}$ be the number of indices $i \in N(k)$ where CONFLICT(i) is set to 1 on stage t. Then by definition $n_k = \Sigma_{t=1}^{\log n} n_{k,t}$.
We now apply the probabilistic bounds given in Appendix A.1, and we consider upper bounds on probability variables to be over the range of probability densities from $1/n^{\alpha}$ to $1-1/n^{\alpha}$. By Lemma A1.3, each $n_{j,t}$ is upper bounded by a hypergeometric variable with parameters $|I(k)|/\log n$, $|I(k)|/\log n$, $|I(k)|$. Then Lemma A1.4 implies each $n_{k,t}$ is upper bounded by a binomial variable with parameters $N(k)/\log n$, $1/\log n$. Hence by (Hoeffding's inequality) Lemma A1.2, $n_k = \Sigma_{t=1}^{\log n} n_{k,t}$ is upper bounded by a binomial variable with parameters $N(k)$, $1/\log n$. Furthermore $\Sigma_{k\in[D]} N(k) \leq O(n)$, so $\Sigma_{k\in[D]} n_k$ is upper bounded (by Hoeffding's inequality) by a binomial variable with parameters $O(n)$, $1/\log n$. The Chernoff bounds given in Lemma A1.1 immediately imply the claimed probabilistic bounds on n'. □

Proof of Corollary 3.1. We execute the following algorithm.

Step 1 for each processor $\pi \in [P]$ in parallel
do for each $t = 1,\ldots,\log n$
do $i_\pi \leftarrow (\pi-1)\log n+t$
randomly chose $k_{i_\pi} \in [P]$
od od

Step 2 Sort $k_1,\ldots,k_n$ and compute $I(k) = \{i \mid k_i = k\}$ for each key value $k \in [P]$

Comment. The sort can be done by Lemma 3.1 in time $\tilde{O}(\log n)$ using P processors.

CLAIM 3.2. With high likelihood, $|I(k)| \leq O(\log n)$ for each $k \in [P]$. In particular $\exists c\ \forall \alpha \geq 1\ \text{Prob}(|I(k)| \leq c\alpha \log n) \geq 1-1/n^{\alpha}$.

Proof. Each $|I(k)|$ is upper bounded by a binomial variable with parameters n, $\log n/n$. Hence the claimed bounds follow from the Chernoff bounds of Lemma A1.1. □

Step 3 for each $\pi \in [P]$ in parallel

do let $L(k)$ be a random permutation of the elements of $I(k)$ od

Comment. A random permutation $I(k)$ can easily be sequentially computed in $O(|I(k)|)$ time by a single processor.

Step 4 Compute $\sigma = (\sigma(1),\ldots,\sigma(n))$, the permutation of $(1,\ldots,n)$ which gives the order of appearance of the indices in $L(1),\ldots,L(P)$.

Comment. This can be done in $O(\log n)$ time by Lemma 3.3.

output random permutation of σ.

The total time for the steps 1-4 is $\tilde{O}(\log n)$ using P processors. □

APPENDIX A4: Proof of Theorem 4.1

A4.1 Analysis of RANDOM-MATE

Proof of Lemma 4.1

Let F be a spanning forest of G. By deleting at most $1/2$ the edges of F (but no active vertices), we get $F' \subseteq F$, a forest of trees of height 1, which contains all the active vertices. On the average, at least $1/4$ of the leaves of each tree of F' are mated, since their root has probability $1/2$ of being assigned female, and half of the leaves on the average will be (independently) assigned male. Hence with probability $1/2$, at least $1/8$ of all active vertices are mated. (Note: we can improve this result to show $\geq 1/4$ of all active vertices are mated on the average.) □

Proof of Claim 4.1. We prove this by induction on the number of relations of the main loop. This initially holds when $R(v) = v$ for all $v \in V$. Suppose the claim holds up to the $t-1$ iteration of the main loop. Then a R-root r is merged into an R-root r' by assigning $R(R(r)) \leftarrow R(r')$ only if $\exists\{v,u\} \in E$ such that $r = R(v)$ and $r' = R(u)$. Hence the claim holds after the t'th iteration of the main loop. □

Proof of Lemma 4.2.

