Ultrafast Graph Algorithms on Reconfigurable Meshes

(Extended Abstract)

Douglas M. Van Wieren Quentin F. Stout

Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109

dvw@umich.edu qstout@umich.edu

Abstract

We present a selection of related algorithms which run in $\Theta(\log^\star(n))$ expected time for random graphs on the modestly reconfigurable architecture known as the Mesh with Row and Column Subbuses (RCS-Mesh). Notably, this base model has been implemented as part of the communication architecture of the MasPar MP series [17]; in addition, algorithms for RCS-Mesh can be directly implemented on the more powerful (and more widely studied) Reconfigurable Mesh (RMesh).

Specifically, we show that the $n \times n$ RCS-Mesh can find and mark a Hamiltonian cycle (or determine that none exists) in a random graph $G_{n,p}$ on n vertices (p is the probability of an edge existing between any two given vertices) in $\Theta(\log^\star(n))$ expected time. Moreover, with only additional constant time the RCS-Mesh is capable of labeling the vertices in the order in which they appear in the Hamiltonian cycle.

Our approach hinges on an algorithm with the same time bounds for performing bipartite semi-matching—given a random bipartite graph with constraints on the relative sizes of the partitions, the algorithm finds a set of edges which map the smaller partition injectively into the larger. This algorithm forms the basis of a $\log^\star$ paradigm for the RCS-mesh and can be applied to problems outside the consideration of this paper.

Beyond the immediate implications such as exact matchings and depth-first search trees, our approach to the Hamiltonian cycle problem can be ported simply to other architectures and allows unusual generalizations. In particular, our method allows probabilistic location of regular structures of bounded degree in a random graph in $\Theta(\log^\star(n))$ expected time, with probability of failure bounded above by r^n for some $r \in (0,1)$. Examples of this include finding tori and complete binary trees.

We also show that RCS-Mesh can determine, in $\Theta(1)$ expected time, if a random graph is strongly n^ϵ-connected, where ϵ is an arbitrary constant less than one.

Keywords: Mesh with Subbuses, Hamiltonian cycles, NP-Completeness, Random Graphs, Average Case Analysis

1 Introduction

This paper shows that a variety of difficult problems on random graphs can be solved very rapidly using the modest reconfigurable bus structure of the RCS-mesh. The RCS-Mesh is an $n \times n$ mesh with a unique, partitionable bus for each row and column. Comparatively, the stronger RMesh model is an $n \times n$ mesh with bus segments between adjacent processors and arbitrarily complex connections between the segments are allowed. Since we may impose the additional restriction on our base model that both row and column bus operations may *not* occur within the same time step, even the variation of the RMesh model which does not allow crossover connections can utilize algorithms for the RCS-Mesh directly. In fact, unlike many models of parallel computation, the RCS-Mesh model has been, in effect, implemented as part of the MasPar MP series [17].

Further, the CRCW PRAM with n^2 processors and priority-write protocol can also trivially simulate the RCS-Mesh with constant time overhead, implying that any complexity lower bound for either the RMesh or the relevant PRAM model must also apply to the RCS-Mesh.

The RCS-Mesh is a very natural architecture for the study of graph problems. In [24], the authors have shown that a variety of problems — breadth-first search, k-connectivity, and small subgraph isomorphism — have simple, direct solutions for random graphs on the RCS-Mesh. Much of the simplicity is owed to the natural representation of a graph on an $n \times n$ mesh, where each vertex v_i is represented by the ith processor on the diagonal and each directed-edge (v_i, v_j) is represented by the processor in the ith row and jth column. For each vertex v_i, the ith column contains all the information relating to the in-edges of v_i; the ith row, all the information relating to the out-edges. We refer to this representation of input as having the graph *stored in the natural manner*, and we may refer to processors in the mesh arrangement by the corresponding graph element.

To define our other input assumption, let n be a positive integer and $0 < p < 1$ be a real constant. The *random graph on n vertices*, $G_{n,p}$, is a random object, where, for each distinct pair of vertices, v_1 and v_2, the edge (v_1, v_2) exists with independent probability p (the parameter p is referred to as the *edge probability* and is treated as part of the problem definition or, equivalently, as input). Modifications of this definition will produce random bipartite graphs, random directed graphs, and semi-random graphs (the probability of an edge existing is at least p). Thus, given a particular graph problem, the corresponding random graph problem provides a well-defined probabilistic input.

