Discrete Applied Mathematics 41 (1993) 75-79
North-Holland

Note

Fast algorithms for finding Hamiltonian paths and cycles in in-tournament digraphs*

Jørgen Bang-Jensen

Department of Mathematics and Computer Science, Odense University, Odense, DK-5230, Denmark

Pavol Hell

School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada

Received 4 September 1991
Revised 15 April 1992

Abstract

Bang-Jensen, J. and P. Hell, Fast algorithms for finding Hamiltonian paths and cycles in in-tournament digraphs, Discrete Applied Mathematics 41 (1993) 75–79.

An *in-tournament digraph* is a digraph in which the set of in-neighbours of every vertex induces a tournament. For in-tournament digraphs we give $O(m + n \log n)$ algorithms to find a Hamiltonian path and a Hamiltonian cycle if they exist. Here n, m denote the number of vertices respectively arcs of the in-tournament digraph.

A *local tournament digraph* (*local tournament* for short) [2, 5], is an oriented graph in which the set of in-neighbours as well as the set of out-neighbours of each vertex induce a tournament. A graph is orientable as a local tournament if and only if it is a proper circular arc graph, [9], cf. also [5]. Using this fact, we developed an $O(\Delta m)$ algorithm to recognize, [5], and represent, [3], proper circular arc graphs

Correspondence to: Professor J. Bang-Jensen, Department of Mathematics and Computer Science, Odense University, Campusvej 55, Odense, DK-5230, Denmark.

* Partially supported by grants from NSERC and ASI. This note was written while the authors were visiting the Laboratoire de Recherche en Informatique, Université de Paris-Sud, whose hospitality and financial support is gratefully acknowledged.

(Δ denotes the maximum degree). Here we develop efficient algorithms for Hamiltonian problems (finding a Hamiltonian path and a Hamiltonian cycle if they exist, a longest path and a longest cycle otherwise) for a somewhat larger class of oriented graphs. An *in-tournament digraph* (we shall use the term *in-tournament* for short) is an oriented graph in which the set of in-neighbours of every vertex induces a tournament. In other words, the presence of arcs xv and yv implies that x and y are adjacent, i.e., that either xy or yx is an arc. In-tournaments include of course all tournaments, and all local tournaments, but also all out-branchings, and other classes of oriented graphs. An *out-branching* is a spanning tree in which each vertex except one (called the root) has exactly one incoming arc. An *in-branching* is defined analogously. The problem of characterizing graphs orientable as in-tournaments has been proposed in [9], and some partial answers are given in [4]. (In particular, it is shown in [4] that they can be recognized in polynomial time.)

Here we present two algorithms for constructing a Hamiltonian (or, if not possible, then a longest) path and cycle in an in-tournament. These results were presented in a preliminary form (and only for local tournaments) in [3,5]. For our purposes we only need to deal with connected digraphs, whence we assume that all digraphs mentioned in this paper are connected (i.e., their underlying graphs are connected). Both algorithms are based on the following lemma, showing an efficient way to insert one new vertex into an already constructed path. Whenever we say that a vertex x *dominates* a vertex y this means that xy is an arc.

Lemma 1. *Let D be an in-tournament and let P be a directed path of length r in D. Let x be a vertex of $D-P$ which dominates a vertex v of P. Then with* $O(\log r)$ *steps we can find in D a directed path P' of length $r+1$ containing x and all the vertices of P.*

Proof. Suppose $P=p_0,p_1,\dots,p_r$ and $v=p_i$. Since D is an in-tournament, x either dominates or is dominated by p_{i-1}. By the same argument we conclude that there is a j, $1\le j\le i-1$, such that p_j dominates x and x dominates p_{j+1}, or that x dominates p_0. In either case, it is clear how to define P' by "inserting" x between p_j and p_{j+1}, or before p_0, in P.