Let R_t be the value of the array R just before the beginning of the t'th iteration of the main loop. Let a R_t-root r be *active* if $\exists\{u,v\} \in E$ such that $R_t(v) = r$ but $R_t(v) \neq R_t(u)$. Let n_t be the number of distinct active R_t-roots on the t'th iteration. Let the execution of RANDOM-MATE of the t'th iteration be a *success* if $n_{t+1} \leq \gamma n_t$ where $\gamma = 1/8$. By Lemma 4.1, the total number of successes after t_0 iterations is lower bounded by a binomial variable with parameters t_0, $1/2$. Observe that if we have $\log_\gamma n + 1$ successes after t_0 iterations, then $n_{t_0} = 0$. By the Chernoff bounds on the binomial given in Lemma A1.1 of the Appendix A, $\forall \alpha \geq 1\ \exists c_0$ such if $t_0 = c_0 \log n$ then

$\mathrm{Prob}(n_{t_0} = 0) \geq \mathrm{Prob}(\text{the number of successes after } t_0 \text{ iterations is} \geq 1 + \log_\gamma n) \geq 1 - 1/n^\alpha.$

Thus with probability $\geq 1-1/n^{\alpha}$, after $c_0 \log n$ iterations of RANDOM-MATE there are no remaining active vertices. □

A4.2 An Optimal Algorithm for $\geq n(\log n)^2$ Edges

In this subsection we take as input a graph $G=(V,E)$ such that $V=\{1,\ldots,n\}$ and the edge set E is of size $m \geq n(\log n)^2$.

Our algorithm RANDOM-MATE$'$ will be a simple modification of RANDOM-MATE.

To avoid unnecessary notation (ie, the use of ceiling and floor functions) we assume without loss of generality that $\log n$ divides m.

We will use a total of $P=n/\log n$ processors. We will begin by sorting the list $D(E)$ of directed edges into adjacency list arrays $E(1),\ldots,E(n)$ where $E(v)$ is an array containing the sets of directed edges departing vertex v. Since $|D(E)| = 2|E|$, by Theorem 3.1, this sorting can be done in $\tilde{O}(\log n)$ time using P processors.

We assign to each vertex $v \in V$ a set of $\log n$ consecutive processors $P_v = \{(v-1)\log n + 1,\ldots,v \log n\}$. We alter the main loop of RANDOM-MATE to execute $c_1 \log n$ times (instead of $c_0 \log n$ times) where c_1 is a constant to be determined below. We also delete the original code at label merge, and substitute in its place;

<u>merge</u>: <u>for</u> each $v \in V$ in parallel

    <u>do</u> <u>for</u> each processor $\in P_v$ in parallel

        <u>do</u> <u>if</u> $E(v) \neq \emptyset$ <u>then</u>

            choose a random edge $(v,u) \in E(v)$ <u>fi</u>

            MATE (u,v)

        <u>od</u>

    <u>od</u>

An edge $\{v,u\}$ is an *R-loop* if $R(v)=R(u)$.

<u>Claim 4.2.</u> $\forall \alpha \geq 1\ \exists c_1$, with probability $\geq 1-1/n^{\alpha}$ there are at most $m/\log n$ edges of E which are not R-loops after the $c_1 \log n$ iterations of the main loop of RANDOM-MATE$'$.

Proof. Let R_t be the value of the R array just before the t'th iteration of the main loop. Let R_t-root r be *semiactive* if at least $1/\log n$ of the edges $\{\{v,u\} \in E \mid R(v) = r\}$ are not R_t-loops. Let n'_t be the number of semiactive R_t-roots. We can assume without loss of generality that $n \geq 4$. For any semiactive R_t-root r, with probability at least $(1 - 1/\log n)^{\log n} \geq 1/4$, some process of P_v chooses an edge $\{v,u\} \in E$ on step t such that $R(v) = r$, $R(u) \neq r$ and we execute MATE(v,u). Also, prob(SEX(R(v)) = male and SEX(R(u)) = female) = 1/4. Hence using arguments similar to Lemma 4.1 we have with probability at least 1/2, at most $\gamma' n_t$ semiactive R_t-roots are not merged on step t to other R_t-roots where $\gamma' = 31/32$. Let the t'th iteration of the main loop be *successful* if $n'_{t+1} \leq n'_t \gamma'$. We have just shown the t'th iteration is successful with probability at least 1/2. The total number of successes after $t_1 = c_1 \log n$ iterations is lower bounded by a binomial variable with parameters t_1, 1/2. The Chernoff bounds of Lemma A1.1 imply: $\forall \alpha \geq 1\ \exists c_1$ with probability $\geq 1 - 1/n^{\alpha}$, the number of successes after t_1 iterations is $> \log_{\gamma'} n$. But $n'_{t_1} = 0$ after $1 + \log_{\gamma'} n$ successful iterations, and hence there are no remaining semiactive R-roots.

