Minimum-Cost Spanning Tree as a Path-Finding Problem

Bruce M. Maggs Serge A. Plotkin

Laboratory for Computer Science
MIT
Cambridge MA 02139

July 8, 1994

Abstract

In this paper we show that minimum-cost spanning tree is a special case of the closed semiring path-finding problem. This observation gives us a non-recursive algorithm for finding minimum-cost spanning trees on mesh-connected computers that has the same asymptotic running time but is much simpler than the previous recursive algorithms.

1 Introduction

In this paper we show that minimum-cost spanning tree is a special case of the closed semiring path-finding problem [1, sections 5.6–5.9]. For a graph of n vertices, the path-finding problem can be solved sequentially in $O(n^3)$ steps by a dynamic programming algorithm [7, 12] of which the algorithms of Floyd [5] and Warshall [15] are special cases. This dynamic programming algorithm has a well known $O(n)$ step implementation on an $n \times n$ mesh-connected computer [2, 3, 4, 6, 13].

Previously known minimum-cost spanning tree algorithms for the mesh [2, 11] are based on the recursive algorithm of Boruvka (also attributed to Sollin) [14, pp. 71–83], which is complicated to implement. For example, the algorithm of [2] achieves $O(n)$ steps by reducing the fraction of the mesh in use by a constant factor at each recursive call. The dynamic programming algorithm has the same asymptotic running time but is much simpler.

[0]This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014–80–C–0622 and by the Office of Naval Research under Contract N00014–86–K-0593. Bruce Maggs is supported in part by an NSF graduate fellowship.

The rest of this paper consists of two short sections. In section 2 we show how to cast minimum-cost spanning tree as a path-finding problem. In section 3, we briefly describe an $O(n)$ step mesh algorithm to solve the problem.

2 Minimum-cost spanning tree

In this section we define the minimum-cost spanning tree problem and a related path-finding problem. We give a recurrence for solving the path-finding problem via dynamic programming. We then prove that the solution to the path-finding problem contains the solution to the minimum-cost spanning tree problem.

Given an n-node connected[1] undirected graph $G = (V, E)$, where V is the set $\{1, \ldots n\}$, and where each edge $\{i, j\}$ in E has cost $C_{ij}^0 = C_{ji}^0$, the minimum-cost spanning tree problem is to find a subgraph that connects the vertices in V such that the sum of the costs of the edges in the subgraph is minimum. We assume that the edge costs are unique. (If not, lexicographical information can be added to make them unique.) For convenience, we also assume that if $\{i, j\}$ is not in E then it has cost $C_{ij}^0 = C_{ji}^0 = \infty$.

The path-finding problem is to compute the cost C_{ij}^k for each $1 \leq i, j, k \leq n$ of the shortest (lowest-cost) path from i to j that passes through vertices only in the set $\{1, \ldots, k\}$, where *the cost of a path is defined to be the highest cost of any edge on the path.* For any i and j, the shortest path from i to j with no intermediate vertex higher than k either passes through k or does not. In the first case, the cost of the shortest path from i to j is either the cost of the shortest path from i to k or the cost of the shortest path from k to j, whichever is higher. In the second case, we have $C_{ij}^k = C_{ij}^{k-1}$. Thus, C_{ij}^k can be computed by the recurrence

$$C_{ij}^k = \min\{C_{ij}^{k-1}, \max\{C_{ik}^{k-1}, C_{kj}^{k-1}\}\}.$$

The following theorem shows that the unique minimum-cost spanning tree can be recovered from the costs of the shortest paths.

Theorem 1 *An edge $\{i, j\}$ is in the unique minimum-cost spanning tree if and only if $C_{ij}^0 = C_{ij}^n$.*

[1]For simplicity, we assume that the graph is connected. The same technique will find a minimum-cost spanning forest of a disconnected graph.

