In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures* (1993), pp. 224-229.

Optimal Parallel Construction of Hamiltonian Cycles and Spanning Trees in Random Graphs

(Preliminary Version)

Philip D. MacKenzie[1] Quentin F. Stout[2]
Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

We give tight bounds on the parallel complexity of some problems involving random graphs. Specifically, we show that a Hamiltonian cycle, a breadth first spanning tree, and a maximal matching can all be constructed in $\Theta(\log^* n)$ expected time using $n/\log^* n$ processors on the CRCW PRAM. This is a substantial improvement over the best previous algorithms, which required $\Theta((\log\log n)^2)$ time and $n\log^2 n$ processors. We then introduce a technique which allows us to prove that constructing an edge cover of a random graph from its adjacency matrix requires $\Omega(\log^* n)$ expected time on a CRCW PRAM with $O(n)$ processors. Constructing an edge cover is implicit in constructing a spanning tree, a Hamiltonian cycle, and a maximal matching, so this lower bound holds for all these problems, showing that our algorithms are optimal. This new lower bound technique is one of the very few lower bound techniques known which apply to randomized CRCW PRAM algorithms, and it provides the first nontrivial parallel lower bounds for these problems.

1 Introduction

By a *random graph on n vertices*, denoted $G_{n,p}$, we mean that each of the $\binom{n}{2}$ edges is included with probability p, where p is a constant, $0 < p < 1$. Random graphs have been extensively studied and have very interesting properties. For instance, with high probability, in $G_{n,p}$ a Hamiltonian cycle exists; every breadth first spanning tree has height 2; and, for n even, there is a maximal matching which includes all the vertices. In this paper, we study the parallel complexity of constructing a Hamiltonian cycle, a breadth first spanning tree, and a maximal matching in $G_{n,p}$.

Perhaps the most surprising result in this paper is a nearly-constant time algorithm for constructing a Hamiltonian cycle in $G_{n,p}$. It is well known that the general problem of finding a Hamiltonian cycle in a graph is NP-complete. Because this is a very important problem with many practical applications, researchers have studied under what conditions this problem becomes tractable. Some of the most successful results of this type are for the class of random graphs. Bollobás, Fenner and Frieze [BFF87] give an algorithm which constructs a Hamiltonian cycle in $G_{n,p}$ if one exists. This algorithm runs in polynomial expected time for $p \geq \frac{1}{2}$. Gurevich and Shelah [GS87] and Thomason [Tho89] independently improve on this result, giving algorithms which run in linear expected time for any constant p. These algorithms are optimal, since linear time is needed just to write the output. Frieze [Fri87] gives a parallel algorithm which constructs a Hamiltonian cycle

[1]Supported by an AT&T Fellowship and by NSF/DARPA grant CCR-9004727. Current address: Dept. of Computer Sciences, Univ. of Texas, Austin, TX, 78712-1188

[2]Supported by NSF/DARPA grant CCR-9004727.

from $G_{n,p}$ in $O((\log\log n)^2)$ expected time and uses $n\log^2 n$ processors. In this paper we improve on this result substantially, showing that one can find a Hamiltonian Cycle in $\Theta(\log^* n)$ expected time while using only $n/\log^* n$ processors. This achieves linear speedup, and is thus PT-optimal.

A slight modification to the algorithm for constructing a Hamiltonian cycle allows us to construct a maximal matching with the same time and processor bounds. Also, we give a fairly simple algorithm for constructing a breadth first spanning tree, again with the same time and processor bounds.

Implicit in each of the problems above is finding an *edge cover*. We describe a new technique which allows us to prove a lower bound on finding an edge cover for $G_{n,p}$ from its adjacency matrix. Specifically, we show that finding an edge cover requires $\Omega(\log^* n)$ expected time using $O(n)$ processors in any parallel model of computation in which each processor can read at most one entry in the adjacency matrix in a single time step. (The CRCW PRAM is one such model.) Consequently, this implies that all the algorithms in this paper are optimal with respect to running time for up to $O(n)$ processors.