After completing execution of these modified main loop, RANDOM-MATE' deletes each R-loop edge $\{u,v\} \in E$ (where $R(u) = R(v)$) in time $O(\log n)$ using P processors. Finally, RANDOM-MATE' executes the original procedure RANDOM-MATE described in 4.1 to collapse the resulting graph to its connected components. Hence we have

LEMMA 4.3. *In time* $\tilde{O}(\log n)$ *using* $m/\log n$ *processors we can compute* CC(G) *for any graph* G *with* n *vertices and* $m \geq n(\log n)^2$ *edges.*

A4.3 An Optimal Algorithm for $\geq n(\log n)^{1/3}$ Edges

LEMMA 4.4. *Given any graph* $G = (V,E)$ *with* n *vertices and* $m \geq n(\log n)^{1/3}$ *edges, we can compute* CC(G) *in time* $\tilde{O}(\log n)$ *using* $(m+n)/\log n$ *processors.*

To prove this lemma, we describe another modification of RANDOM-MATE which we call RANDOM-MATE". We will give a simplified description of RANDOM-MATE". We will take as input a graph $G = (V,E)$ with n vertices $m \geq n(\log n)^{1/3}$ edges.

In this case, we assign to each processor $\pi \in [m/\log n]$ a set V_π of $(\log n)^{1/2}$ distinct consecutive vertices of $V = \{1,\ldots,n\}$. Also we again construct, by sorting E, adjacency list arrays $E(1),\ldots,E(n)$.

In this case we will execute the main loop only $c_2(\log n)^{1/4}$ iterations where c_2 is a constant to be defined below. We modify the main loop by substituting in place of the code at label merge, an assignment of $R'(v) \leftarrow R(v)$ for each vertex $v \in V$ and then the following code:

```
merge: for each processor π ∈ [m/log n] in parallel do
           for each v ∈ V_π
               do for i = 1,...,(log n)^(1/4)
                   if R(v) = R'(v) and E(v) ≠ ∅ then
                   do choose a random edge (v,u) ∈ E(v)
                       MATE(v,u) fi od
               od
```

The test $R(v) = R'(v)$ insures that the resulting R-trees will be of height ≤ 1 after executing the code at label collapse. Note that the resulting main loop takes time $O(\log n)^{3/4}$ per iteration, and so the total time is $O(\log n)$ using $m/\log n$ processors.

CLAIM 4.3. $\exists c_2$ such that with probability 1 as $n \to \infty$, there are at most $m/(\log n)^{1/12}$ edges of E which are not R-loops after $c_2(\log n)^{1/4}$ iterations of the main loop of RANDOM-MATE".

Proof of Claim 4.3. The proof is almost identical to that of Claim 4.2, except that in this case we must redefine a R_t-root to be semiactive if at least $1/(\log n)^{1/12}$ of the edges $\{\{v,u\} \in E | R(v) = v\}$ are not R_t-loops. If we let n''_t be the number of (so defined) semiactive R_t-roots, then again we have $\mathrm{Prob}(n''_{t+1} \leq n''_t \gamma') \leq 1/2$ where again $\gamma' = 31/32$. Hence with probability $\geq 1 - 2^{-(\log n)^{1/4}}$ no semiactive R-root exists after $c_2(\log n)^{1/4}$ iterations, where c_2 is determined by Lemma A1.1. □

Claim 4.3 implies that after 12 applications of RANDOM-MATE", the resulting graph has only $m/\log n$ edges, and hence we can apply RANDOM-MATE, Lemma 4.1, to completely collapse the graph and hence to determine its connected components in $\tilde{O}(\log n)$ time using $m/\log n$ processes.

A4.4 An Optimal Algorithm for $\leq n(\log n)^{1/3}$ Edges

Let $G=(V,E)$ to be a graph with n vertices and $m \leq n(\log n)^{1/3}$ edges. By Lemma 4.4, it suffices to show in time $\tilde{O}(\log n)$ using $P=(m+n)/\log n$ processors we can reduce the problem of computing $CC(G)$ to the problem of computing the connected components of a partially collapsed graph with $\leq O(m/(\log n)^{1/3})$ vertices and $\leq m$ edges. Without loss of generality we can assume $m \geq n-1$ and $2m$ is divisible by $\log n$.

