ELSEVIER

Information Processing Letters 63 (1997) 13–18

Information Processing Letters

A linear time algorithm for finding depth-first spanning trees on trapezoid graphs

Hon-Chan Chen, Yue-Li Wang *

Department of Information Management, National Taiwan Institute of Technology, Taipei, Taiwan, ROC

Received 14 October 1996; revised 18 April 1997
Communicated by T. Asano

Abstract

Let G be a connected graph of n vertices and m edges. The problem of finding a depth-first spanning tree of G is to find a subgraph of G connecting the n vertices with $n-1$ edges by depth-first search. In this paper, we propose an $O(n)$ time algorithm for solving this problem on trapezoid graphs. Our algorithm can also find depth-first spanning trees of permutation graphs in linear time, improving the recent algorithm on permutation graphs which takes $O(n \log \log n)$ time. © 1997 Elsevier Science B.V.

Keywords: Depth-first search; Spanning tree; Trapezoid graphs; Permutation graphs

1. Introduction

Let $G=(V, E)$ be a connected graph with vertex set V and edge set E, where $|V|=n$ and $|E|=m$. A *spanning tree* of G is a spanning subgraph of G which is a tree and connects the n vertices. Typically, there are many different spanning trees in a graph. A *depth-first spanning tree* is a spanning tree which is found by *depth-first search* (DFS) [16]. In DFS, we select and visit a vertex a, then visit a vertex b adjacent to a, continuing with a vertex of c adjacent to b (but different from a), followed by an unvisited d adjacent to c, and so forth. As we go deeper and deeper into the graph, we will eventually visit a vertex y with no unvisited neighbors; when this happens, we return to the vertex x immediately preceding y in the search and revisit x. When all vertices are visited, we stop the search. The edge (x, y) is placed into the depth-first spanning tree if vertex y was visited for the first time immediately following a visit to x. In this case, x is called the *parent* of y and y is a *child* of x.

In this paper, we will find depth-first spanning trees on trapezoid graphs. A trapezoid i is defined by four corner points $[a_i, b_i, c_i, d_i]$ such that a_i and b_i are on the top channel and c_i and d_i are on the bottom channel of the trapezoid diagram. A graph $G=(V, E)$ is a *trapezoid graph* if it can be represented by a trapezoid diagram such that each trapezoid corresponds to a vertex in V and $(i, j) \in E$ if and only if trapezoids i and j intersect in the trapezoid diagram [4]. Fig. 1 presents a trapezoid graph with its trapezoid diagram. In the diagram,

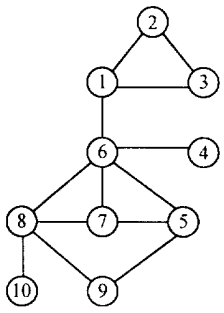
* Corresponding author. Mailing address: Department of Information Management, National Taiwan Institute of Technology, 43, Section 4, Kee-Lung Road, Taipei, Taiwan 106, Republic of China, Email: ylwang@cs.ntit.edu.tw.

PII S0020-0190(97)00088-4

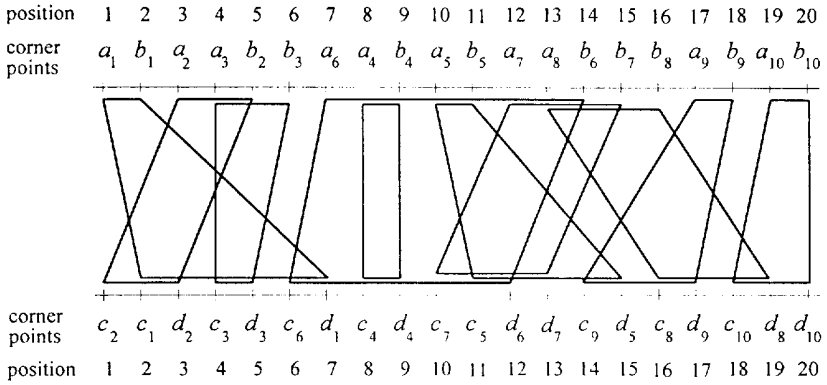
there are 10 trapezoids, and the four corner points of trapezoid i are a_i, b_i, c_i and d_i, $i = 1, 2, \ldots, 10$. The class of trapezoid graphs includes two well-known classes of intersection graphs: the permutation graphs and the interval graphs [5]. The permutation graphs are obtained in the case where $a_i = b_i$ and $c_i = d_i$ for all i, and the interval graphs are obtained in the case where $a_i = c_i$ and $b_i = d_i$ for all i.