This new lower bound technique is one of the very few lower bound techniques known which apply to randomized CRCW PRAM algorithms, and it provides the first nontrivial parallel lower bounds for these problems. The technique is formulated in terms of a new model of parallel computation, the *parallel target shooting model*. This model is conceptually similar to Valiant's parallel comparison model, and can be used to analyze any problem which can be reduced to processors probabilistically hitting targets. In our case the problem is to construct an edge cover, and hitting a target implies locating an edge covering a specified vertex. Finding other such problems could be very helpful in proving more lower bounds for randomized parallel algorithms.

2 Definitions

The algorithms presented here run on the Arbitrary Concurrent Read, Concurrent Write (CRCW) Parallel Random Access Machine (PRAM). In this model, both concurrent reads and concurrent writes are allowed, and if two or more processors write to any memory cell, an arbitrary processor succeeds in writing its value to that cell.

We define *high probability* as meaning probability $\geq 1 - n^{-\alpha}$ for any constant $\alpha > 1$.

For any base $z \geq 2$, we define $\log_z^{(1)} n$ as $\log_z n$ and $\log_z^{(i)} n$ recursively as $\log_z(\log_z^{(i-1)} n)$. We define $\log_z^* n$ as the smallest integer i such that $\log_z^{(i)} n \leq 1$. When z is omitted we assume base 2. It is well known that $\log^* n$ is an extremely slow growing function of n, and in fact $\log^* n \leq 5$ for $n \leq 2^{65536}$. We can convert between bases using the following lemma.

Lemma 2.1 (MacKenzie [Mac92]) *For $z \geq 4$, $\log^* n \leq \log_z^* n + \log^* z$.*

3 Hamiltonian Cycle

We present here a parallel algorithm for finding a Hamiltonian cycle in $G_{n,p}$ which will succeed with high probability. If it does not succeed, we will simply revert to the linear expected time algorithm of Gurevich and Shelah [GS87]. We will explain later how we guarantee that failure in the parallel algorithm does not increase the expected time of the serial algorithm.

Our algorithm runs in three stages as follows:

Stage 1 Remove $\sqrt{n}$ vertices, and find a Hamiltonian cycle through them in constant time.

Stage 2 Connect the remaining $n - \sqrt{n}$ vertices into at most $\log^6 n$ simple disjoint paths in $O(\log^* n)$ time

Stage 3 In constant time, hook the $O(\log^6 n)$ simple disjoint paths found in Stage 2 into the Hamiltonian cycle found in Stage 1.

The next three subsections describe Stages 1, 2, and 3 respectively. We will need to use the following lemma.

Lemma 3.1 (Frieze [Fri87]) *The probability that a Hamiltonian cycle does not exist in a random graph $G_{n,p}$ is at most $n^2(1-p)^{n-1}$.*

3.1 Hamiltonian cycle through $\sqrt{n}$ vertices

We attempt to find a Hamiltonian cycle through the first $\sqrt{n}$ vertices using n processors using the following procedure:

Step 1 Assign $\sqrt{n}$ processors to each vertex. Let $s = \sqrt{\log n}$, and partition the vertices into consecutive groups of size s. Then with the processors assigned to the vertices in each group we can check all $s!$ possible Hamiltonian cycles in each group in constant time, and choose one, if one exists.

Step 2 Each vertex in a group without a Hamiltonian cycle uses $\log^2 n$ processors to try to connect to another group's Hamiltonian cycle. Each processor for a vertex v randomly picks a vertex v' from the set of $\sqrt{n}$ vertices and checks to see if v' both has not been chosen by another processor, and is part of a Hamiltonian cycle. If this is so, then the processor checks to see if v can be hooked into that cycle between v' and next(v') by checking if (v', v) and $(v$,next$(v'))$ exist. A processor which has found a place to hook is then chosen for vertex v, if one exists, and v is hooked into the chosen place.