Proof: The proof has two parts. We first show that if $\{i,j\}$ is a tree edge then $C^0_{ij} = C^n_{ij}$. We then show that if $C^0_{ij} = C^n_{ij}$ then the edge $\{i,j\}$ is in the tree. First, assume that $\{i,j\}$ is a tree edge, but that $C^0_{ij} \neq C^n_{ij}$. Consider the cut of the graph that $\{i,j\}$ crosses, but no other tree edge crosses. Since $C^0_{ij} \neq C^n_{ij}$, there must be some path from i to j whose highest-cost edge has cost $C^n_{ij} < C^0_{ij}$. Hence, every edge on this path has cost less than C^0_{ij}. This path must cross the cut at least once. Replacing the edge $\{i,j\}$ by any edge on the path that crosses the cut reduces the cost of the tree, a contradiction. Conversely, assume that $C^0_{ij} = C^n_{ij}$, but that $\{i,j\}$ is not a tree edge. Adding the edge $\{i,j\}$ to the tree forms a cycle whose highest-cost edge costs more than than C^0_{ij}. Replacing this edge by $\{i,j\}$ yields a tree with smaller cost, a contradiction. ∎

3 Implementation on a mesh-connected computer

In this section we give a short description of an $O(n)$ step algorithm for solving the minimum-cost spanning tree problem on an $n \times n$ mesh-connected computer. We assume that the diagonal element in each mesh row can broadcast a value to the other elements of the row in a single step. This type of broadcast can be simulated by a mesh without this capability by slowing the algorithm down by a constant factor [8, 9, 10]. The algorithm proceeds as follows. We assume that the input graph is given in the form of a matrix of edge costs C^0 which enters row-by-row through the top of the mesh. Matrix row i is modified as it passes over rows 1 through $i-1$ and is stored when it reaches mesh row i. When matrix row i passes over mesh row k, the value C^{k-1}_{ik} is broadcast right and left from the diagonal cell (k,k). Each cell $(k,j), 1 \leq j \leq n$ knows the value of C^{k-1}_{kj} and computes

$$C^k_{ij} = \min\{C^{k-1}_{ij}, \max\{C^{k-1}_{ik}, C^{k-1}_{kj}\}\}.$$

which is passed down to the next mesh row. After reaching mesh row i, matrix row i stays there until each matrix row l, $i < l \leq n$, above it has passed over it and then continues to propagate down, passing over the rest of the matrix rows. The output matrix C^n exits row-by-row from the bottom of the mesh. By theorem 1, the adjacency matrix of the minimum-cost spanning tree can be constructed by comparing the input and output matrices.

4 Acknowledgments

We would like to thank Tom Leighton and Charles Leiserson for their helpful suggestions.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[2] M. J. Atallah and S. R. Kosaraju. Graph problems on a mesh-connected processor array. *Journal of ACM*, 31(3):649–667, July 1984.

[3] T. W. Christopher. *An implementation of Warshall's algorithm for transitive closure on a cellular computer.* Technical Report 36, Inst. for Comp. Research, Univ. of Chicago, Chicago, IL, 1973.

[4] E. Dekel, D. Nassimi, and S. Sahni. Parallel matrix and graph algorithms. *SIAM Journal on Computing*, 10(4):657–675, November 1981.

[5] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[6] L. J. Guibas, H. T. Kung, and C. D. Thompson. Direct VLSI implementation for combinatorial algorithms. In *Proc. of the Caltech Conference on Very Large Scale Integration*, pages 509–525, January 1979.

[7] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41, Princeton University Press, 1956.

[8] F. T. Leighton. *Introduction to the Theory of Networks, Parallel Computation and VLSI Design.* Unpublished manuscript.

[9] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimization of synchronous circuitry by retiming. In *Third Caltech Conference on VLSI*, pages 87–116, March 1983.

[10] C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, 1(1):41–46, Spring 1983.

[11] K. N. Levitt and W. H. Kautz. Cellular arrays for the solution of graph problems. *Communications of the ACM*, 15(9):789–801, September 1972.

[12] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. on Electronic Computers*, 9(1):39–47, 1960.

[13] F. L. Van Scoy. The parallel recognition of classes of graphs. *IEEE Transactions on Computers*, C-29(7):563–570, July 1980.

[14] R.E Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, PA, 1983.

[15] S. Warshall. A theorem on boolean matrices. *Journal of ACM*, 9(1):11–12, 1962.