

A LINEAR-TIME RECOGNITION ALGORITHM
FOR INTERVAL DAGS

by

Harold N. Gabow
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-180-80

July, 1980

A Linear-Time Recognition Algorithm
for Interval Dags

by

Harold N. Gabow*
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

Key Words: scheduling, dag, interval order, transitive closure.

1. Introduction

In [PY], Papadimitriou and Yannakakis solve the unit-execution-time scheduling problem [C,pp.51-3] for a family of partial orders called interval orders. Only one other family (the forest orders [H]) is known to have an efficient solution.

The development in [PY] assumes the interval order is given in transitively closed form. This note extends ideas that are implicit in [PY] to remove this assumption. The main tool is a linear-time algorithm to recognize an interval dag, i.e., a dag whose transitive closure is an interval order. This is given in Section 2. Section 3 indicates how the recognition algorithm applies to the scheduling problem.

Before proceeding we establish some notation and give a convenient definition of interval dag. For a dag (directed acyclic graph), n represents the number of vertices and m the number of edges; the vertices are numbered $1, \dots, n$. For a vertex i , the set of vertices adjacent from i is $A(i) = \{j \mid \text{there is an edge from } i \text{ to } j\}$; the successor set of i is $S(i) = \{j \mid \text{there is a path of one or more edges from } i \text{ to } j\}$.

An interval dag satisfies the following nesting property:

*This research was supported in part by National Science Foundation Grant MCS 78-18909.

Definition A dag is an interval dag iff for any two vertices i, j , either $S(i) \subseteq S(j)$ or $S(j) \subseteq S(i)$.

So the distinct successor sets $S(j)$ can be labelled as $\phi = T_0 \subsetneq T_1 \subsetneq \dots \subsetneq T_{\ell-1} \subsetneq T_\ell$. Figure 1 shows an interval dag. The transitively closed interval dags are exactly the interval orders; this can be seen by examining the proofs of [PY].

2. The Recognition Algorithm

The algorithm processes vertices one-by-one, checking the nesting property is always satisfied. To do this, the vertices are numbered in topological order [K], so if (i, j) is an edge, $i < j$ (Each vertex is identified with its number.). Vertices are processed in decreasing order.

Suppose the nesting property has been verified for all vertices $j > i$. So the distinct successor sets $S(j)$, $j > i$, can be labelled as $T_0 \subsetneq \dots \subsetneq T_{k-1} \subsetneq T_k$, where for convenience, T_k does not denote a successor set but rather $T_k = \{1, \dots, n\}$. Let T_q be the smallest set containing $A(i)$. The nesting property implies $T_{q-1} \subsetneq S(i) \subseteq T_q$. So if $T_{q-1} \subsetneq S(i)$ is false, the dag is not interval. Otherwise the dag seen so far is interval. If $S(i) = T_q$, the next vertex $i-1$ can be processed. If $S(i)$ lies properly between T_{q-1} and T_q , $S(i)$ can be made a new T- set and then vertex $i-1$ can be processed.

To implement this approach efficiently, the following data structure is used (see Figure 2). A set T_j is represented by a node t ; it has a field $COUNT(t) = |T_j|$.

The nodes T_k, T_{k-1}, \dots, T_0 are linked, in this order, to form a linear list L. A vertex i has entries in two arrays:

SUCC(i) points to the node for set $S(i)$;

IN(i) points to the node for the smallest set containing i .

Note the nesting property allows us to use COUNT fields to compare sets. Now we present the algorithm in pseudo-Algol.

begin comment This algorithm halts indicating whether or not the given dag is interval. If it is, a node t in L represents the successor set $\{i | IN(i) \text{ is } t \text{ or is after } t \text{ in } L\}$, L gives the nesting order of successor sets, and vertex i has successor set $SUCC(i)$;

initialization:

1. number the vertices in topological order;
2. let L be T_1, T_0 , where $T_1 = \{1, \dots, n\}$, $T_0 = \phi$; let $IN(i)$ point to T_1 and $SUCC(i)$ point to T_0 , for $1 \leq i \leq n$;

processing:

3. for $i \leftarrow n$ to 1 by -1 do
4. if $A(i) \neq \phi$ then begin
5. comment find T_q , the smallest successor set containing $A(i)$, and T_p , the largest successor set of some $j \in A(i)$;
 let q point to the node that maximizes $COUNT(IN(j))$ for $j \in A(i)$;
 let p point to the node that maximizes $COUNT(SUCC(j))$ for $j \in A(i)$;
6. comment let $LO = A(i) \cap T_{q-1} - T_p$, $HI = A(i) \cap T_q - T_{q-1}$;
 let $LO = \{j | j \in A(i), COUNT(p) < COUNT(IN(j)) < COUNT(q)\}$;
 let $HI = \{j | j \in A(i), IN(j) = q\}$;
7. comment find T_{q-1} and check $T_{q-1} \not\subseteq S(i)$;
 let q' point to the node following q in L ;
 if $COUNT(q') > COUNT(p) + |LO|$ then halt (G is not interval);
8. comment check if $S(i) = T_q$;
 if $COUNT(q) = COUNT(q') + |HI|$ then $SUCC(i) \leftarrow q$
 else begin comment create a new successor set for $S(i)$;
9. let t point to a new node between q and q' , with
 $COUNT(t) = COUNT(q') + |HI|$, $SUCC(i) = t$, and
 $IN(j) = t$ for $j \in HI$;

end end

halt (G is interval)

end.

Theorem: An interval dag can be recognized in time $O(n+m)$.

Proof: Correctness of the algorithm follows from the comments and the preceding discussion. The linear time bound is obvious. \square

In an actual implementation, the algorithm would be modified for greater speed. The topological numbering could be eliminated by exploring the graph depth-first, processing successors before predecessors. In line 6 it is more economical to compute only $|LO|$ and $|HI|$; only in line 9 might it be necessary to reexamine $A(i)$ to find HI .

3. Application to Scheduling

To implement the scheduling algorithm of Papadimitriou and Yannakakis, it is necessary to form a "priority list" of the vertices i in decreasing order of $|S(i)|$. This is done by traversing L , listing all vertices i with $SUCC(i) = t$ when node t is visited. It is easy to implement this in time $O(n+m)$.

The complete scheduling algorithm runs in almost-linear time, $O(n\alpha(n)+m)$.* Here $\alpha(n)$ is the inverse of Ackermann's function and is ≤ 3 for practical applications [T]. The factor $\alpha(n)$ results from the time to convert the priority list to a schedule $[S]$. There seems to be no simple way to take advantage of the special structure of interval dags to do the conversion faster.

*This is slightly worse than the linear bound claimed in [PY].

References

- [C] Coffman, E.G. Jr., Ed., Computer and Job-Shop Scheduling Theory, Wiley & Sons, New York, 1976.
- [H] Hu, T.C., "Parallel sequencing and assembly line problems," Op. Res. 9, 6, 1961, pp. 841-848.
- [K] Knuth, D.E., The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Mass., 1968.
- [PY] Papadimitriou, C.H. and Yannakakis, M., "Scheduling interval-ordered tasks," SIAM J. Comput. 8, 3, 1979, pp. 405-409.
- [S] Sethi, R., "Scheduling graphs on two processors," SIAM J. on Comp. 5, 1, 1976, pp. 73-82.
- [T] Tarjan, R.E., "Efficiency of a good but not linear set union algorithm," J.ACM 22, 2 (April 1975), 215-225.

Figure 1. Example graph

Figure 2. Data structure after vertex 3
is processed.