Step 3 The lowest numbered vertex in each of the original groups with a Hamiltonian cycle breaks its cycle by cutting its outgoing edge. Now we are left with Hamiltonian paths which we must try to connect. Each group which originally contained a Hamiltonian path P_1 uses $\sqrt{n}$ processors to find the next group which originally contained a Hamiltonian path P_2 in constant time. Then it can simply check for every vertex v in P_2's original group whether P_1 can be hooked into it, i.e. if v_1 and v_2 are the endpoints of P_1, then the processor checks if (v, v_1) and $(v_2$,next$(v))$ exist. A processor which has found a place to hook is chosen for P_1, if one exists.

Step 4 Find a set of $\log^4 n$ consecutive groups which are all connected and disconnect it from the rest by having the last group in this set remove its connection to the next group and close its cycle using the outgoing edge from the first vertex which we removed in the last step. Now partition the $\sqrt{n}/s$ groups up into consecutive *supergroups* of $\log^6 n$ groups. Now each path with endpoints in a supergroup of $\log^6 n$ groups uses $\log^2 n$ processors to randomly try to hook into a path at a vertex from the next supergroup of $\log^6 n$ processors, except for paths with endpoints in the last supergroup, which try to connect to the cycle of size $\log^4 n$ which we just formed.

Obviously, the procedure given takes constant time. The following lemmas prove that it works correctly.

Lemma 3.2 *After Step 1, the probability of a group not having a Hamiltonian cycle is $O(\log^{-7} n)$.*

Proof: By lemma 3.1, the probability of a group of $s = \sqrt{\log n}$ vertices in a random graph not containing a Hamiltonian cycle is $\leq s^2(1-p)^{s-1}$ which can be loosely bounded by $O(\log^{-7} n)$. □

Lemma 3.3 *After Step 2, all the vertices which were attempting to hook onto a Hamiltonian cycle succeeded with high probability.*

Proof: By lemma 3.2 and a Chernoff bound, we can show that with high probability, no more than $O(\sqrt{n}/s\log^7 n)$ groups failed in the first step, so there are $O(\sqrt{n}/\log^7 n)$ vertices attempting to hook in Step 2. Thus there will be $O(\sqrt{n}/\log^5 n)$ processors randomly picking vertices. The probability that a processor picks a vertex which is not part of a Hamiltonian cycle or picked by another processor is obviously $O(\log^{-5} n)$. Also, the probability that a chosen vertex can not be hooked into is $\leq 1-p^2$. Then the probability that all $\log^2 n$ processors assigned to one vertex fail is $\leq c^{-\log^2 n}$ for some constant c, and thus with high probability, all vertices will succeed. □

Lemma 3.4 *After Step 3, each supergroup of* $\log^6 n$ *consecutive groups will have at most* $O(\log n)$ *endpoints of paths, with high probability.*

Proof: The probability of not being able to hook the endpoints of a Hamiltonian path into another Hamiltonian path of s vertices is $\leq (1-p^2)^{s-1} \leq \log^{-7} n$. Then using Chernoff bounds, the lemma follows. □

Lemma 3.5 *After Step 3, there will be* $\log^4 n$ *consecutive groups which are connected, with high probability.*

Proof: By lemma 3.2 and lemma 3.4, with high probability, there will be $O(\sqrt{n}/s\log^5 n)$ groups which did not succeed in steps 1 and 3. If we then partition the $\sqrt{n}/s$ groups into consecutive sets of size $\log^4 n$, we will have more sets than failed groups, and thus there will be a set for which all the groups succeeded. □

Lemma 3.6 *After Step 4, we will have a Hamiltonian cycle, with high probability.*

Proof: In each supergroup except the last, there will be $O(\log^3 n)$ processors which randomly pick from $\log^6 n$ vertices, and there will be at most $O(\log n)$ vertices which are endpoints of Hamiltonian paths. Thus the probability that a processor picks an endpoint or picks a vertex which is picked by another processor is $O(\log^{-3} n)$. For the last supergroup, the only change is that the probability that a processor picks a vertex which is picked by another processor is $O(\log^{-1} n)$, and it obviously cannot pick an endpoint. The probability that the endpoints of a path can be hooked in at a chosen vertex is $\leq 1-p^2$. Thus for some constant c, the probability that any path cannot be hooked is $\leq c^{-\log^2 n}$. Then with high probability, all paths are successfully hooked.

