In *Proc. 1985 International Conference on Parallel Processing*, pp. 727–630.

Tree-Based Graph Algorithms for some Parallel Computers

(Preliminary Version)

Quentin F. Stout*
Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109 USA

Abstract

This paper gives several optimal mesh computer, VLSI, and pyramid computer algorithms for determining properties of an arbitrary undirected graph, where the graph is given as an unordered collection of edges. The algorithms first find spanning trees and then use them to determine properties of the graph. By using edges, instead of requiring an entire adjacency matrix, these algorithms use only $\Theta(e^{1/2})$ time on a 2-dimensional mesh, instead of the $\Omega(v)$ time required with matrix input. Further, the edge-based algorithms extend naturally to meshes of arbitrary dimension d, finishing in $\Theta(e^{1/d})$ time. All of the times are optimal, and the algorithms extend to VLSI and pyramid models.

1 Introduction

This paper gives a collection of optimal algorithms for determining various properties of an undirected graph, where the graph is given as an unordered collection of edges. This is the most flexible form of graph input and is usually the most space efficient, and, as will be shown, also yields the most time efficient algorithms for the computational models considered here. (In this paper, the space is either the number of processing elements or the area on a chip.) All of the algorithms are based on first finding a spanning tree for each connected component, and then using it to determine properties of the graph. A *spanning forest* consists of a spanning tree for each connected component.

The primary contribution of this paper is in producing a collection of optimal algorithms for the general edge input format. For serial computers, it has long been known that many graph properties can be efficiently determined by using spanning forests, particularly when the forest is generated by a depth-first search [Ta]. Depth-first search does not parallelize well [Re], but by alterating the algorithms often any spanning tree can be used. For parallel computers this approach has been used before [TV], and was utilized extensively in [AK] to give several 2-dimensional mesh computer algorithms for determining properties of graphs. However, the algorithms in [AK] required that one start with an adjacency or weight matrix, which implies that all algorithms must take $\Omega(v)$ time, where v is the number of vertices. For important classes such as planar graphs this input format uses the square of the area needed by edge input.

This paper gives algorithms for the problems considered in [AK], plus additional problems, but eliminates the requirement of matrix input. (The algorithms will also work with matrix input, or, indeed, with any standard form of graph input.) As a reviewer and others have noted, it is a fairly straightforward matter to convert the matrix algorithms in [AK] to edge-based ones requiring $\Theta(e^{1/d}\log e)$ time for a graph of e edges stored in a d-dimensional mesh of e processing elements. Further, with such input any algorithm must take $\Omega(e^{1/d})$ time, so the main contribution of this paper is in achieving optimality by eliminating the extra logarithmic factor. One important step is the optimal algorithm in [RS,St3] which finds spanning forests in $\Theta(e^{1/d})$ time, versus the well-known $\Theta(e^{1/d}\log e)$ algorithm of [NS]. Other tree operations are based on repeated use of path compression, random access read, and random access write [NS,MS1]. Most of these tree operations were introduced in [St1,St3], and all are optimal, finishing in $\Theta(e^{1/d})$ time.

Section 3 contains mesh algorithms for determining properties of trees, and Section 4 uses tree algorithms to derive graph algorithms for meshes. In Section 5 it is shown that the same algorithms are optimal for a physically realistic model of VLSI, and in Section 6 it is shown that they can be used to give good algorithms for a pyramid computer. Another use of the mesh algorithms appears in [MS3]. Because of space limitations, only sketches of proofs can be included. Complete proofs will appear in the final version.

2 Terminology

Let d and n be positive integers, where $n = m^d$ for some integer m. A *d-dimensional mesh (computer) of size* n consists of n *processing elements* (*PEs*) arranged in a d-dimensional

*Research supported by National Science Foundation Grant MCS-83-01019

integer lattice of edgelength m, where each PE has communication links to its $2d$ neighbors. (PEs along the sides have fewer links.) The mesh is usually considered to be an SIMD machine, although for the purposes of this paper it is irrelevant whether it is SIMD or MIMD. All operations, such as addition or exchanging a word of data with a neighbor, take unit time, and the word size is $\Omega(\log n)$. Some basic mesh algorithms appear in [AK,MS1,NS,RS,TK,Va].

