ELSEVIER

Information Processing Letters 56 (1995) 83-87

Information Processing Letters

An O(log n) parallel algorithm for constructing a spanning tree on permutation graphs ☆

Yue-Li Wang *, Hon-Chan Chen, Chen-Yu Lee

National Taiwan Institute of Technology, Taipei, Taiwan, ROC

Received 10 August 1994; revised 17 February 1995
Communicated by K. Ikeda

Abstract

Let $G=(V, E)$ be a graph with n vertices and m edges. The problem of constructing a spanning tree is to find a connected subgraph of G with n vertices and $(n-1)$ edges. For a weighted graph, the minimum spanning tree problem can be solved in $O(\log m)$ time with $O(m)$ processors on the CRCW PRAM, and for an unweighed graph, the spanning tree problem can be solved in $O(\log n)$ time with $O(n+m)$ processors on the CRCW PRAM. In this paper, we shall propose an $O(\log n)$ time parallel algorithm with $O(n/\log n)$ processors on the EREW PRAM for constructing a spanning tree on an unweighted permutation graph.

Keywords: Parallel algorithms; Spanning tree; Permutation graphs; Graph theory; EREW computational model

1. Introduction

Let $G=(V, E)$ be a graph, $w(e)$ be the weighting function of the edges of G, where V and E are the vertex and edge sets, respectively. Every connected graph G contains a spanning subgraph that is a tree, called a *spanning tree* [6]. Typically, there are many different spanning trees in a connected graph, and for a spanning tree there are some properties which are described as follows:

The following are equivalent on a graph $T=(V, E)$, where n is the number of vertices and m is the number of edges.

(1) The graph T is a tree.
(2) The graph T is connected and $m=n-1$.
(3) Every pair of distinct vertices of T is joined by a unique path.
(4) The graph T is acyclic and $m=n-1$.

If there is a weight for each edge of G, then the minimum spanning tree problem (MST) is to find a spanning tree with the property that the sum of the weights of all the edges is the minimum among those spanning trees of G. Algorithms for the minimum spanning tree problem date back to the early work of Kruskal [9] and Prim [12]. In the last two decades, the complexity of these sequential algorithms has been reduced. Yao [15] provided an $O(m \log\log n)$ algorithm for a network with n vertices and m edges. Fredman and Tarjan [3] improved upon this

☆ This work was supported by the National Science Council, Republic of China, under contract NSC-84-2213-E-011-008.
* Corresponding author. Email: ylwang@cs.ntit.edu.tw.

0020-0190/95/$09.50
SSDI 0020-0190(95)00125-5

bound with an $O(m\beta(m, n))$ procedure ($\beta(m, n) = \min\{i \mid \log^{(i)} n \leqslant m/n\}$ where $\log^{(i)}(n)$ is the iterated logarithm). Gabow et al. [4] gave a further improvement. Good descriptions of MST algorithms appear in [1] and [14]. The well-known parallel algorithm to solve minimum spanning tree problem for a weighted graph takes $O(\log m)$ time with $O(m)$ processors on the CRCW PRAM (Concurrent-Read-Concurrent-Write Parallel Random Access Machine) computational model [13]. Moreover, for an undirected unweighted graph, the problem of constructing a spanning tree can be solved in $O(\log n)$ time with $O(n+m)$ processors on CRCW PRAM by the algorithm for eliminating cycles [8].

In this paper we consider the problem of constructing a spanning tree for a permutation graph. For simplicity, we only consider the case of a connected permutation graph with n vertices and m edges. We present a parallel algorithm which runs in $O(\log n)$ time with $O(n/\log n)$ processors, and our approach uses the EREW PRAM (Exclusive-Read-Exclusive-Write Parallel Random Access Machine) computational model.

Let the sequence $P = [p_1, p_2, \ldots, p_n]$ be a permutation of the numbers $1, 2, \ldots, n$. Then the *permutation graph* of P, $G(P) = G(V, E)$, is defined as follows:

$$V = \{1, 2, \ldots, n\},$$

$$E = \left\{(i, j) \mid (i-j)\left(p_i^{-1} - p_j^{-1}\right) < 0\right\}.$$

p_i^{-1} is the position in the sequence where the number i can be found. In a more pictorial way, we write the numbers $1, 2, \ldots, n$ horizontally from left to right. In this matching diagram the line connecting the two i's intersects the line connecting the two j's if and only if (i, j) is in E [2,5,11]. Fig. 1 shows a permutation graph and its corresponding permutation diagram.

