1983

Parallel Strong Orientation of an Undirected Graph

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
83-441

Parallel Strong Orientation of an Undirected Graph

Mikhail J. Atallah
Department of Computer Science
Purdue University
West-Lafayette, IN 47907

Parallel computation, graph algorithms

1. Introduction

There has recently been an increasing interest in designing parallel solutions to graph problems, and some of the proposed solutions differ radically from the standard sequential algorithms (e.g. [H78], [SJ81], [JS82], ...). Various models of computation have been proposed: The one we use here is the *shared memory model*, where the processors operate synchronously and share a common memory, and read conflicts are allowed but write conflicts are not allowed (i.e. no two processors should simultaneously attempt to write in the same memory cell). The graph problem we consider is the following one: Given a connected, bridgeless undirected graph, assign directions to its edges so that the resulting digraph is strong (recall that a bridge in a connected undirected graph is an edge whose removal disconnects the graph, and that a directed graph is strong iff for every two vertices u,v there is a $u-v$ directed path). It is well known that such a *strong orientation* exists iff the original undirected graph is connected and bridgeless. This problem can easily be solved sequentially in $O(|E|)$ time: Simply find a depth-first spanning tree and orient its edges in the father-to-child direction, and orient the non-tree edges in the descendant-to-ancestor direction (the resulting digraph can easily be shown to be strong). However, our goal is to design an $O(\log^2 n)$ parallel solution to this problem, and finding a depth-first-tree in $O(\log^2 n)$ parallel time seems exceedingly hard. The

solution we outline in the next section makes use of a spanning tree (not necessarily depth-first) which can be found in $O(\log^2 n)$ using $O(n^2)$ processors [SJ81]. Before proceeding, we introduce some terminology and a few conventions. Throughout, we assume that the original undirected graph $G=(V,E)$ is connected and bridgeless (i.e. it has a strong orientation), that $|V|=n$ and $|E|=m$, and that all logarithms are to the base 2. Our exposition will concentrate on the algorithmic aspect and we often refrain from giving the full implementation details (such details would not be particularly enlightening and may actually obscure the basic simplicity of the main idea).

2. The algorithm

Intuitively, the algorithm works as follows: If we let T be a spanning tree of the graph G, then every edge $(i,j)\in E-T$ defines a fundamental cycle in G which consists of edge (i,j) followed by the $j-i$ path in T. There are q such cycles (where $q=|E-T|$): List them (arbitrarily) as a sequence $C_1, C_2, \cdots, C_q$ and consider that C_i has *priority* i (so that C_q has highest priority). Note that a given edge of T may belong to many such cycles, and we say that these cycles are *competing* for that edge. Now, there are two possibilities for orienting the edges on C_i in order to turn it into a directed cycle: Arbitrarily pick one of these two possible orientations as the *official* orientation of C_i (loosely speaking, this represents the way that cycle C_i "wishes" the edges on it were oriented). However, for a given edge $e\in T$, the C_i's competing for it need not agree on which of the two possible ways e should be oriented, and therefore we decide that of all the C_i's competing for e, the winner is the one having highest priority: e gets oriented according to the official orientation of the highest-priority cycle competing for it. (Note that C_q will manage to have all the undirected edges belonging to it oriented according to its own official orientation, since it has highest priority). The digraph D resulting from this algorithm turns out to be strong.

A direct implementation of the above algorithm would result in an $O(\log^2 n)$ running time on $O(n^4)$ processors. In order to decrease the number of processors needed, the actual algorithm we describe is slightly different from the above one: We avoid the explicit computation of the C_i's as sets of edges or vertices. Instead, for every fundamental cycle, we let its (unique) edge that belongs to $E-T$ act as its *representative*. More specifically, if a certain edge $e \in E-T$ is the (lexicographically) k^{th} smallest edge in $E-T$, then we choose to give priority k to the fundamental cycle that e represents (i.e. e represents C_k). By convention, we write an edge (i,j) as $(Min\{i,j\},Max\{i,j\})$ before lexicographically comparing it to another edge.

Even though we will still mention the C_k's and their respective priorities, it is important to remember that they are not explicitly computed by the algorithm.

The various steps of the algorithm are listed below:

1. Let T be an undirected spanning tree of the graph G, and let H be a directed version of T (i.e. one where the edges of T are oriented in the father-to-son direction). Finding T and H takes $O(\log^2 n)$ time with $O(n^2)$ processors [SJ81].

Note: The fact that an edge of T appears in H with a father-to-son direction does *not* mean that that is the way it will be oriented in the desired directed version of G.

2. Compute the transitive closure of H (call it H^*). This can be done in $O(\log n)$ time with $O(n^3)$ processors [S77].

3. For every $(i,j) \in E-T$, compute the lowest common ancestor (in H) of vertices i and j, call it $LCA(i,j)$. (Recall that the lowest common ancestor of two nodes i and j in a tree is ancestor of i and of j while none of its children is ancestor of both i and j.)

This step can be done in $O(\log n)$ time with $O(n^3)$ processors [S77].