By an algorithm being *optimal* we mean that no algorithm can be faster by more than a multiplicative constant. The optimality of mesh algorithms taking $\Theta\left(n^{1/d}\right)$ time follows from the fact that information going from one corner of a d-dimensional mesh of size n to the opposite corner must cross at least $d\left(n^{1/d}-1\right)$ wires. For each problem considered here, it is easy to show that there are instances in which data must travel from one corner to the opposite one.

Let n be a positive integer which is an integral power of 4. A (2-dimensional) *pyramid (computer) of size n* consists of layers of mesh computers, with additional links between the layers. The base is a mesh of size n, the next layer is a mesh of size $n/4$, then a layer of size $n/16$, and so on to the apex, which is a mesh of size 1. Notice that a pyramid of size n actually has $(4/3)n - (1/3)$ PEs. Each PE is connected to its four children on the layer below, and its parent on the layer above, as well as to its four neighbors on the same layer. The wordsize is again $\Omega(\log n)$, and operations take unit time. Some basic pyramid algorithms appear in [Mi,MS2,St2,Ag].

A graph will be given as a set of edges stored in an unsorted fashion one per PE in the mesh, or one per base PE in the pyramid. When analyzing times on a d-dimensional mesh of size n, d will be considered to be fixed and the time will be analyzed as a function of n. Not all papers analyze times in precisely this manner, but there are difficulties when varying d due to the fact that a PE in a d_1-dimensional mesh is different from a PE in a d_2-dimensional mesh when $d_1 \neq d_2$.

Only undirected graphs are considered in this paper. Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the *cyclic index* of G is the largest integer s such that V can be partitioned into sets $V_0, V_1, \ldots, V_{s-1}$, so that, for any edge $(x, y) \in E$, if $x \in V_i$ then $y \in V_{(i\pm 1) mod\ s}$. G is said to be *bipartite* if the cyclic index is even. An edge is a *bridge edge* if its removal increases the number of connected components, and a vertex is an *articulation point* if its removal (along with all incident edges) increases the number of components. G is *biconnected* if for any two disjoint vertices, there are at least two disjoint paths between them.

3 Tree Algorithms on the Mesh

The following result is used repeatedly:

3.1 Theorem [RS,St3] *Given a graph stored one edge per PE in a d-dimensional mesh of size n, in $\Theta\left(n^{1/d}\right)$ time a spanning forest can be found, where finding a spanning forest means that each PE has a flag, and the flag is set true if and only if the PE's edge is in the spanning forest. Further, this time is optimal.* □

If a forest of rooted trees is being stored one edge per PE, and if each PE knows which endpoint of its edge is closer to the root, then we say the computer is storing an *oriented forest*.

3.2 Theorem *Given an unoriented forest stored one edge per PE in a d-dimensional mesh of size n, and given a vertex in each tree which has been chosen to be the root, the forest can be oriented in $\Theta\left(n^{1/d}\right)$ time. Further, this time is optimal.*

Sketch of proof: The algorithm follows the basic outline of the algorithm in Theorem 3.1 [RS]. The "standard" parallel version of Sollin's component labeling algorithm, as in [HCS], is used for $d + 1$ iterations, dividing the number of unfinished "clubs" (in the terminology of [HCS]) by a factor of 2^{d+1}. These form "super vertices". Each quadrant finds a spanning forest for its super vertices, and then the spanning forests of the super vertices and the clubs are combined at the end. The [HCS] procedure gives each club a natural orientation. When finished, the super vertices form a tree which is then oriented by a recursive application of this algorithm. This orientation is combined with that of the clubs, rearranging part of each clubs' orientation. Only a single path in each club needs to change its orientation, and path compression techniques handle this easily. □

The next theorem shows that we can rapidly calculate some functions defined on oriented forests. This is done with a slight bit of generality, which unfortunately increases the notation. It may be useful to keep the example at the end of this paragraph in mind when examining the theorem. Let S be some set and let $*$ be an associative binary operation with identity on S which can be computed in $\Theta(1)$ time. Let $F = (V, E)$ be a forest of rooted trees, and let $L\colon V \to S$ be any map. Define $D : V \to S$ by

$$D(v) = *\{L(w)\colon w \text{ a descendant of } v\},$$

and define $U : V \to S$ by

$$U(v) = *\{L(w) : w \text{ an ancestor of } v\}.$$

(If a set is empty, then the value is the identity element for $*$, while if a set has just one element then the value is that element.) For example, if S is the natural numbers, $*$ is addition, and L is the constant 1 function, then $D(v)$ is the number of descendents of v, and $U(v)$ is the depth of v.