Let $D(E)=((v_1,u_1),\ldots,(v_{2m},u_{2m}))$ be a list of the directed edges derived from E. We begin by computing a random permutation σ of $(1,\ldots,2m)$ by Corollary 3.1 in time $\tilde{O}(\log n)$ using P processors. We initially assign $R(v)=v$ and $SEX(v)=$ female for each vertex $v \in V$. This can easily be done in $O(\log n)$ time using P processors. Then we execute the following $\log n$ steps:

<u>for</u> $t=1,\ldots,\log n$ <u>do</u>

  <u>for</u> each processor $\pi \in [2m/\log n]$ in parallel

    <u>do</u> $MATE'(v_{\sigma((\pi-1)\log n+t)}, u_{\sigma((\pi-1)\log n+t)})$ <u>od</u>

  <u>do</u>

where we define:

<u>procedure</u> MATE'(v,u)

  $SEX(R(v)) \leftarrow$ <u>male</u>

  <u>if</u> $SEX(R(u))=$ <u>female</u> <u>then</u> $R(R(v)) \leftarrow R(u)$ <u>fi</u>

Note that each of iteration step takes only time $O(1)$ using P processors. Let a vertex of $R(G)$ be *special* if either it is isolated, or has degree $\geq (\log n)^{1/3}$, or is adjacent (by an edge of $R(G)$) to a vertex of degree $\geq (\log n)^{1/3}$.

CLAIM 4.4. The resulting partially collapsed graph $R(G)$ has $\leq \tilde{O}(n/\log n)^{1/3})$ vertices which are not special, and $\leq m$ edges.

<u>Proof of Claim 4.4</u>. Let R_t be the value of R just before the t'-th iteration. Let E_t be the set of directed edges chosen on the t'-th iteration, so $D(E)=\cup_t E_t$. Let M_t be the number of edges $(v,u) \in E_t$ such that

(i) v has degree $(\log n)^{1/3}$ in $R_t(G)$ and

(ii) a processor π executes MATE(v,u) but finds $SEX(R(u)) \neq$ <u>female</u>, so does not assign $R(R(v)) \leftarrow R(u)$.

Observe that initially, all vertices $v \in V$ have been assigned $SEX(v) =$ female, and that on successive stages $t = 1, \ldots, \log n$ at most $m/(\log n - t) \leq n(\log n)^{1/3}/(\log n-t)$ vertices $v \in V$ have been assigned $SEX(v) =$ male.

We can upper bound M_t by a hypergeometric variable, and then apply Lemma A1.4 to show that M_t is upper bounded (for probabilities in the range from $1/n^{\alpha}$ to $1-1/n^{\alpha}$) by a binomial variable with parameters $m/\log n$, $\max((\log n)^{1/3}/(\log n-t), 1)$. Applying (Hoeffding's inequality) Lemma A1.2, we get $\Sigma_{t=1}^{\log n} M_t$ is upper bounded by a binomial with mean $\Sigma_{t=1}^{\log n} m/((\log n)^{2/3}(\log n-t)) \leq ((m \operatorname{loglog} n)/(\log n)^{2/3}) \leq O(n/(\log n)^{1/3})$ and parameters m, $O((\operatorname{loglog} n)/(\log n)^{2/3})$. Then $\Sigma M_t + O(n/(\log n)^{1/4})$ gives an upper bound on the number of vertices of $R(G)$ which are not special. Finally we apply the Chernoff bounds of Lemma A1.1 proving the Claim. □

To complete the reduction, we delete each isolated R-root of $R(G)$, and for each $r \in R(V)$ with degree $< (\log n)^{1/3}$ in $R(G)$, we reassign $R(r) \leftarrow r'$ if there exists an edge $(r,r') \in R(E)$ such that r' has degree $\geq (\log n)^{1/3}$ in $R(G)$. We also update $R'(v) \leftarrow R(R(v))$ for each $v \in V$. These final steps can easily be done in $O(\log n)$ time using $(m+n)/\log n$ processors. The resulting further collapsed graph $R'(G)$ has $\leq \tilde{O}(n/(\log n)^{1/3})$ vertices and $\leq m$ edges. Therefore we can apply Lemma 4.4 to completely collapse $R'(G)$ to $R''(G)$. The array R'' specifies the connected components of G. Thus we have shown:

LEMMA 4.5. *Given any graph* G *with* n *vertices and* $m \leq n(\log n)^{1/3}$ *edges, we can compute* $CC(G)$ *in* $\tilde{O}(\log n)$ *time using* $(m+n)/\log n$ *processors.*

This completes the proof of Theorem 4.1.

END

FILMED

1-86

DTIC