This obviously forms at least one cycle. To form more than one, some endpoints would have to connect to their own path. But this cannot happend since endpoints only connect to paths at higher numbered vertices, except for the last supergroup. The last supergroup only connects to the cycle formed in Step 4, and thus could not connect to itself, since the cycle is disconnected from the other paths. Thus exactly one cycle is formed through all the vertices. □

3.2 Log-star paradigm

Let $m = n - \sqrt{n}$. Here we show how to use the log-star pardigm of Gil, Matias, and Vishkin [GMV91] to perform Stage 2 of our algorithm, connecting the vertices of a random graph $G_{m,p}$ into a small number of simple disjoint paths. Gil, Matias, and Vishkin [GMV91] consider the following abstract problem.

Problem 3.1 *Given an array of* m active items *and a constant* $0 < p < 1$*, assume that in a single round, a processor can be assigned to try to deactivate an active item, and if* q *processors are trying to deactivate an active item, they succeed with probability* $1-(1-p)^q$*. The problem is to deactivate all the items.*

The log-star paradigm solves this problem using m processors in an average of $\log^* m$ rounds. The expected number of active items at the beginning of round i is at most m/q_i^c, where $\{q_i\}$ is a sequence defined by $q_{i+1} = 2^{q_i}$,$q_1, c > 0$ are sufficiently large constants.

Round i consists of two steps:

Allocation Allocate to each active item a team of size q_i. The allocation succeeds for all but $\frac{1}{2}m/q_{i+1}^c$ active items with high probability. Only allocated items participate in the deactivation step.

Deactivation Deactivate each participating item. The deactivation succeeds for all but $\frac{1}{2}m/q_{i+1}^c$ participating items with high probability.

Matias and Vishkin [MV91] provide the allocation step, and thus only the deactivation step needs to be implemented for any specific problem.

In our specific problem, an active element will be the highest numbered endpoint of a simple disjoint path, or the single endpoint of a path of length 0. Each active element will have a pointer to the other endpoint of its path. Deactivation of an active element will consist of hooking the endpoints of a path into another path to create a longer path. A path $P_1 = (v_1, v_2, \ldots, v_k)$ can be hooked into another path $P_2 = (w_1, w_2, \ldots, w_l)$ to make a larger path $P_3 = (w_1, \ldots, w_j, v_1, \ldots, v_k, w_{j+1}, \ldots, w_l)$, if the edges (w_j, v_1) and (v_k, w_{j+1}) exist, and no other path is also trying to hook between vertices w_j and w_{j+1}. Note that in one time step, multiple paths could be hooked into a single path (between different vertices), and a path which is being hooked into can also hook into another path.

Let k and h be large enough constants such that the following analysis holds. Let $q_1 = \log^{(\log^* n - h)} n$. Note that q_1 is bounded by a constant (depending on h). For the first deactivation step, we partition the vertices into consecutive groups of size kq_1. These will be called the *initial groups*. For each initial group, we try all $(kq_1)!$ permutations to see if a Hamiltonian cycle through it exists. If so, then we will cut the edge out of the highest numbered vertex, and use the remaining Hamiltonian path through this initial group as part of the full Hamiltonian cycle. We let the highest numbered vertex (which is one of the endpoints of the Hamiltonian path through the initial group) be an active element, and store the number of the other endpoint of the path with this active element. If no Hamiltonian cycle through the initial group exists, then we consider each vertex in the group to be an active element. To complete the first step, for all initial groups with Hamiltonian paths, we use the kq_1 processors assigned to the group to try to hook into the next initial group in the ordering, if that group also has a Hamiltonian path.

For step $i > 1$, partition the vertices into groups of size q_i^6, and let $H_{i,j}$ be the jth group. Consider an active endpoint v_1 in $H_{i,j}$, which we assume is not the last group. We try to deactivate v_1 by hooking its path P to a vertex in $H_{i,j+1}$ which is in the interior of a path. We allocate q_i processors to v_1 using the allocation routine of Matias and Vishkin [MV91]. Each of these processors will randomly choose k vertices in $H_{i,j+1}$. A processor checks to see if any chosen vertex v' both has not been chosen by another processor, and has a successor in its path. If this is so, then it checks to see if P can be hooked into that path at this vertex, i.e. if v_1 and v_2 are the endpoints of P, then it checks if (v', v_1) and $(v_2,\text{next}(v'))$ exist. Then a processor which has found a place to hook is chosen for P, if one exists.