The following theorem is proven by an extensive use of path compression, random access reads, and random access writes.

3.3 Theorem *Given an oriented forest stored one edge per PE in a d-dimensional mesh of size n, given S, L, and $*$ as above, and given that a PE with edge (x, y) also contains $L(x)$ and $L(y)$, then in $\Theta(n^{1/d})$ time D and U can be calculated, meaning that a PE with edge (x, y) will also contain $D(x), U(x), D(y),$ and $U(y)$. Further, this time is optimal.* □

3.4 Corollary *Given an oriented forest stored one edge per PE in a d-dimensional mesh of size n, in $\Theta(n^{1/d})$ time each PE can determine the depth and height of each endpoint of the edge it contains, and can determine the number of nodes in their subtrees. Further, this time is optimal.* □

3.5 Corollary *Given an oriented forest stored one edge per PE in a d-dimensional mesh of size n, in $\Theta(n^{1/d})$ time the mesh can determine both a preorder numbering and postorder numbering for each tree. Further, this time is optimal.*

Proof: The string generation algorithm in [St1] assigns vertices to locations from which one can directly generate preorder or postorder numberings. (The algorithm was only given for a 2-dimensional mesh since the topological transformation only made sense for 2-dimensional data, but it extends naturally to other dimensions.) The algorithm in [St1] requires that the depth and size of the subtree of each vertex be known, which is provided by Corollary 3.4. □

4 Graph Algorithms for the Mesh

The following theorem was announced in [St3].

Theorem 4 *Suppose trees T1 and T2 are stored in a mesh of size n, with one edge of each per PE. Then in $\Theta(n^{1/d})$ time the mesh can decide if T1 and T2 are isomorphic. Further, this time is optimal. (This is actually two results: if the trees both have roots, then it is rooted isomorphism which is being decided, while otherwise it is unrooted isomorphism.)*

Sketch of proof: For rooted isomorphism, [St1] showed that the isomorphism could be decided in the indicated time if each PE knew the depth and subtree size of each of the edge endpoints stored in the PE. Corollary 3.4 shows that this information can be obtained for arbitrary trees.

For unrooted isomorphism the goal is to identify "natural" roots, reducing the problem to rooted isomorphism. For the root we use a node which minimizes the height of the resulting rooted tree. Such a node may not be unique (consider a tree of only two nodes), but there are at most two such nodes per tree. If each tree has a unique natural root, then apply the rooted isomorphism, while if each tree has two candidates then pick one of *T1*'s candidates as its root and try both ways of matching *T2*'s candidates. (If one tree has a unique candidate and one has two candidates then they are not isomorphic.)

To find the root, pick any node p as a root and apply Corollary 3.4. Each PE can now determine, for each vertex of its edge, if the vertex is on a path of maximal length originating from p. Either there are nodes which have two children on such a path, in which case the root is between p and the node of least depth with this property, or else there is only a single path of maximal length, in which case the root is along this path. In either case, a straightforward search can locate it. □

The next theorem follows the matrix algorithms of [AK] quite closely. Those algorithms repeatedly used trees and to determine properties of the graph, using several matrix operations. These operations are replaced by using the algorithms of Section 3 to generate the needed information.