Within this context, we describe a routine forming the basis of $\log^\star$ paradigm for this architecture. A related paradigm exists for the CRCW PRAM with n processors [20]. For PRAM problems involving only n processors and $\Theta(n)$ memory cells, stepwise simulation of the PRAM $\log^\star$ paradigm by an $n \times n$ RCS-Mesh would yield the same time bounds. However, if n processors and $\omega(n)$ memory are involved then the RCS-Mesh may require significantly more time even if it can simulate the memory cells one per processor, because simulating the PRAM may force the RCS-Mesh to try to move m values from a $\sqrt{m} \times \sqrt{m}$ subsquare, which will take $\Theta(\sqrt{m})$ time. Thus real-time simulation of the $\Theta(\log^\star(n))$ expected time PRAM algorithms is not always possible; in particular, real-time simulation of the $\Theta(\log^\star(n))$ expected time PRAM Hamiltonian cycle algorithm of [19] is not possible.

Using our paradigm, the RCS-Mesh can find and mark a Hamiltonian cycle in a random graph or demonstrate that no such cycle exists in $\Theta(\log^\star(n))$ expected time. Immediate corollaries include results for directed Hamiltonian cycles, depth-first search trees, and exact matchings — including results for k-partite random graphs. Our method gives a simpler, easier-to-generalize approach to many other problems related to Hamiltonian cycles, which can be easily ported to other architectures.

The average case solutions of NP-complete problems (with varying probability distributions on input) have always been of great interest. Bollobás, Fenner, Frieze [5], demonstrated a serial algorithm which finds a Hamiltonian cycle in $G_{n,p}$ (if one exists) in expected polynomial time when $p \geq \frac{1}{2}$. Both Gurevich and Shelah [14] and Thomason [22] gave serial algorithms which run in linear expected time and find a Hamiltonian cycle (if one exists) for $p > 0$. Frieze [11], using $n \log^2 n$ processors and $\mathrm{O}((\log \log n)^2)$ expected time, and MacKenzie and Stout [19], using $n/\log^*(n)$ processors and $\Theta(\log^\star(n))$ expected time, demonstrated Hamiltonian Cycle algorithms for the CRCW PRAM. In the latter paper the expected running time $\Theta(\log^\star(n))$ was shown to be *optimal* for the construction of a Hamiltonian cycle, a maximal matching, or a spanning tree.

We demonstrate here that the RCS-Mesh, arguably a more feasible architecture than the PRAM, can accomplish many of the same tasks within the same expected running time; furthermore, as shown in [24], the problem of finding a breadth-first spanning tree can be solved in $\Theta(1)$ expected time on the RCS-Mesh. We also show that if the graph is Hamiltonian, the RCS-Mesh is capable of marking the cycle index with only constant additional time. This extension is beyond what has been previously shown for the PRAM model.

The PRAM results [19] depend upon base operations which the RCS-Mesh is incapable of performing in $\Theta(\log^\star(n))$ time; thus, the algorithm used for solving the Hamiltonian cycle problem in that reference could not be ported. The algorithm developed here is conceptually different. With minimal changes, it can be ported to a variety of architectures — including both the target architecture used in [19] with the same time bounds and related PRAM models with correspondingly altered time bounds. We believe that the RCS-Mesh-based method is simpler and easier to extend.

We also show that general structures—such as tori—can also be found/constructed within a random graph in $\Theta(\log^\star(n))$ time with very high probability. This kind of result has unexpected applications, and we provide an example: Given a directed random graph stored in the natural manner on a RCS-Mesh, one can decide whether or not the graph is strongly k-connected where $k = n^\epsilon$ for a fixed $\epsilon \in (0, 1)$, in $\Theta(1)$ expected time. This problem, when expressed for arbitrary graphs and values of k, is co-NP-complete.

Throughout this paper, we often provide results only for directed graphs. Most of the methods can be easily generalized to work with undirected graphs and other related kinds of input.

1.1 Probability

When an event E occurs with probability at least $p(n)$ in the execution of some algorithm on a problem of size n, we say that E occurs with *high probability* if, for some real values $r > 1$ and $\epsilon > 0$, $r^{(n^\epsilon)} \in \mathrm{O}((1 - p(n))^{-1})$.