Trapezoid graphs can be recognized in $O(n^2)$ time by Ma and Spinrad's algorithm [11]. Applying their algorithm, trapezoid diagrams can also be constructed. It is easy to show that a trapezoid diagram can be reconstructed into another one corresponding to the same trapezoid graph such that each trapezoid has four distinct corner points and no two trapezoids share common corner points. Therefore, for simplicity, we assume that the corner points on our trapezoid diagram are all distinct, and they are given consecutive positions $1, 2, \ldots, 2n$ from left to right on both channels. We also assume that trapezoids are labelled in increasing order of their b corner points for ease of description. That is, for two trapezoids i and j, $i < j$ if b_i lies to the left of b_j. For example, in Fig. 1(b), trapezoid 6 is before trapezoid 7 since b_6 is at position 14 and b_7 is at position 15 on the top channel.

There are a wide variety of papers discussing the depth-first spanning tree problem [1,2,6,7,10,13–15]. In [16], Tarjan described the technique of DFS in detail. The time complexity of DFS in general graphs is $O(n + m)$, where n is the number of vertices and m the number of edges. Trapezoid graphs were first studied in [3,4]. Dagan et al., in [4], introduced a coloring algorithm for trapezoid graphs. In [11], Ma and Spinrad presented an $O(n^2)$ time algorithm for



(a)



(b)

Fig. 1. (a) A trapezoid graph, (b) The corresponding trapezoid diagram.

recognizing this class of graphs. Recently, Liang gave some sequential algorithms for dominating and breadth-first spanning tree problems on trapezoid graphs [8,9].

In this paper, we assume trapezoid diagrams are given. We propose a linear time algorithm for finding depth-first spanning trees on connected trapezoid graphs. Since the class of permutation graphs is a subclass of trapezoid graphs, depth-first spanning trees on permutation graphs can be found in $O(n)$ time by our algorithm, improving the recent algorithm on permutation graphs which takes $O(n \log \log n)$ time [12]. The remaining part of this paper is organized as follows. In Section 2, we introduce our algorithm of finding a depth-first spanning tree. The correctness of our algorithm is shown in Section 3. Finally, in Section 4, we give the conclusion of this paper.

2. An algorithm for finding a depth-first spanning tree

Before describing our algorithm, we introduce some notations which will be used later. Let G be a trapezoid graph of n vertices. We will proceed our algorithm on the corresponding trapezoid diagram of G. Denote by $pos(\cdot)$ the position of some corner point. For example, in Fig. 1(b), $pos(b_3) = 6$ on the top channel and $pos(d_3) = 5$ on the bottom channel. On the contrary, a position corresponds to a corner point as well as to a trapezoid. We denote by $V_t(\cdot)$ (respectively, $V_b(\cdot)$) the corresponding trapezoid of some position on the top (respectively, the bottom) channel. For instance, $V_t(7)$ is trapezoid 6 and $V_b(7)$ is trapezoid 1 since the corner point at position 7 on the top channel is a_6 while d_1 is at position 7 on the bottom channel. Index *top* (respectively, *bottom*) indicates the latest scanned corner point on the top (respectively, bottom) channel. When a vertex v of G is visited, set $flag(v)$ = TRUE; otherwise, $flag(v)$ = FALSE.

Our algorithm of finding a depth-first spanning tree is presented as follows. In the algorithm, *parent*(v) stands for the parent of v in the depth-first spanning tree, and (i, j) stands for the edge incident to i and j.