The remaining part of this paper is organized as follows. In Section 2, we introduce an algorithm which can be parallelized to construct a spanning tree of a permutation graph. And the correctness of this algorithm will be validated in Section 3. Finally, the conclusion of this paper is presented in Section 4.

2. An algorithm for constructing a spanning tree

In this section we show an algorithm for constructing a spanning tree of a permutation graph. The algorithm can be parallelized by applying parallel prefix computation [10]. In the following, we use (u, v) to denote an edge incident to two distinct vertices u and v. Algorithm A which is used to construct a spanning tree is presented as follows.

Algorithm A

Input: A sequence $P = [p_1, p_2, \ldots, p_n]$ of a permutation graph G.
Output: A spanning tree T^* of G.

Method:

Step 1. Let T^* be a graph with n vertices $(1, 2, \ldots, n)$ and no edges.
Step 2. Scan the sequence P from p_n to p_1. Let l_i be the minimum element in $\{p_n, p_{n-1}, \ldots, p_i\}$, $i = n, n-1, \ldots, 1$.
Step 3. Scan the sequence P from p_1 to p_n. Let r_i be the maximum element in $\{p_1, p_2, \ldots, p_i\}$, $i = 1, 2, \ldots, n$.

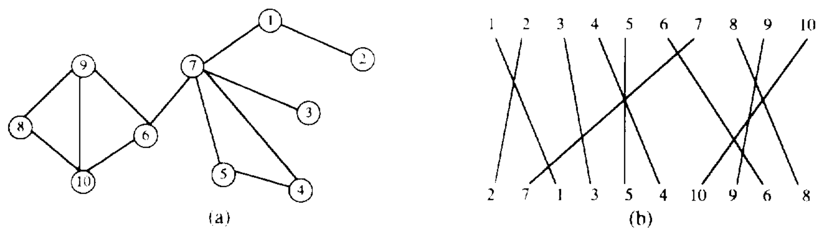

Fig. 1. (a) A permutation graph. (b) Its corresponding permutation diagram.

i	1	2	3	4	5	6	7	8	9	10
p_i	2	7	1	3	5	4	10	9	6	8
l_i	1	1	1	3	4	4	6	6	6	8
r	2	7	7	7	7	7	10	10	10	10

Fig. 2. Illustration of Step 4.

Step 4. For $i = 1$ to n, if $p_i \neq l_i$, then $T^* = T^* \cup (p_i, l_i)$.
Step 5. For $i = 1$ to $n-1$, if $l_i \neq l_{i+1}$, then $T^* = T^* \cup (r_i, l_{i-1})$.

We use the graph of Fig. 1 as an example to illustrate Algorithm A step by step.

Step 1. Initially, T^* contains n vertices and no edges.
Step 2. The sequence of l_i, $i = 1, 2, \ldots, n$, is [1,1,1,3,4,4,6,6,6,8].
Step 3. The sequence of r_i, $i = 1, 2, \ldots, n$, is [2,7,7,7,7,7,10,10,10,10].
Step 4. There are five edges, (2,1), (7,1), (5,4), (10,6), and (9,6), which are included into T^* (see Fig. 2).
Step 5. There are four edges, (7,3), (7,4), (7,6) and (10,8) of T^*, which are obtained in this step (see Fig. 3).

Finally, we obtain a spanning tree T^* which contains nine edges, (2,1), (7,1), (5,4), (10,6), (9,6), (7,3), (7,4), (7,6) and (10,8). We show the spanning tree pictorially in Fig. 4.

Since each step of Algorithm A takes O(n) time in sequential, the time-complexity of Algorithm A is O(n). However, we have known that the parallel prefix computation can be done in O(log n) time with O(n/log n) processors on the EREW PRAM for n-object lists [7,10], Steps 2 and 3 can be done in O(log n) time with O(n/log n) processors as well. And in parallel the other steps take O(log n) time with O(n/log n) processors each. Thus, the parallel time-complexity of Algorithm A is O(log n) time with O(n/log n) processors on the EREW PRAM. Besides, linear space is needed in this algorithm.

i	1	2	3	4	5	6	7	8	9	10
p_i	2	7	1	3	5	4	10	9	6	8
l_i	1	1	1	3	4	4	6	6	6	8
r_i	2	7	7	7	7	7	10	10	10	10

Fig. 3. Illustration of Step 5.