For the graph problems we're about to consider, we will show a probabilistic algorithm $\mathcal{P}$ for the RCS-Mesh which *attempts* to solve the problem with a random graph as input and demonstrate that it succeeds (signaling failure otherwise) with high probability. In the event of failure, the algorithm will employ another approach to the problem, a routine $\mathcal{P}'$ which solves the problem within $f(n)$ expected time. When $f(n)$ is polynomial or mildly exponential, the expected additional time needed to handle failures is eventually dominated by a constant.

For problems such as the Hamiltonian cycle problem, the existence of a serial solution running in polynomial expected time for a semi-random graph is a known result — the second stage of the Gurevich and Shelah algorithm [14] suffices. Unfortunately, given that the first stage fails with $G_{n,p}$ as input, we can no longer assume that $G_{n,p}$ is a random graph. Borrowing another method from Gurevich and Shelah (also described in [14]), we can still construct the desired algorithm with $\mathcal{P}$ as a base, where the additional expected time needed to handle failures is $\Theta(1)$.

2 A $\log^\star$ Paradigm for the RCS-Mesh

Many sublogarithmic PRAM algorithms involve assigning processors to a rapidly decreasing set of problems, and, with a $\log^\star$ paradigm, the relative number of problems decreases exponentially. As noted before, for PRAM problems involving $\omega(n)$ memory cells direct stepwise simulation of the PRAM paradigm by the RCS-Mesh or RMesh may not be possible. Moreover, unlike the RMesh, the RCS-Mesh is incapable of counting in constant time, requiring that natural randomization techniques for the stronger model be adapted to work work with *estimates* and/or close bounds on certain quantities. Despite these handicaps, the RCS-Mesh architecture has a similar paradigm in that we may assign *rows of processors* to handle an exponentially decreasing number of problems.

In this section, we describe an algorithm which runs in $\Theta(\log^\star(n))$ time and solves a specific random graph problem on the RCS-Mesh with high probability. Although the problem is expressed in graph-theoretic terms in order to it make directly applicable to the problems considered in this paper, the general principles may be abstracted to other problems, allowing simple construction of algorithms which run in $\Theta(\log^\star(n))$ expected time on reconfigurable architecture. Specifically, we describe this paradigm relative to random graph problems as the randomized matching of vertices between two distinct sets, where the ratio of sizes is at least a constant K, K not dependent on n.

Problem 2.1 (Bipartite Semi-Matching) *Let $G = (V_1, V_2, E)$ be a bipartite directed semi-random graph (V_1 and V_2 are disjoint sets of vertices, E is the set of edges) such that the size of the second partition, V_2, is more than K times the size of the first, V_1, and the independent probability of any directed edge existing is p_1 if it leads from V_1 to V_2, p_2 if it leads from V_2 to V_1, and zero otherwise. Mark a subset E' of E such that*

1. *If $(v_1, v_2) \in E'$, then $(v_2, v_1) \in E'$.*
2. *For each v_1 in the smaller partition, there exists a unique v_2 such that $(v_1, v_2) \in E'$.*

In an instance of the above, we do not assume that the values of $|V_1|$ and $|V_2|$ are known initially; instead, we use the values of K and n to form, respectively, strict upper and lower bounds on those quantities — F_{ub} and P_{lb} — including a ϵ measure of strictness.

In the informal description of how our solution to the above is applied — in the context of finding Hamiltonian cycles — the larger partition V_2 corresponds to the set of "patches" and the smaller partition corresponds to the set of "flaws." These labels are sufficiently descriptive to warrant their employment here. The fixed constant K will be a lower bound on the ratio of patches to flaws, a requirement for our algorithm. Here, our solution is demonstrated with K at least $(e \ln 2)/(p_1 p_2)$ (initially). The weaknesses of the RCS-Mesh architecture — the inability to count, to assign rows arbitrarily, etc., make this bound necessary; however, the authors have developed nontrivial variations which show that this restriction can be lowered.

The core of our solution to the above is the routine presented as Algorithm 2.2 which, with high probability, matches a large number of vertices from the smaller set with unique members of the larger. Careful examination of relevant probabilities allows the bounds to be adjusted and thus yields a semi-matching problem where the ratio of sizes has been increased dramatically — such that $\Theta(\log^\star(n))$ repetitions of the routine will completely exhaust the smaller set with high probability. For brevity, define R by

$$R = F_{ub} \left(e^{\frac{p_1 p_2}{e}} \right)^{-\frac{P_{lb}}{F_{ub}}} .$$

Partial Uniform Selection

Step 1: Each processor within a column with a vertex in V_1 and a row with a vertex in V_2 chooses a random integer between 1 and F_{ub} (uniformly and independently of any other processor). It records a success if it has chosen 1.