Algorithm A
Input: A trapezoid diagram with n trapezoids
Output: A depth-first spanning tree T starting from vertex 1.
Method:
Step 1. {Initialize all conditions.}
  $T := \emptyset$;
  for $i := 1$ to n **do**
    $flag(i) :=$ FALSE;
  $parent(1) := 0$;
  $i := 1$;
  $top := 1$;
  $bottom := 1$;
  $flag(i) :=$ TRUE;
Step 2. {Scan corner points on the top channel to find an unvisited neighbor.}
  while $flag(V_t(top)) =$ TRUE **and**
      $top < pos(b_i)$ **do**
    $top := top + 1$;
Step 3. {Scan corner points on the bottom channel to find an unvisited neighbor.}
  while $flag(V_b(bottom)) =$ TRUE **and**
      $bottom < pos(d_i)$ **do**
    $bottom := bottom + 1$;
Step 4. {If vertex i has no unvisited neighbors, go back to its parent.}
  if $top \geqslant pos(b_i)$ **and** $bottom \geqslant pos(d_i)$ **then**
    begin
      while $top \geqslant pos(b_i)$ **and** $bottom \geqslant pos(d_i)$
          and $i \neq 0$ **do**
        $i := parent(i)$;
      if $i \neq 0$ **then**
        goto Step 2;
    end;
Step 5. {Find the next vertex to visit or stop the algorithm.}
  if $i \neq 0$ **then**
    begin
      if $top < pos(b_i)$ **and** $bottom < pos(d_i)$ **then**
        $u := \min\{V_t(top), V_b(bottom)\}$
      else if $top < pos(b_i)$ **and**
          $bottom \geqslant pos(d_i)$ **then**
        $u := V_t(top)$
      else if $top \geqslant pos(b_i)$ **and**
          $bottom < pos(d_i)$ **then**
        $u := V_b(bottom)$;
      $parent(u) := i$;
      $T := T \cup (i, u)$

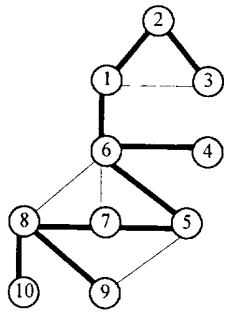


Fig. 2. The resulting depth-first spanning tree starting from vertex 1.

```
        i := u;
        flag(i) := TRUE;
        goto Step 2;
      end
    else
      output T;
End of Algorithm A
```

We use the graph of Fig. 1(b) as an example to illustrate Algorithm A. After the initialization in Step 1, we consider trapezoid 1. Scanning the top channel and the bottom channel, we find trapezoid 2 intersects trapezoid 1. Let vertex 1 be the parent of vertex 2, and let (1, 2) be an edge of T. Now consider vertex 2. Continuing the scanning on both channels, we find trapezoid 3 intersects trapezoid 2. Let vertex 2 be the parent of vertex 3, and insert edge (2, 3) into T. Continuing the scanning, we find that no unvisited trapezoids intersect trapezoid 3. At the moment, go back to the parent of vertex 3; i.e. vertex 2. Since all corner points before $pos(b_2)$ and $pos(d_2)$ were scanned, no unvisited trapezoids intersect trapezoid 2. Therefore, we still go back to the parent of vertex 2; i.e. vertex 1. Continuing the scanning, we find trapezoid 6 intersects trapezoid 1. Thus, let vertex 1 be the parent of vertex 6, and let (1, 6) be an edge of T. After Algorithm A terminates, T is a depth-first spanning tree of G as shown in Fig. 2.

3. The correctness of Algorithm A

In this section, we will prove the correctness of Algorithm A. Let G be a connected trapezoid graph of n vertices. When we visit a vertex i of G in the execution of our algorithm (no matter i is first visited or not), i is, at the moment, called the *currently visited vertex*. The vertex which is visited immediately after visiting the currently visited vertex is called the *next visited vertex*. Remember that *top* (respectively, *bottom*) always indicates the latest scanned corner point on the top (respectively, bottom) channel. For completing the correctness, we will show that

(i) the graph T constructed by Algorithm A is in depth-first search,
(ii) T is a tree connecting n vertices, and
(iii) Algorithm A takes $O(n)$ time.