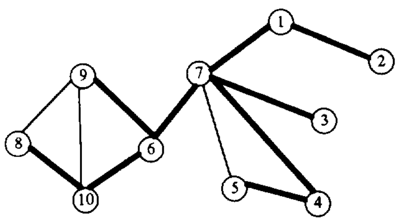

Fig. 4. The spanning tree obtained by Algorithm A of the graph in Fig. 1(a).

3. The correctness of algorithm A

In this section, we prove the correctness of Algorithm A. In the following lemmas, we assume that the permutation graph $G = (V, E)$ has more than one vertex and is connected.

Lemmas 3.1 and 3.2 prove that the edges obtained at Steps 4 and 5 are the edges of a permutation graph G.

Lemma 3.1. *If* $p_i \neq l_i$, $i = 1, 2, \ldots, n$, *then* (p_i, l_i) *is an edge of* G.

Proof. Since $p_i \neq l_i$ and $l_i = \min(l_{i+1}, p_i)$, $p_i > l_i = l_{i+1}$. Furthermore, since $l_i = l_{i+1}$, we obtain $p_{pi}^{-1} < p_{li}^{-1}$, where $p_{pi}^{-1} = i$ and $p_{li}^{-1} \geqslant i + 1$. Thus, $(p_i - l_i)(p_{pi}^{-1} - p_{li}^{-1}) < 0$. By the definition of a permutation graph, (p_i, l_i) must be an edge of G. □

Lemma 3.2. *Every* (r_i, l_{i+1}), $1 \leqslant i < n$, *is an edge of* G.

Proof. For proving (r_i, l_{i+1}) is an edge, we have to show $r_i > l_{i+1}$. By Algorithm A, r_i is the maximum element in $\{p_1, p_2, \ldots, p_i\}$ and l_{i+1} is the minimum element in $\{p_{i+1}, p_{i+2}, \ldots, p_n\}$. We shall prove the following two cases are impossible.

Case 1: $r_i < l_{i+1}$. This means there exists no element in $\{p_1, p_2, \ldots, p_i\}$ greater than any element in $\{p_{i+1}, p_{i+2}, \ldots, p_n\}$. Since $(p_x - p_y)(x - y) > 0$ for $1 \leqslant x \leqslant i$ and $i + 1 \leqslant y \leqslant n$, there is no edge incident to both p_x and p_y, where $p_x \in \{p_1, p_2, \ldots, p_i\}$ and $p_y \in \{p_{i+1}, p_{i+2}, \ldots, p_n\}$. Thus, G is not connected and this case contradicts our assumption.

Case 2: $r_i = l_{i+1}$. Since $r_i = \max\{p_1, p_2, \ldots, p_i\}$, $l_{i+1} = \min\{p_{i+1}, p_{i+2}, \ldots, p_n\}$ and $p_i \neq p_j$ if $i \neq j$, this condition cannot hold.

Therefore, $r_i > l_{i+1}$ and $(r_i - l_{i+1})(p^{-1}_{r_i} - p^{-1}_{l_{i+1}}) < 0$, where $p^{-1}_{r_i} \leqslant i$ and $p^{-1}_{l_{i+1}} \geqslant i+1$. We conclude that if G is a connected permutation graph, every (r_i, l_{i+1}), $1 \leqslant i \leqslant n$, is an edge of G. □

Before we prove that the tree T^* found by Algorithm A is a spanning tree, we need the following definitions. Two different vertices p_i and p_j belong to the same *subtree component* if $l_i = l_j$. For p_i, if there exists no other vertex p_j which has $l_i = l_j$, then p_i is a *single vertex subtree component*. By the definition of l_i, every subtree component contains consecutive p_i's. Using Fig. 1 as an example, Fig. 5 illustrates our definitions. {2,7,1}, {3}, {5,4}, {10,9,6} and {8} are subtree components while {3} and {8} are single vertex subtree components.

Lemma 3.3. *Let* $S = \{p_i, p_{i+1}, \ldots, p_j\}$ *be a subtree component. Then* $T = (S, E)$ *forms a subtree of G, where* $E = \{(p_x, l_j) \mid i \leqslant x < j\}$.

Proof. Since S is a subtree component, $l_i = l_{i+1} = \cdots = l_j$. By the definition of l_i, we know $p_j = l_j$ and $p_x > l_j$ for $i \leqslant x < j$. By Lemma 3.1, every (p_x, l_j) is an edge of G. This means that $T = (S, E)$ forms a subtree of G. □

i	1	2	3	4	5	6	7	8	9	10
p_i	2	7	1	3	5	4	10	9	6	8
l_i	1	1	1	3	4	4	6	6	6	8
r_i	2	7	7	7	7	7	10	10	10	10

Fig. 5. Subtree components.