We now show that this procedure connects all the vertices into at most $\log^6 n$ simple disjoint paths.

Lemma 3.7 *After every step, and for every path, the endpoints of that path always belong to the same initial group.*

Proof: Through all the deactivation steps we are simply hooking paths into the interior of other paths. Therefore the endpoints of any path are simply the endpoints of one of the paths of an initial group. □

Lemma 3.8 *The active element of a path (which is an endpoint of the path) is the highest numbered vertex in the path.*

Proof: This is obviously true since we only connect endpoints to higher numbered vertices. □

Lemma 3.9 *After every step i, when the vertices are partitioned into consecutive groups of q_{i+1}^6 vertices, the probability that a group contains the endpoints of more than q_{i+1}^3 paths is $\leq 2^{-q_{i+1}^2}$.*

Proof: We show this by induction.

After step 1, by lemma 3.1, the probability that an initial group does not contain a Hamiltonian cycle is $\leq (kq_1)^2(1-p)^{kq_1-1}$, which for sufficiently large k and q_1 is less than $2^{-6q_1} = q_2^{-6}$. Using a Chernoff bound, the probability of over $2q_2^2$ initial groups not containing a Hamiltonian cycle out of q_2^6/kq_1 initial groups is $\leq 2^{-2q_2^2}$.

For all the other initial groups which contain Hamiltonian cycles, and whose next neighbors contain Hamiltonian cycles, the probability of not being able to connect to the next group is $\leq (1-p^2)^{(kq_1-1)/2}$, which for sufficiently large k and q_1 is less than $2^{-6q_1} = q_2^{-6}$. Using a Chernoff bound, the probability of over $2q_2^2$ groups out of q_2^6/kq_1 initial groups not being able to connect to the next group is $\leq 2^{-2q_2^2}$. In total, the probability of over $4q_2^2$ groups still containing unconnected endpoints is $\leq 2^{-2q_2^2+1} \leq 2^{-q_2^2}$. Since $4kq_2^2q_1 \leq q_2^3$, this proves the base step of the induction.

Now assume the lemma is true through step $i-1$. By induction the probability of any group $H_{i,j}$ of q_i^6 vertices containing over q_i^3 endpoints is $\leq 2^{-q_i^2} \leq q_{i+1}^{-q_i}$. Then by a Chernoff bound, the probability that over $2q_{i+1}^2$ groups $H_{i,j}$ in a group $H_{i+1,j}$ have over q_i^3 unconnected endpoints is $\leq 2^{-2q_{i+1}^2}$. We also know at most $4q_{i+1}^2q_i^6 \leq q_{i+1}^3/2$ vertices come from these groups, and groups which directly precede them.

For each j where $H_{i,j}$ and $H_{i,j+1}$ both contain at most q_i^3 unconnected endpoints, the probability of a processor assigned to an endpoint in $H_{i,j}$ not finding a vertex in $H_{i,j+1}$ to which it can hook its endpoint is the sum of the probabilities that it chooses an endpoint, that another processor chooses the same vertex, and that the two edges necessary for hooking do not exist. This sum is $\leq q_i^{-3} + kq_i^{-2} + (1-p^2) \leq c$ for some constant $c < 1$. Thus the probability of a vertex in $H_{i,j}$ not finding a place to hook is $\leq c^{kq_i} \leq 2^{-6q_i} \leq q_{i+1}^{-6}$. Using a Chernoff bound, the probability that over $q_{i+1}^3/2$ out of $\leq q_{i+1}^6$ vertices fail to hook is $\leq 2^{-q_{i+1}^3/2} \leq 2^{-2q_{i+1}^2}$. Thus the probability that there will be over q_{i+1}^3 endpoints in a group $H_{i+1,j}$ will be $\leq 2^{-2q_{i+1}^2+1} \leq 2^{-q_{i+1}^2}$. □