Theorem 5 *Suppose a graph G is stored one edge per PE in a d-dimensional mesh of size n. Then in $\Theta(n^{1/d})$ time the mesh can:*

a) *decide if G is bipartite.*

b) *determine the cyclic index of G.*

c) *find all the bridge edges of G.*

d) *find all the articulation points of G.*

e) *decide if G is biconnected.*

Further, all these times are optimal. □

5 VLSI

A 2-dimensional mesh computer is one model of 2-dimensional computation, and VLSI is another. Actually, there are many different models of VLSI. See, for example, [Ja] for a discussion of some of these and their relationships to lower bounds for graph problems. We will consider a quite realistic model in which there is sufficient area to hold all of the information, the chip is a square with input/output ports only along its border, and information has a finite velocity, i.e., it takes $\Omega(s)$ time for information to travel along a wire of length s. (The finiteness of information velocity is often ignored. See [BPP] for a discussion of information velocity.)

With these assumptions it is quite easy to show that the minimal time to solve any problem considered herein is at least the square root of the area. Notice that the 2-dimensional mesh algorithms in this paper solved all the problems in time which was the square root of the number of PEs. There are, however, two principle differences between the mesh and our VLSI model:

i) each edge, and each PE, requires $\Theta(\log n)$ area.

ii) operations need $\Theta\big(\log^{1/2} n\big)$ time.

For all of the problems considered in this paper, the input size is $\Theta(n * \log n)$ bits, i.e., it is a factor of $\log n$ larger than the "area" (number of PEs) of the 2-dimensional mesh. Further, the VLSI algorithms will be a factor of $\log^{1/2} n$ slower than the 2-dimensional mesh algorithms. Thus we will still have $T = A^{1/2}$, which is optimal. Therefore all of the 2-dimensional mesh algorithms in this paper also yield optimal VLSI algorithms, at least for our model of VLSI. (Further, if 3-dimensional VLSI ever becomes a reality, the 3-dimensional mesh algorithms yield optimal 3-dimensional VLSI algorithms.)

6 Pyramids

The pyramid computer algorithms are a combination of the mesh algorithms and the following result.

Theorem 6 [Mi,MS2] a) *Suppose a graph G with v vertices is stored one edge per base PE of a pyramid computer of size n. Then in $\Theta\big(\log(n) + v^{1/2}\big[1 + \log(n/v)\big]^{1/2}\big)$ time a spanning forest of G can be determined.*

b) *Suppose the adjacency matrix of a graph G is stored in natural order, one entry per PE, in the base of a pyramid computer of size n. Then in $\Theta\big(n^{1/4}\big)$ time a spanning forest of G can be determined.* □

Whether the pyramid has an adjacency matrix or unsorted edges in its base, the mesh algorithms are tranported to the pyramid by finding a spanning tree and then moving its edges to the highest level with v PEs. At that level, mesh algorithms taking $\Theta\big(v^{1/2}\big)$ time can be run, in some cases requiring additional information from the base. The movement up and down uses pyramid data movement operations introduced in [Mi,MS2]. Using this general transference technique from the mesh to the pyramid, we obtain the following results first noted in [Mi,MS2].

Theorem 7 *Suppose a graph G with v vertices is stored one edge per base PE of a pyramid computer of size n. Then in $\Theta\big(\log(n) + v^{1/2}\big[1 + \log(n/v)\big]^{1/2}\big)$ time the pyramid can:*

a) *decide if G is bipartite.*

b) *determine the cyclic index of G.*

c) *find all the bridge edges of G.*

d) *find all the articulation points of G.*

e) *decide if G is biconnected.* □

Theorem 8 *Suppose the adjacency matrix of a graph G is stored in natural order, one entry per PE, in the base of a pyramid computer of size n. Then in $\Theta\big(n^{1/4}\big)$ time the pyramid computer can:*

a) *decide if G is bipartite.*

b) *determine the cyclic index of G.*

c) *find all the bridge edges of G.*

d) *find all the articulation points of G.*

e) *decide if G is biconnected.* □

It seems that the algorithms in Theorems 7 and 8 are optimal, but currently the best lower bound known for pyramid computer solutions of these problems is a factor of $\log^{1/2} n$ lower [MS2].