Step 2: Each row corresponding to a vertex in V_2 is tested for the condition that exactly one success occurred. If true, the successful processor's id number is broadcast to the processor on the diagonal.

Step 3: With two time steps, the processor corresponding to a vertex in V_2 can verify whether or not the double-edge pair exists with respect to the vertex in V_1.

Step 4: Each of the vertices in V_1 marks the least–numbered distinguished in-edge (if it exists). Within constant time, it can then select *only* the elements corresponding to that match. All other distinguishing properties are erased.

Algorithm 2.2

Bipartite Semi-Matching

Step 1: Calculate F_{ub} and P_{lb} initially. Set $T = \sqrt{Plb}$.

Step 2: While F_{ub} is more than T, use the procedure described in Algorithm 2.2, removing and saving all successful matches, and resetting F_{ub} to $\lceil R \rceil$ and P_{lb} to $\left\lfloor \frac{1}{2} P_{lb} \right\rfloor$.

Step 3: Reset F_{ub} to T and use the procedure described in Algorithm 2.2, removing and saving all successful matches.

Step 4: If any vertices in V_1 have not been successfully matched, signal failure and terminate.

Algorithm 2.5

The value of R can be shown to be greater than (but relatively close to) the expected number of unmatched flaws remaining after an execution of Algorithm 2.2.

In the analysis of Algorithm 2.2, we may eliminate consideration of those vertices from the larger set, patches, which may have become biased during prior executions of the routine. Simply, all the directed edges will exist with the desired *independent* probability if we consider only those patches which have not attempted to test the existence of any edges as being available for subsequent iterations. From a Chernoff bound, we expect, with high probability, less than half of the rows representing vertices in the larger set to have the unique success described in Steps 1 and 2.

In a similar way, we may state two deeper results concerning the number of unmatched vertices remaining in V_1 after an execution of Algorithm 2.2.

Lemma 2.3 *Let F_r be such that $F_r \geq R$. With probability greater than $1 - e^{-F_r}$, the number of vertices in V_1 which were not matched to unique vertices in V_2 after an execution of Algorithm 2.2 is not more than $(1 - \epsilon)F_r$.*

Lemma 2.4 *With probability greater than $1 - R$, Algorithm 2.2 has matched every vertex v_1 in V_1 to a unique vertex v_2 in V_2 such that both directed edges, (v_1, v_2) and (v_2, v_1), exist.*

With these lemmas, we can now examine the main algorithm for solving the bipartite semi-matching problem when K is strictly greater than $(e \ln 2)/(p_1 p_2)$ (given as Algorithm 2.5).

Note that we avoid the transition between large and small deviations by careful adjustment of the bounds at the value T. Restating the relevant lemmas with this value in place, we have Lemma 2.3 demonstrating the high probability of the next bound being valid, and Lemma 2.4 demonstrating the high probability of success with the final use of the subroutine.

Moreover, the ratio of patches to flaws can be shown to be increasing (with high probability), and an examination of the implicitly iterated function demonstrates a fixed point; showing the necessity for the bound on K.

Theorem 2.6 *For an $n \times n$ RCS-Mesh, given an instance of Problem 2.1 with the graph G stored in the natural manner, the problem can be solved in $\Theta(\log^\star(n))$ time, correctly signaling failure or success and succeeding with high probability.*

3 Hamiltonian Cycles and Exact Matchings

Using the $\log^\star$ paradigm, we can now develop RCS-Mesh algorithms for random graph problems related to Hamiltonian cycles which run in $\Theta(\log^\star(n))$ expected time.

Our approach to random graph problems involving Hamiltonian cycles on the RCS-Mesh is based on the failure of a naively simple approach — testing the vertices in their natural order for the desired structure. Although we do not expect all the required edges to be present, with high probability, a large number may be expected to exist. For each missing edge, which we describe as a *flaw*, we try to locate a unique element from a set of potential *patches*. The critical step of matching flaws and patches will be accomplished by our solution to the bipartite semi-matching problem.

In the case of a simple Hamiltonian cycle on an undirected random graph when n is even, we may describe those odd-numbered vertices incident to missing edges as flaws, and all other odd-numbered vertices as potential patches. The mechanism of repair is to transpose vertices. Explicitly, we can transpose a flaw i and a patch j if neither would be classified as a flaw after the transposition.