4. For every edge $(a,b)\in T$, where a is the father of b, choose the lexicographically largest edge in $E-T$ which represents a cycle competing for (a,b) and call the chosen edge the *master* of (a,b). An edge $(i,j)\in E-T$ represents a cycle which is competing for (a,b) iff $LCA(i,j)$ is an ancestor of a and in addition b is ancestor of i or of j (both conditions can be tested in $O(1)$ time since we already have computed $LCA(i,j)$ and H^*). Deciding which edges of $E-T$ represent cycles competing for (a,b) can therefore be done in $O(1)$ time with $O(q)$ processors, and choosing the lexicographically largest such edge requires a further $O(\log n)$ time with $O(q)$ processors.

Since there are $n-1$ tree edges, the total cost of this step is $O(\log n)$ time on $O(nq)$ processors.

Comment: The master of $(a,b)\in T$ is the representative of the highest-priority fundamental cycle containing (a,b), the cycle according to whose "wish" edge (a,b) will later be directed. Of course every $(a,b)\in T$ has a master since otherwise it would be a bridge.

5. Assign (arbitrarily) a direction to every edge in $E-T$. This takes $O(1)$ time and $O(q)$ processors.

Comment: The direction assigned to an edge $(i,j)\in E-T$ implicitly induces an orientation of the undirected cycle C_k which (i,j) represents, and we consider this to be the official orientation of C_k, so that C_k "wishes" all the tree edges on it were oriented accordingly. An edge (a,b) on C_k which is assigned a direction consistent with C_k's official orientation will be said to *agree with* C_k. (Obviously C_k's representative always agrees with it.)

6. To every edge $(a,b)\in T$, assign a direction that agrees with the fundamental cycle represented by its master. This is done as follows. Let $(i,j)\in E-T$ be the

master of $(a,b)\in T$, and without loss of generality, assume that a is father of b and that (i,j) was given the i-to-j direction during step 5 of the algorithm. One of two cases can occur:

(i) If b is ancestor of i, then the a-to-b direction is assigned to (a,b).

(ii) If b is ancestor of j, then the b-to-a direction is assigned to (a,b).

Note that which of cases (i) or (ii) holds can be decided in $O(1)$ time, since we already have H^*. Therefore, this step can be done in $O(1)$ time with $O(n)$ processors.

(End of algorithm)

Theorem The above algorithm runs in time $O(\log^2 n)$ on $O(n^3)$ processors, and produces a strong orientation of the input graph G if G has such an orientation.

Proof: The time and processor bounds easily follow from the above outline of the algorithm. To prove correctness, note that the algorithm assigns a direction to every edge of G (because G is bridgeless), and let D be the digraph resulting from the algorithm. Since the undirected version of D is connected, it suffices to show that every arc $(i\rightarrow j)$ of D is on a directed cycle, i.e. that for every arc $(i\rightarrow j)$ of D there is in D a (directed) $j-i$ path. We prove this by backward induction on k, where C_k is the highest-priority fundamental cycle containing edge (i,j) (recall that C_k has priority k). If $k=q$ then clearly the undirected $j-i$ path in C_q is also a directed $j-i$ path in D (since C_q has the highest priority). If $k<q$, then examine the (undirected) $j-i$ path in C_k and find a directed $j-i$ path in D in the following way: If the undirected edge you are currently examining is oriented in D in the same direction you are going then just follow it (it agrees with C_k), otherwise that edge (call it (a,b)) has a b-to-a direction that agrees with some C_l which contains (a,b) and has larger priority than C_k, i.e. $l>k$ and therefore by the induction hypothesis there is in D an $a-b$ directed path which

you can follow. Therefore it is possible to go in D from j to i, which completes the correctness proof. ▪

3. Conclusion

We gave an $O(\log^2 n)$ time algorithm for strongly orienting a connected bridgeless undirected graph. The technique used was that of assigning "priorities" to a number of (synchronous) processes that are "competing" for resources. In this case the "processes" were the fundamental cycles and the "resources" were the edges to be directed, but we suspect that the same technique can be applied in designing parallel algorithms for solving other problems as well. Finally, it is worth mentioning that the algorithm presented here can be implemented to run in time $O(\log n)$ if many processors were allowed to simultaneously attempt to write in the same memory cell and the "write conflicts" were resolved in any reasonable way, such as: Only one processor succeeds in writing but we do not know which one, or only the processor with the smallest index succeeds, or ... (the details of how this can be done involve no new ideas and are therefore omitted).

Acknowledgement The author is grateful to Susanne Hambrusch and Greg Frederickson for stimulating discussions, and to the referee for helpful comments.

References

[H78] D.S. Hirschberg, "Parallel Algorithms for the Transitive Closure and the Connected Components Problems", *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pp. 55-57, 1976.

[JS82] J. Ja'Ja' and J. Simon, "Parallel Algorithms in Graph Theory: Planarity Testing", *SIAM Journal on Computing*, pp. 314-328, 1982.

[P78] F.P. Preparata, "New Parallel Sorting Schemes", *IEEE Transactions on Computers*, pp. 669-673, 1978.

[S77] C. Savage, "Parallel Algorithms for Graph Theoretic Problems", Ph.D. Thesis, CSL Report ACT-4, Coordinated Science Laboratory, University of Illinois, August, 1977.

[SJ81] C. Savage and J. Ja'Ja', "Fast, Efficient Parallel Algorithms for some Graph Problems", *SIAM Journal on Computing*, pp. 682-691, 1981.