7 Conclusion

It has been shown that, given an arbitrary undirected graph specified as an unsorted collection of edges, a d-dimensional mesh computer can determine many properties of the graph in time which grows as the communication diameter of the mesh, which is the best that one can ask for. This is done by first constructing a spanning forest, and then determining properties of the forest and of the graph. This "tree algorithms yield graph algorithms" approach has been used previously (e.g., [Ta,TV]). For 2-dimensional mesh computers it had been previously used by [AK] for some of the problems considered here, but that paper required that the entire adjacency matrix of the graph be given. This may significantly increase the size of the mesh needed, and guarantees that each algorithm must use time linear in the number of vertices, rather than the square root of the number of edges. Of course, the edge algorithms given here work if given an adjacency matrix, but for important classes of graphs such as planar graphs, trees, forests, trivalent graphs, and functional digraphs, the number of edges grows as the square root of the size of the adjacency matrix. In general, edge input is the most efficient, and our contribution has been to optimize a collection of algorithms for this input.

It was further shown that all the optimal mesh algorithms give optimal VLSI algorithms. Finally, they also form the core of efficient algorithms for the pyramid computer. It is interesting that the mesh's edge-input graph algorithms are needed for the pyramid even when the pyramid starts with an adjacency matrix. The reason for this is that the pyramid is faster than the 2-dimensional mesh only when the amount of data remaining can be rapidly reduced, for then this data can be moved up the pyramid. Edge-based mesh algorithms permit a more concise representation of spanning forests which

can be moved up to the middle level of the pyramid, but matrix-based ones do not.

REFERENCES

[Ag] A. Aggarwal, A comparative study of X-tree, pyramid, and related machines, *24th Symp. on Found. Comp. Sci.* (1984), 89–99.

[AK] M.J. Attalah and S.R. Kosaraju, Graph problems on a mesh-connected processor array, *J. ACM 31* (1984), 649–667.

[BPP] G. Bilardi, M. Pracchi, and F.P. Preparata, A critique and appraisal of VLSI models of computation, *Proc. CMU Conf. VLSI Syst. Comp.*, H.T. Kung, R. Sproull, and G. Steele, Eds. (1981), 81-88.

[HCS] D.S. Hirschberg, A.K. Chandra, and D.V. Sarwate, Computing connected components on parallel computers, *C. ACM 22* (1979), 461–464.

[Ja] J. Ja'Ja, the VLSI complexity of selected graph problems, *J. ACM 31* (1984), 377-391.

[Mi] R. Miller, *Pyramid Computer Algorithms*, Ph.D. thesis, State University of New York at Binghamton, 1984.

[MS1] R. Miller and Q.F. Stout, Geometric algorithms for digitized pictures on a mesh-connected computer, *IEEE T. on PAMI* 7 (1985), 216–228.

[MS2] R. Miller and Q.F. Stout, Data movement techniques for the pyramid computer, submitted.

[MS3] R. Miller and Q.F. Stout, Varying diameter and problem size in mesh-connected computers, *Proc. 1985 Int'l. Conf. Parallel Proc.*, 697–699.

[NS] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM J. Computing* 9 (1980), 744–757.

[Re] J. Reif, Depth first search is inherently sequential, *Info. Proc. Letters* 20 (1985), 229–234.

[RS] J. Reif and Q.F. Stout, Optimal component labeling algorithms for mesh computers and VLSI, to appear.

[St1] Q.F. Stout, Topological matching, *Proc. 15th ACM Symp. Theory of Comput.* (1983), 24-31.

[St2] Q.F. Stout, Sorting, merging, selecting, and filtering on tree and pyramid machines, *Proc. 1983 Int'l. Conf. Parallel Proc.*, 214–221.

[St3] Q.F. Stout, Optimal component labeling algorithms for mesh-connected computers and VLSI, *Abstracts AMS* 5 (Jan. 1984), 148.

[Ta] R.E. Tarjan, Depth-first-search and linear graph algorithms, *SIAM J. Computing* 1 (1972), 146–160.

[TV] R.E. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, *Proc. 25th Symp. Found. Comp. Sci.* (1984), 12–20.

[TK] C.D. Thompson and H.T. Kung, Sorting on a mesh-connected parallel computer, *C. ACM 20* (1977), 263–271.

[Va] F.L. Van Scoy, The parallel recognition of classes of graphs, *IEEE T. Computers* 29 (1980), 563–570.