Corollary 3.1 *With high probability, after step $r = \log^* m - h - 1$, there will be at most $\log^6 n$ simple disjoint paths remaining.*

Proof: By a simple calculation, $q_r = \log n$. After step $r-1$, the probability that any group of q_r^6 vertices has over q_r^3 unconnected endpoints is $\leq 2^{-q_r^2} \leq 2^{-\log^2 n}$. Then, from the analysis above, the probability that any endpoint which is not located in the last group of q_r^6 vertices does not hook to a vertex in step r is $\leq 2^{-q_r^2} \leq 2^{-\log^2 n}$. Thus with high probability, only vertices from the last group will be active, and the last group is of size $\leq q_r^6 \leq \log^6 n$. □

3.3 Connecting the remaining paths

Each of the remaining $O(\log^6 n)$ active elements (endpoints) can easily be assigned $\log^2 n$ processors to try to randomly hook into the cycle of $\sqrt{n}$ vertices. The probability of a vertex failing is obviously $\leq c^{-\log^2 n}$ for some constant c, so with high probability, they all will succeed.

3.4 In case of failure

It is possible that our parallel algorithm could fail, though this can only occur with very small probability. If it does fail, we revert to the $O(n)$ expected time serial algorithm of Gurevich and Shelah [GS87]. Since the probability of failure will be less than $1/n$, this will not add anything to the expected time of the algorithm. Unfortunately, failure in our algorithm might imply failure in the linear expected time algorithm. To solve this problem, we use a nice technique from Gurevich and Shelah. Two coins will be flipped for each edge, one which decides if the edge is to be colored red, the other if it is to be colored blue. (For instance, with probability $\frac{1}{4}$ it will be colored both red and blue.) Then the probability that there is a red edge between any pair of vertices is $p/2$ and the probability that there is a blue edge between a pair of vertices given that whether a red edge exists is already known, will be at least $p/4$. We can then use the red edges in the parallel algorithm using the probability $p/2$, and the blue edges in the serial algorithm using probability $p/4$.

3.5 Reducing the number of processors

We can use the general technique for optimizing asynchronous geometric decaying algorithms in Gil, Matias and Vishkin [GMV91] to reduce the number of processors to $n/\log^* n$.

We have thus proven the following theorem.

Theorem 3.1 *For any constant $0 < p < 1$, in a random graph $G_{n,p}$, if any Hamiltonian cycles exist then one can be found, or it can be shown that none exist, in $\Theta(\log^* n)$ expected time on a CRCW PRAM of $n/\log^* n$ processors.*

If n is even, we can slightly modify the algorithm given so that vertices along the Hamiltonian cycle alternate between even and odd. It is fairly obvious that this algorithm will have the same time and processor bounds. Then if remove all the edges from odd to even vertices, the remaining edges will form a matching which includes all the vertices, which is therefore a maximal matching. If n is odd, we can remove one vertex and perform the procedure just described to the remaining vertices to form a matching which includes all but one vertex, which is also a maximal matching. If we are unable to find a Hamiltonian cycle in $\Theta(\log^* n)$ time, then we revert to the polynomial time algorithm in Lawler [Law76] to find a maximal matching. Similar modifications can be made to find depth-first spanning forests, and to decide if a graph is connected, giving the following theorem.

Theorem 3.2 *For any constant $0 < p < 1$, for a random graph $G_{n,p}$, a maximal matching can be found, a depth-first spanning forest can be found, and it can be decided if the graph is connected, all in $\Theta(\log^* n)$ expected time on a CRCW PRAM of $n/\log^* n$ processors.*

4 Breadth First Spanning Forest

Here we give an algorithm for constructing a breadth first spanning forest in $G_{n,p}$. With high probability $G_{n,p}$ will be connected and this will actually be a breadth first spanning tree. First

assume we have n processors. Without loss of generality, let the first vertex be the root. Then every vertex which is connected to the root marks itself as a node in the tree at level 1. We can show that with high probability all the rest of the nodes are at level 2. We simply must find for each of these vertices a vertex it is connected to at level 1.