The following property of trapezoid graphs is useful for our proofs.

Property. *Let i and j, $i < j$, be two vertices of G. Then, i is adjacent to j if and only if $pos(b_i) > pos(a_j)$ or $pos(d_i) > pos(c_j)$ in the trapezoid diagram.*

Lemma 1. *Let i, $1 \leqslant i \leqslant n$, be the currently visited vertex in the execution of Algorithm A. If $top < pos(b_i)$ in Step 5, then $V_t(top)$ is an unvisited neighbor of i in G. Similarly, if $bottom < pos(d_i)$ in Step 5, then $V_b(bottom)$ is an unvisited neighbor of i in G.*

Proof. We shall only prove the case where $top < pos(b_i)$. The other case, $bottom < pos(d_i)$, can be proved similarly. Since $top < pos(b_i)$ in Step 5, $flag(V_t(top))$ must be FALSE in Step 2. If $V_t(top) > i$, then $V_t(top)$ is an unvisited neighbor of i since $top = pos(a_{V_t(top)}) < pos(b_i) < pos(b_{V_t(top)})$. If $V_t(top) < i$, then $V_t(top)$ is also an unvisited neighbor of i. If it is not a neighbor, then $pos(b_{V_t(top)}) < pos(a_i)$ and $pos(d_{V_t(top)}) < pos(c_i)$. But i is visited. This implies that $V_t(top)$ is also visited since $b_{V_t(top)}$ or $d_{V_t(top)}$ have been scanned. A contradiction. Thus, if $top < pos(b_i)$, then $V_t(top)$ is an unvisited neighbor of i in G. □

If $top < pos(b_i)$ and $bottom < pos(d_i)$ for currently visited vertex i, $1 \leqslant i \leqslant n$, in Step 5 of Algorithm A, then both $V_t(top)$ and $V_b(bottom)$ are unvisited neighbors of i in G. Either $V_t(top)$ or $V_b(bottom)$ can be the next visited vertex. In this case, we

select $u = \min\{V_t(top), V_b(bottom)\}$ as the next visited vertex for simplicity.

Corollary 2. *If $top \geqslant pos(b_i)$ and $bottom \geqslant pos(d_i)$ for currently visited vertex i in Step 4 of Algorithm A, $1 \leqslant i \leqslant n$, then i has no unvisited neighbors in G, and the next visited vertex is $parent(i)$.*

Lemma 3. *The graph T constructed by Algorithm A is in depth-first search.*

Proof. The main idea of DFS is that if currently visited vertex i has unvisited neighbors in G, then one of the unvisited neighbors will be the next visited vertex. Otherwise, if all the neighbors of i were visited, then $parent(i)$ is the next visited vertex. In the execution of Algorithm A, we have to consider four cases:
Case 1. $top < pos(b_i)$ and $bottom < pos(d_i)$;
Case 2. $top < pos(b_i)$ and $bottom \geqslant pos(d_i)$;
Case 3. $top \geqslant pos(b_i)$ and $bottom < pos(d_i)$;
Case 4. $top \geqslant pos(b_i)$ and $bottom \geqslant pos(d_i)$.
If it is in one of the first three cases, by Step 5 of Algorithm A and Lemma 1, either $V_t(top)$ or $V_b(bottom)$ is the next visited vertex which is an unvisited neighbor of i in G. Then, we insert edge (i, u) into T, where u is the next visited vertex. If it is in Case 4, by Corollary 2, $parent(i)$ is the next visited vertex and no new edge is added on T. Thus, the graph T constructed by Algorithm A is in depth-first search. □

Lemma 4. *The graph T constructed by Algorithm A is a tree connecting n vertices.*

Proof. In Step 5, we always insert an edge (i, u) into T only when we find an unvisited neighbor u of currently visited vertex i. It is impossible to insert an edge incident with two visited vertices to form a cycle. Thus, T is a tree. Since G is connected and *top* and *bottom* scan all corner points on both channels, all of the n vertices of G can be visited in Algorithm A. Therefore, T is a tree containing n vertices. □

Obviously, each edge of T was visited at most twice. This is because we go through $(parent(i), i)$ if i was first visited and go through $(i, parent(i))$ if i has no unvisited neighbors in G.