To accomplish this insertion efficiently, we shall store the growing path P in a balanced binary tree, B, cf. [7, 6.2.3]. The order of the vertices on P is represented by the in-order of B. For each vertex p of P, we keep the father, $f(p)$, as well as the left and right children, $l(p)$ and $r(p)$, in B. It is well known, [7], that the height of B is $O(\log r)$ and that once we know the place to insert x (i.e., an arc pq of P such that p dominates x and x dominates q, or the arc from x to the initial vertex of P), B can be updated with $O(\log r)$ operations. Thus it remains to explain how to find the insertion point. Let us keep track of an "active" segment of P, as a portion of P which is guaranteed to allow the insertion of x. Initially, we take as active the segment from the starting vertex of P up to v. (We know from the above that x can be inserted somewhere in this segment.) We shall be able to continue by binary search, provided we can find the midpoint of each current active segment. For this purpose we devote another $O(\log r)$ operation to obtain the two sequences $s_0 = v$, $s_1, \ldots, s_m$ and $t_1, t_2, \ldots, t_m$ such that $s_i = f(s_{i-1})$ and $t_i = l$ or r depending on whether $s_{i-1} = l(s_i)$ or $s_{i-1} = r(s_i)$. Because the path order on P is represented by the in-order traversal of B, $t_i = r$ means that s_i precedes all s_j with $j < i$, including $s_0 = v$. Let k be the greatest subscript such that $t_k = r$ and x dominates s_k. (If such a subscript doesn't exist, we let $k = 0$.) If all subscripts h greater than k have $t_h = l$, then we let the new active segment be the portion of P from the starting point up to s_k. Otherwise, let h be the smallest subscript greater than k with $t_h = r$. Since x does not dominate s_h, we know that x can be inserted between s_h and s_k. Then the new active segment is the segment from s_h to s_k. In either case the new active segment of P is represented precisely by the subtree of B rooted at $l(s_k)$, and now binary search is easy to perform in the standard style (because each active segment is a subtree and its midpoint corresponds to the root of the subtree). □

Theorem 2. *Suppose F is an in-branching of D represented by lists of in-neighbours ($I(x)$ is the list of in-neighbours of x in F). Then a directed Hamiltonian path in D can be found with* $O(n \log n)$ *operations.*

Proof. Starting from the root, we grow a path, always keeping a stack S of candidates for the next insertion (application of Lemma 1). Initially S contains all the in-neighbours of the root, and in general taking for insertion the vertex x from the top of S, we place in S all of $I(x)$. □

It follows from Theorem 2 that an in-tournament has a Hamiltonian path if and only if it has an in-branching, cf. also [4]. A digraph has an in-branching if and only if it does not have two *terminal* strong components, i.e., components with no out-neighbours. The final algorithm for finding a Hamiltonian path in D (if one exists) consists of finding the strong components of D, and, provided there is only one terminal strong component, finding an in-branching F of D by breadth first search from any vertex of that component, and finally by applying the above theorem.

In these algorithms we assume the digraph D is represented by the lists of in- and out-neighbours. This allows us to get all in- and out-neighbours of a vertex v with $O(\deg v)$ operations. We also need additional information suitable to decide, in time $O(1)$, whether, for given vertices u and v, u dominates v, v dominates u, or neither. It is possible to obtain, in time $O(m)$, a version of the adjacency matrix of D (with valid entries certified by the means of an additional stack), which allows us to do this, cf. [1, Exercise 2.12].

Thus the above algorithm has time complexity $O(m + n \log n)$. Here m is the number of arcs of D. Thus $O(m)$ is the time for finding the strong components, and for the breadth first search.

If the in-tournament D has more than one terminal component, we shall describe a method (of the same complexity) to find a longest directed path in D. It should be clear that this path is the Hamiltonian path of some maximal subgraph of D

which admits an in-branching. These maximal subgraphs correspond to the in-branchings starting in one of the terminal strong components. The "right" terminal component can be found in time $O(m)$: Since D is an in-tournament, it has only one initial strong component (component with no in-neighbours). We label this initial component by its number of vertices. It is now easy to see how to label, in time $O(m)$, each strong component of D by the number of vertices on all strong components preceding it. The component with a highest label is the right one to start the in-branching.