We can use the log-star paradigm defined in the previous section to connect the rest of the vertices to the level 1 vertices. We simply must implement the deactivation step. Here we consider an active element to be a vertex which has not found a level 1 vertex to connect to.

In deactivation step i, we simply have each of the q_i processors assigned to a vertex v choose k random vertices and check whether they are level 1 vertices which connect to v. If one is found, then it can be marked as vertex v's parent and vertex v can be deactivated. The probability of a vertex failing to find a connection to a level 1 vertex in step i is $(1-p^2)^{kq_i}$. Then for any constant c, a sufficiently large k can be found so that this probability is less than $2^{-cq_i}/4$. Using a Chernoff bound, we can show that with high probability, at most twice the average number, or $\frac{1}{2}n/q_{i+1}^c$, fail. This shows that the deactivation step succeeds with high probability. If the log-star paradigm fails, then we can simply perform a $\Theta(n^2)$ time serial breadth-first spanning forest algorithm

We can use the general technique for optimizing asynchronous geometric decaying algorithms in Gil, Matias and Vishkin [GMV91] to reduce the number of processors to $n/\log^* n$.

We have thus proven the following theorem.

Theorem 4.1 *For any constant $0 < p < 1$, a breadth first spanning forest in a random graph $G_{n,p}$ can be found in $\Theta(\log^* n)$ expected time using a CRCW PRAM of $n/\log^* n$ processors*

5 Parallel Target Shooting Model

Consider a model of computation based on the abstract deactivation problem given in Section 3.2. In this model, the *Parallel Target Shooting Model*, there are n targets, n processors, a constant integer $m \geq 1$, and a constant probability $0 < p < 1$. In a single round, a processor can try to shoot m active targets. With probability p it hits all of them, and with probability $1-p$ it hits none of them. If q processors are trying to shoot an active target, then the target is hit with probability $1-(1-p)^q$, deactivating the target. It is assumed that at the start of each round, the processors know all the results from previous rounds, and jointly decide how to allocate themselves to active targets in the current round. Note that this is similar to Valiant's parallel comparison model, with a probabilistic hitting operation substituted for a comparison operation.

We will show that the problem of hitting all n targets in this model requires $\Omega(\log^* n)$ expected time. We will then use this to show that constructing an edge cover of a random graph represented by its adjacency matrix requires $\Omega(\log^* n)$ expected time on any model of computation in which a processor can read at most one entry in a single step. This includes the CRCW PRAM model.

We define k_t recursively as $k_t = (1-p)^{-3k_{t-1}m}$ and we let k_0 be a large enough constant so our analysis holds.

Lemma 5.1 *After step t, with high probability at least n/k_t targets will still be active.*

Proof: We prove this by induction. Assume that the lemma holds up to step $t-1$. Let $k = k_{t-1}$. At step t, each of n processors picks a group of m targets to attempt to hit. On average, each of the n/k active targets is picked by at most km processors, and at least half of these targets are picked by at most twice the average number of processors. Thus at least $n/2k$ of the active targets are picked by at most $2km$ processors each. Whether these targets are hit may not be independent, however, so we must find an independent set of these items as follows.

Consider each target as a vertex, and each group of m targets picked by a processor as a clique of m vertices. If we let $v = n/2k$ and $e = n\binom{m}{2} \leq nm^2$, then by Turán's theorem [Ber73], we can find an independent set of these vertices (targets) of size at least $v^2/(v+2e) = n/2k(4km^2+1)$. Each target has at least a $(1-p)^{2km}$ independent chance of not being hit in this step. Thus on average at least $n(1-p)^{2km}/2k(4km^2+1)$ targets will remain active, and by Chernoff's bound, for large enough n and $k \leq \log\log n$, with high probability at least half of these targets will remain active. For large enough k, $(1-p)^{2km}/2k(4km^2+1) \geq (1-p)^{3km}$, proving the lemma. □

Theorem 5.1 *For any constant $0 < p < 1$, hitting n targets with n processors in the parallel target shooting model requires $\Omega(\log^* n)$ expected time.*