With high probability, the original number of flaws will not exceed $(1+\epsilon)(1-p^2)n/2$ and the original number of patches will be at least $(1-\epsilon)p^2n/2$. Thus, when p is sufficiently large, we may apply our solution to the bipartite semi-matching problem immediately. In the general case, however, the ratio of patches to flaws will not necessarily exceed any given constant. The following lemma, due to Frieze, implies a solution:

Lemma 3.1 (Frieze [11]) *A random graph $G_{n,p}$ does* not *contain a Hamiltonian cycle with probability at most $n^2 (1-p)^{(n-1)}$.*

Simply, any subgraph of a random graph is also a random graph. We partition the RCS-Mesh into submeshes of size $X \times X$ (where X is a constant not dependent on n) and look at the submeshes along the diagonal. Each represents a random graph, $G_{X,p}$. Using the submeshes concurrently, we can search exhaustively for a Hamiltonian path in each subgraph. Since X is a constant, the time required is constant. Some submeshes will fail, but, if we've chosen X carefully, enough will succeed so that we may claim that the expected percentage of flaws is below any fixed constant. The lemma suggests how to choose the value of X. Other cases follow similarly. In the case when we are seeking a directed Hamiltonian Cycle in a directed random graph, we may use the probability function in the following corollary.

Corollary 3.2 *A directed random graph $G_{n,p}$ does* not *contain a directed Hamiltonian cycle with probability at most $n^2 (1-p^2)^{(n-1)}$.*

In this particular case, we may select our value of X so that $1/f(X) > (1+\epsilon)(e \ln 2/p^4+1)$ where ϵ is a positive fixed constant. (To make our calculations easier, we may also assume the X exceeds any constant.) Careful analysis of Chernoff bounds, even with overly pessimistic assumptions for intractable quantities, yields the desired ratio of patches to flaws with high probability.

After demonstrating some subtleties regarding information movement, we can prove the following (where the phrase *a simple Hamiltonian structure* includes directed Hamiltonian cycles, uni-directional Hamiltonian cycles, bi-directional Hamiltonian cycles and related structures such as exact matchings, etc.):

Theorem 3.3 (Enumerated Hamiltonian Cycle) *For an $n \times n$ RCS-Mesh, given a directed random graph $G_{n,p}$ stored in the natural manner, one can attempt to mark and enumerate the vertices and edges in a simple Hamiltonian structure or decide that no such structure exists in $\Theta(\log^\star(n))$ time, correctly signaling failure or success and succeeding with high probability.*

Utilizing the method of Gurevich and Shelah [14] and appending their routine in the event of failure, gives us the complete result as a corollary.

Corollary 3.4 (Enumerated Hamiltonian Cycle) *Given the same conditions as in Theorem 3.3, one can mark and enumerate the vertices and edges in a simple Hamiltonian structure or decide that no such structure exists in $\Theta(\log^\star(n))$ expected time.*

Also, since we expect success with high probability, we can obtain related results, such as finding and marking depth-first search trees (including the enumeration) in $\Theta(\log^\star(n))$ expected time.

Remark 3.5 *Continuing in this manner, it is possible to find exact matchings and Hamiltonian cycles in k-partite random graphs where all the partitions are of equal size and strongly ordered (vertex v_i is in partition i modulo k). Moreover, the same is possible for k-cyclic random graphs which are strongly ordered (the probability of an edge existing between vertices v_i and v_j is p if $j - i \equiv 1 \pmod{k}$, 0 otherwise). All of the above may be accomplished in $\Theta(\log^\star(n))$ expected time.*

4 General Structures and Other Applications

The term *structure* is highly overloaded; unfortunately, for brevity, we must avoid precisely formalizing the notion here. The notion with which we're specifically concerned we will merely describe as *simple structures*.

A directed Hamiltonian cycle is a particular example of a well-defined simple structure, a directed cycle involving n vertices. Another example based on cycles is that of a uni-directed Hamiltonian cycle — where we're concerned with the *nonexistence* of edges as well as their existence. Examples of simple structures include d-dimensional tori with all dimensions having the same extent, balanced binary trees, and cube-connected cycles.