We summarize this as a corollary.

Corollary 3. *There is an $O(m+n\log n)$ algorithm to find a longest directed path in an in-tournament.*

Theorem 4. *There is an $O(m+n\log n)$ algorithm for finding a directed Hamiltonian cycle in a strong in-tournament D.*

Proof. One can use the above Hamiltonian path algorithm to do this. We offer the following algorithm, which is a generalization of an algorithm for tournaments by Manoussakis [8].

Using no more than $O(m+n\log n)$ steps we find a directed Hamiltonian path $x_1,x_2,\dots,x_n$. Next we find x_i such that x_i is dominated by x_n and i is as small as possible. If $i=1$ then $x_1,x_2,\dots,x_n,x_1$ is the desired cycle. Otherwise let $C=x_i,x_{i+1},\dots,x_n,x_i$. Since $x_{i-1}x_i$ and x_nx_i are arcs, either x_{i-1} dominates x_n, or x_n dominates x_{i-1}. The latter case contradicts the choice of i, hence x_{i-1} dominates x_n. Now consider arcs $x_{i-1}x_n$ and $x_{n-1}x_n$. We find that either x_{i-1} can be inserted in C to give a longer directed cycle or C is completely dominated by x_{i-1}. Suppose C is completely dominated by x_{i-1}, and that x_j, $j<i$, is dominated by a vertex x_t of C. Then we obtain a longer cycle $x_j,\dots,x_{i-1},x_{t+1},\dots,x_n,x_i,\dots,x_t,x_j$. Continuing this way we always have a cycle $C'=x'_q,x'_{q+1},\dots,x'_n,x'_q$ and a Hamiltonian path $x'_1,x'_2,\dots,x'_n$. Since we have a strong digraph, it cannot happen that none of the vertices $x'_1,x'_2,\dots,x'_{q-1}$ is dominated by a vertex of C. Thus this process terminates with a Hamiltonian cycle C'. Since we only consider each arc once in this part of the algorithm, we will have used no more than $O(m)$ new steps. $\square$

Corollary 5. *There exists an $O(m+n\log n)$ algorithm for finding a longest directed cycle in a local tournament.*

Proof. One can find all strong components, find a Hamiltonian cycle in each, and choose the longest of them, all in time $O(m+n\log n)$. $\square$

Since sorting corresponds to finding a Hamiltonian path in a transitive tournament, we have an algorithm doing $O(n\log n)$ tests in this case. In [6] we extended this to $O(n\log n)$ tests for finding Hamiltonian paths in arbitrary tournaments. The algorithms above may be viewed as further extension to the class of in-tournaments.

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] J. Bang-Jensen, Locally semicomplete digraphs: a generalization of tournaments, J. Graph Theory 14 (1990) 371-390.

[3] J. Bang-Jensen, P. Hell and J. Huang, Local tournaments and proper circular arc graphs, Tech. Rept. CSS/LCCR TR90-11, Simon Fraser University, Burnaby, B.C. (1990).

[4] J. Bang-Jensen, J. Huang and E. Prisner, In-tournaments, J. Combin. Theory Ser. B, to appear.

[5] P. Hell, J. Bang-Jensen and J. Huang, Local tournaments and proper circular arc graphs, Lecture Notes in Computer Science 450 (Springer, Berlin, 1990) 101-108.

[6] P. Hell and M. Rosenfeld, The complexity of finding generalized paths in tournaments, J. Algorithms 4 (1983) 303-309.

[7] D. Knuth, Art of Computer Programming, Vol. 3: Sorting and Searching (Addison-Wesley, Reading, MA, 1973).

[8] Y. Manoussakis, A linear algorithm for finding hamiltonian cycles in tournaments, Discrete Appl. Math. 36 (1992) 199-202.

[9] D.J. Skrien, A relationship between triangulated graphs, comparability graphs, proper interval graphs, proper circular arc graphs, and nested interval graphs, J. Graph Theory 6 (1982) 309-316.