Theorem 3.4 *Algorithm* A *finds a spanning tree of a permutation graph.*

Proof. Suppose $S_1, S_2, \ldots, S_k$ are all of the subtree components of G and have $n_1, n_2, \ldots, n_k$, respectively, vertices. $T_1, T_2, \ldots, T_k$ are their corresponding subtrees as defined in Lemma 3.3. First, we have to prove that $n_1 + n_2 + \cdots + n_k = n$, where n is the number of vertices in G. Since every p_i only has a unique l_i, every p_i can only belong to one exact subtree component. It implies that $n_1 + n_2 + \cdots + n_k = n$. Second, we have to show that those k subtrees $T_1, T_2, \ldots, T_k$ can be combined to form a tree. Let $S_x = \{p_i, p_{i+1}, \ldots, p_j\}$, $1 < i < j \leqslant n$, be a subtree component. Since $r_{i-1} \in \{p_1, p_2, \ldots, p_{i-1}\}$ (the maximum element in $\{p_1, p_2, \ldots, p_{i-1}\}$), $r_{i-1} \notin \{p_i, p_{i+1}, \ldots, p_j\}$. This implies that r_{i-1} is not a node of subtree T_x. However, l_i $(= p_j)$ is a vertex in T_x. It means that edge (r_{i-1}, l_i) is an edge that combines T_x with another subtree. By Lemma 3.2, we know (r_j, l_{j+1}) is an edge of G. Also by the property of trees, the combination of two trees by one edge still forms a tree. Step 5 of Algorithm A combines all of subtrees to form a tree T^*. Thus, the number of edges of T^* is

$$(n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) + k - 1 = n - 1.$$

By the definition of a tree, T^* is a spanning tree of G. □

4. Conclusion

Algorithm A can construct a spanning tree of a permutation graph. With the similar argument, we can construct another spanning tree by modifying Steps 4 and 5 of Algorithm A as follows.

Step 4. For $i = 1$ to n, if $p_i \neq r_i$, then $T^* = T^* \cup (p_i, r_i)$.
Step 5. For $i = 1$ to $n - 1$, if $r_i \neq r_{i+1}$, then $T^* = T^* \cup (r_i, l_{i+1})$.

In this paper we present a parallel algorithm to construct a spanning tree on an unweighted connected permutation graph with n vertices.

This problem can be solved in O(log n) time by the above parallel algorithm with O(n/log n) processors and linear space, and our approach used the parallel prefix computation on the EREW PRAM.

References

[1] P.M. Camerini, G. Galbiati and F. Maffioli, Algorithms for finding optimum trees: Description, use, and evaluation, *Ann. Oper. Res.* **13** (1988) 265–397.

[2] S. Even, A. Pnnueli and A. Lempel, Permutation graphs and transitive graphs, *J. ACM* **19** (1972) 400–410.

[3] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in network optimization algorithms, in: *Proc. 25th Ann. IEEE Symp. on FOCS* (1984) 338–346.

[4] H.P. Gabow, Z. Galil, T. Spencer and R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, *Combinatorica* **6** (1986) 109–122.

[5] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs* (Academic Press, New York, 1980).

[6] R. Gould, *Graph Theory* (Benjamin/Cummings, Menlo Park, CA, 1988) 65–67.

[7] J. JáJá, *An Introduction to Parallel Algorithms* (Addison-Wesley, Reading, MA, 1992) 44–47.

[8] P. Klein and C. Stein, A parallel algorithm for eliminating cycles in undirected graphs, *Inform. Process. Lett.* **34** (1990) 307–312.

[9] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Amer. Math. Soc.* **7** (1956) 48–50.

[10] C.P. Kruskal, L. Rudolph and M. Snir, The power of parallel prefix, *IEEE Trans. Comput.* **34** (1985) 965–968.

[11] A. Pnueli, A. Lempel and S. Even, Transitive orientation of graphs and identification of permutation graphs, *Canad. J. Math.* **23** (1971) 160–175.

[12] R.C. Prim, Shortest connection networks and some generalizations, *Bell Systems Tech. J.* **36** (1957) 1389–1401.

[13] F. Suraweera and P. Bhattacharya, An O(log m) parallel algorithm for the minimum spanning tree problem, *Inform. Process. Lett.* **45** (1993) 159–163.

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

[15] A. Yao, An O($|E|$log log$|V|$) algorithm for finding minimum spanning trees, *Inform. Process. Lett.* **4** (1975) 21–23.