A simple structure has three properties meriting some discussion. First, the structure has at most one associated graph with n vertices for each value of n; second, the degree of all graphs associated with the structure must be bounded by a fixed value; third, a single processor may calculate all the relevant aspects of an associated graph — edge existence, coloring functions, etc. for a fixed value of n within constant time.

The first allows the structure to be well-defined; the second is equivalent to many coloring properties (necessary and sufficient conditions for the algorithm to proceed); and the third merely subsumes preprocessing time. In fact, the critical preprocessing involves a coloring of the vertices with a constant number of colors where the percentage of vertices with each color is bounded below by a positive constant. The second property alone is equivalent to the existence of this coloring.

With some limitations, our approach to finding Hamiltonian cycles in random graphs may be extended to these simple structures, resulting in RCS-Algorithms which, with high probability, find the desired structure in a random graph within $\Theta(\log^\star(n))$ time.

Our strategy for finding simple structures in a directed random graph is to proceed sequentially through the vertex colors, fixing the labels or positions of vertices of each color relative to those that have been previously fixed. Thus, the algorithm may be described as a constant number of repetitions of a solution to bipartite exact matching.

The analysis proceeds by induction. We hypthesize that all vertices of colors less than k have already been fixed so that all edges between fixed vertices exist in accordance with the structure. Exactly the vertex processors which have color k are distinguished as being active. An active vertex v_i is a flaw if, for some fixed vertex v_j, there is an edge requirement between $\ell(v_i)$ and $\ell(v_j)$ which is not satisfied by the state of the corresponding edge, (v_i, v_j) in the random graph. A patch will be any active vertex which is not flaw. The mechanism for repair is to exchange labels if, after the exchange, neither vertex will be a flaw. After some tedious detail in showing that all the testing can be done in constant time, the following theorem can be demonstrated.

Theorem 4.1 (Simple Structures) *For an $n \times n$ RCS-Mesh, given a directed random graph $G_{n,p}$ stored in the natural manner, one can attempt to mark and enumerate the vertices and edges in a simple structure or decide that no such structure exists in $\Theta(\log^\star(n))$ time, correctly signaling failure or success and succeeding with a probability which eventually dominates $1 - r^n$ for some value of $r \in (0, 1)$.*

The higher probability is derived in the same manner that Corollary 3.2 was derived from Lemma 3.1. Since the probability that a particular vertex will have no edges is $\Theta(r^n)$ for some value of $r \in (0, 1)$, the probability bound in Theorem 4.1 is close to optimal for general structures whose elements are associated with connected graphs.

Importantly, the above result not only allows us to claim the ability to find, in $\Theta(\log^\star(n))$ expected time, regular tori (of any dimension), binary trees, cube-connected cycles, etc., within a random graph on the RCS-Mesh, but also it extends to virtually *any* structure of bounded degree.

4.1 Connectivity

Here we show an unusual application of our result. In [24], the authors have shown the following results:

Theorem 4.2 *For an $n \times n$ RCS-Mesh, given a random graph $G_{n,p}$ stored in the natural manner, for any fixed constant $\epsilon \in (0, 1)$, one can attempt to determine if $G_{n,p}$ is n^ϵ-connected in $\Theta(1)$ time, correctly signaling failure or success and succeeding with high probability.*

Theorem 4.3 *For an $n \times n$ RCS-Mesh, given a directed random graph $G_{n,p}$ stored in the natural manner, for any fixed constant $\epsilon \in (0, 1/2)$, in $\Theta(1)$ expected time one can decide if $G_{n,p}$ is strongly n^ϵ-connected.*

(As with the parameter p, ϵ is considered part of the problem definition rather than input; hence, we assume that it is fixed relative to n.)

The second theorem follows from appending an exhaustive test in the event of failure. When $\epsilon \geq 1/2$, the expected additional time in the event of failure is no longer constant. The routine for finding simple structures within a random graph will provide a replacement for the second stage of that algorithm when $1/2 \leq \epsilon < 1$. First, we require the following:

Claim 4.4 *Given a regular d-dimensional torus T on n vertices where d is greater than 2. The removal of less than n^ϵ vertices, where $\epsilon < d-2$, will leave a large connected component such that the number of vertices outside that component is $O(n^{(1-1/d)})$.*

The second stage requires two operations. First, we select the least integer d such that $(d-2)/d > \epsilon$, and set $X = (\lfloor \sqrt[d]{n} \rfloor)^d$. Using $p/4$ as the edge probability parameter, we attempt to find a regular d-dimensional torus in the last X vertices of the graph. If the attempt fails, the algorithm signals failure and terminates. Second, for every vertex, we count the number of incident edges, if any vertex obtains a sum less than $pn/4$, the algorithm signals failure and terminates.