Theorem 5. *Algorithm A finds depth-first spanning trees on trapezoid graphs in $O(n)$ time.*

Proof. Lemmas 3 and 4 have shown that the graph T constructed by Algorithm A is a tree connecting n vertices in depth-first search. Since edge (i, u) is added into T only when we find an unvisited neighbor u of currently visited vertex i, (i, u) is an edge of G. This completes that T is a depth-first spanning tree of G. We show the complexity of Algorithm A as follows. Since *top* and *bottom*, respectively, scan the top and the bottom channels once on the trapezoid diagram, Steps 2 and 3 totally take $O(n)$ time. When we visit a vertex i which has no unvisited neighbors, we go back to $parent(i)$ on T to continue our algorithm. Since T has at most $n - 1$ edges and each edge was visited at most twice, Step 4 totally takes $O(n)$ time. Step 5 can totally be done in $O(n)$ time and Step 1 is also in $O(n)$ time. Therefore, Algorithm A takes $O(n)$ time. □

4. Conclusion

In this paper, we present an $O(n)$ time algorithm for finding depth-first spanning trees on trapezoid graphs. Since the class of permutation graphs is a subclass of trapezoid graphs, depth-first spanning trees on permutation graphs can also be found in $O(n)$ time by our algorithm. This improves the recent algorithm of finding depth-first spanning trees on permutation graphs which takes $O(n \log \log n)$ time.

References

[1] A. Aggarwal, R.J. Anderson and M.Y. Kao, Parallel depth-first search in general directed graphs, *SIAM J. Comput.* 19 (1990) 397–409.
[2] P. Chaudhuri, Finding and updating depth-first spanning trees of acyclic digraphs in parallel, *Comput. J.* 33 (1990) 247–251.
[3] D.G. Corneil and P.A. Kamula, Extensions of permutation and interval graphs, *Congr. Numer.* 58 (1987) 267–275.
[4] I. Dagan, M.C. Golumbic and R.Y. Pinter, Trapezoid graphs and their coloring, *Discrete Appl. Math.* 21 (1988) 35–46.

[5] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs* (Academic Press, New York, 1980).
[6] T. Hagerup, Planar depth-first search in O(log n) parallel time, *SIAM J. Comput.* 19 (1990) 678–704.
[7] E. Korach and Z. Ostfeld, On the existence of special depth first search trees, *J. Graph Theory* 19 (1995) 535–547.
[8] Y.D. Liang, Dominations in trapezoid graphs, *Inform. Process. Lett.* 52 (1994) 309–315.
[9] Y.D. Liang, Steiner set and connected domination in trapezoid graphs, *Inform. Process. Lett.* 56 (1995) 101–108.
[10] Y. Liang, C. Rhee, S.K. Dhall and S. Lakshmivarahan, NC algorithms for finding depth-first-search trees in interval graphs and circular-arc graphs, in: *IEEE Proc. SOUTHEASTCON '91*, Vol. 1, pp. 582–585.
[11] T.H. Ma and J.P. Spinrad, On the 2-chain subgraph cover and related problems, *J. Algorithms* 17 (1994) 251–268.
[12] C. Rhee, Y.D. Liang, S.K. Dhall and S. Lakshmivarahan, Efficient algorithms for finding depth-first and breadth-first search trees in permutation graphs, *Inform. Process. Lett.* 49 (1994) 45–50.
[13] H. Salehi-Fathabadi and H. Ahrabian, A new algorithm for minimum spanning tree using depth-first-search in an undirected graph, *Internat. J. Comput. Math.* 57 (1995) 157–161.
[14] G.E. Shannon, A linear-processor algorithm for depth-first search in planar graphs, *Inform. Process. Lett.* 29 (1988) 119–123.
[15] M.B. Sharma, S.S. Iyengar and N.K. Mandyam, An efficient distributed depth-first-search algorithm, *Inform. Process. Lett.* 32 (1989) 183–186.
[16] R.E. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comput.* 1 (1972) 146–160.