Proof: Using lemma 2.1, we see that for $T = \log^* n - \log^*((1-p)^{-3m}) - \log^* k_0 - 3 = \Omega(\log^* n)$, $k_T \leq \log\log n$. Then by lemma 5.1, at step T, with high probability at least $n/\log\log n$ targets will be active. □

6 Lower Bound on Constructing an Edge Cover

Here we show that constructing an edge cover in a random graph represented as an adjacency matrix requires $\Omega(\log^* n)$ expected time using n processors on any model in which a single processor can only read one entry in the adjacency matrix in one step. Thus we can assume that every processor knows every entry in the adjacency matrix which has been read, and at any step, the processors jointly decide where to read based on all the entries read so far.

Theorem 6.1 *For any constant $0 < p < 1$, any problem which requires the construction of an edge cover (with high probability) in a random graph $G_{n,p}$ represented as an adjacency matrix requires $\Omega(\log^* n)$ expected time on any n processor parallel model of computation in which a single processor can only read one entry in the adjacency matrix at each step.*

Proof: We model this in the parallel target shooting model, where each vertex is a target, and where a target is hit whenever an edge is found having the vertex as one of its endpoints. Reading an entry in the adjacency matrix is simply a processor picking two targets and hitting them with probability p. Finding an edge cover is simply hitting all the targets. Thus by theorem 5.1 finding an edge cover requires $\Omega(\log^* n)$ expected time. □

Theorem 6.2 *For any constant $0 < p < 1$, in a random graph $G_{n,p}$ represented as an adjacency matrix, constructing a Hamiltonian cycle, spanning tree, or maximal matching, or deciding if the graph is connected, each require $\Omega(\log^* n)$ expected time on any n processor parallel model of computation in which a single processor can only read one entry in the adjacency matrix at each step.*

Proof: With high probability a Hamiltonian cycle, a spanning tree, and a maximal matching exist in $G_{n,p}$, and constructing any of these implies construction of an edge cover. Thus by theorem 6.1, the stated lower bound holds. □

A Probabilistic Tools

One technique we use is the Chernoff bound [Che52]. This can be used when we wish to bound the distribution of a random variable Z which is the sum of n independent random variables. For a binomial random varible $Z \sim B(n,p)$, where Z is the sum of n independent Bernoulli trials with

probability of success p, Angluin and Valiant [AV79] show that for $0 < \beta < 1$, one can obtain the bounds

$$P(Z \geq (1+\beta)np) \quad \leq \quad e^{-\beta^2 np/3},$$

and

$$P(Z \leq (1-\beta)np) \quad \leq \quad e^{-\beta^2 np/2}.$$

From this we obtain the bound

$$P(Z \geq 2np) \leq 2^{-4np/9}.$$

Also, for $k \geq 6$ we obtain the bound

$$P(Z \geq knp) \quad \leq \quad 2^{-knp}.$$

References

[AV79] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *J. Comput. System Sci.*, 18:155–193, 1979.

[Ber73] C. Berge. *Graphs and Hypergraphs.* North Holland, 1973.

[BFF87] B. Bollobás, T. I. Fenner, and A. M. Frieze. An algorithm for finding hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23:493–507, 1952.

[Fri87] A. M. Frieze. Parallel algorithms for finding hamilton cycles in random graphs. *Inform. Process. Lett.*, 25:111–117, 1987.

[GMV91] J. Gil, Y. Matias, and U. Vishkin. Towards a theory of nearly constant time parallel algorithms. In *Proc. 32nd Symp. on Found. of Comp. Sci.*, pages 698–710, 1991.

[GS87] Y. Gurevich and S. Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[Mac92] P. D. MacKenzie. Load balancing requires $\Omega(\log^* n)$ expected time. In *3rd ACM-SIAM Symp. on Disc. Alg.*, pages 94–99, 1992. submitted to SIAM Journal on Computing.

[MV91] Y. Matias and U. Vishkin. Converting high probability into nearly-constant time – with applications to parallel hashing. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 307–316, 1991.

[Tho89] A. Thomason. A simple linear expected time algorithm for the hamilton cycle problem. *Discrete Math.*, 75:373–379, 1989.