The entire second stage takes polynomial time and succeeds with high probability. In the event of the failure of this stage, the exhaustive approach will add $\Theta(1)$ additional expected time. Specifically, we can demonstrate the following:

Theorem 4.5 (Connectivity) *For an $n \times n$ RCS-Mesh, given a directed random graph $G_{n,p}$ stored in the natural manner, for any fixed constant $\epsilon \in (0,1)$, one can attempt to determine if $G_{n,p}$ is n^ϵ-connected in polynomial time, correctly signaling failure or success and succeeding with a probability which eventually dominates $1-r^n$ for some value of $r \in (0,1)$.*

Now, with all the stages put together (and suitable adjustments for edge probability parameters), we can extend the range of ϵ in Theorem 4.3 to all of $(0,1)$.

5 Conclusion

We have described a $\log^\star$ paradigm tailored for reconfigurable architecture which can be implemented naturally on a modest (and commercially available) model, the RCS-Mesh. In addition, we have demonstrated the power of that paradigm, producing random graph algorithms with time bounds equivalent or superior to those known to exist for the CRCW PRAM architecture. This occurs despite the fact that the RCS-Mesh cannot do a real-time simulation of the CRCW PRAM, nor of many of its $\log^\star$ algorithms. While the algorithms for the RCS-Mesh have similarities with algorithms for the CRCW PRAM, they also have many critical differences to overcome the limitations of the weak RCS-Mesh model.

In particular, we have focused on problems related to Hamiltonian cycles. We have shown an algorithm which can find and mark a Hamiltonian cycle, giving each vertex an index indicating its position within the cycle, in $\Theta(\log^\star(n))$ expected time. Notably, this property has not been demonstrated for the relevant PRAM model. These results follow from a simple approach, a decomposition into patches and flaws, which offers immediate promise of portability as well as a large number (and a wide variety) of simple extensions. We have demonstrated some of these here; most notably, an unexpected relationship with a general connectivity question.

References

[1] D. Aldous. Approximate counting via markov chains. In *Probability and Algorithms*, pages 31–38. National Academy Press, Washington, D.C., 1992.

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

[3] P Beame and J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 83–93, 1987.

[4] B. Bollobás. *Random Graphs.* Academic Press, Inc., London, 1985.

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

[6] J. A. Bondy and U. S. Murty. *Graph Theory with Applications.* North-Holland, New York, 1976.

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

[8] E. Dahlhaus, P. Hajnal, and M. Karpinski. On the parallel complexity of hamiltonian cycle and matching problems on dense graphs. *Journal of Algorithms*, 15:367–384, 1993.

[9] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc.*, pages 69–81, 1952.

[10] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. In *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 165–169, 1982.

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

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.

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

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

[15] W. R. Hamilton. Letter to John T. Graves on The Icosian, 17 oct., 1856. In H. Halberstam and R. E. Ingram, editors, *The Mathemathical Papers of Sir William Rowan Hamilton*, volume 3 (Algebra), pages 612–625. Cambridge University Press, 1931.

[16] B. Jackson. Hamilton cycles in regular 2-connected graphs. *J. Comb. Theory (B)*, 29:27–46, 1980.

[17] R. E. Ladner, J. Lampe, and R. Rogers. Vector prefix addition on sub-bus mesh computers. In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, pages 387–396, 1993.

[18] J. E. Littlewood. On the probability in the tail of a binomial distribution. *Adv. Appl. Prob.*, 1:43–72, 1969.

[19] P. MacKenzie and Q. Stout. Optimal parallel construction of hamiltonian cycles and spanning trees in random graphs. In *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, pages 224–229, 1993.

[20] 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.

[21] P. Ragde. On the parallel simplicity of compaction and chaining. *Journal of Algorithms*, 14:371–380, 1993.

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

[23] B. F. Wang and G. H. Chen. Constant time algorithms for the transitive closure and some related problems on processor arrays with reconfigurable bus systems. In *IEEE Proc. Parallel and Distrib. Sys.*, pages 500–507, 1990.

[24] D. Van Wieren and Q. Stout. Random graph algorithms for the mesh with row and column subbuses. To Appear, 1994.