1983

A Graph Orientation Problem

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
83-457

A Graph Orientation Problem

Mikhail J. Atallah

Department of Computer Sciences

Purdue University

West Lafayette, Indiana 47907.

ABSTRACT

We consider the following problem: Given a connected, undirected graph with a cost associated with every vertex, assign directions to its edges so that the resulting digraph is acyclic, has a root and is such that the sum of the costs of its sinks is as small as possible. We give a linear time algorithm for solving this problem if the costs are nonnegative, and prove that it is NP-hard if negative costs are allowed.

Keywords: Graph algorithms, depth-first search, NP-complete problems

1. Introduction

Graph orientation problems are usually of the following form: Given an undirected graph G, assign directions to its edges (i.e. *orient them*) so that the resulting digraph satisfies a given set of properties and/or is optimal in some sense. For example, if G is bridgeless then in linear time its edges can be oriented so that the resulting digraph is strongly connected: Simply find a depth-first spanning tree of G and orient the tree edges in the father-to-child direction and the other edges in the descendant-to-ancestor direction. Spinrad [S] has given an efficient algorithm for orienting the edges of G such that the resulting digraph is transitive, provided such a transitive orientation exists. If G is connected, then it is trivial to orient its edges so that the resulting digraph is rooted (has a vertex from which there is a path to every other vertex) and acyclic: Find a depth-first spanning tree of G and orient all the edges of G in the ancestor-to-descendant direction. Note, however, that in this case the resulting digraph has as many sinks as the number of leaves in the original depth-first tree (a sink is a vertex whose out-degree is zero). Suppose we are interested in orienting the edges of G so that, in addition to being acyclic and rooted, the resulting digraph has as few sinks as possible. In Sections 2 and 3, we give a linear time algorithm for finding solving this problem. In Section 4, we consider the weighted version of the problem, where a cost is associated with every vertex and we want the sum of the costs of the sinks to be as small as possible. We show that a linear time solution is still possible if the costs are nonnegative, and that the problem is NP-hard if negative costs are allowed.

We assume that the reader is familiar with depth-first search [T,AHU] and with standard graph-theoretic terminology. Throughout, n is the number of vertices of the graph under consideration and m is its number of edges. We assume that the input is an adjacency lists representation of that graph.

2. Orienting a Biconnected graph

Let $G=(V,E)$ be a biconnected undirected graph, and let s and t be any two distinct vertices of G. In this section we give a linear time algorithm for orienting the edges of G so that the resulting digraph is acyclic, rooted at s, and has t as its only sink.

Let $P=v_1 \cdots v_k$ $(v_1=s, v_k=t)$ be an $s-t$ path in G. P can be extended into a depth-first spanning tree T, so that the depth-first number of v_i is i $(1\leq i\leq k)$. We use T to denote both the tree and its set of edges (the context will make it clear which one is meant). From now on we refer to vertices using their depth-first number (so vertex 1 is s and vertex k is t). Let $F(i)$ $(2\leq i\leq n)$ denote the father of i in T, and define the lists $T(i)$ and $U(i)$ $(1\leq i\leq n)$ as follows:

$T(i)=\{j \mid i=F(j)\}$,

$U(i)=\{j \mid j<i,\ (i,j)\in E-T\}$.

In addition, let $LOW(i)$ $(2\leq i\leq n)$ be defined as follows:

$LOW(i)=MIN\{j \mid j=i$ or there exists an edge $(x,j)\in E-T$ such that x is a descendant of i and j is an ancestor of i in $T\}$.

Since G is biconnected, we have $LOW(i)<F(i)$ $(3\leq i\leq n)$ and, in addition, vertex 2 is the only child of 1 in T (for a proof, see [T] or [AHU]).

We describe the orientation algorithm assuming that the tree T is available, and that the $U(i)$'s, $F(i)$'s and $LOW(i)$'s have already been computed (these preliminary computations can be done in time $O(m+n)$ [T,AHU]). The orientation algorithm consists of a preorder traversal of T, during which whenever a node i is visited all the edges $\{(i,j) \mid j\in U(i)\}$ as well as the edge $(i,F(i))$ are oriented. The algorithm makes use of an array $STATUS$, where $STATUS(i)$ is *up* or *down* depending upon the orientation of the most recently oriented tree edge (i,x) $(i=F(x))$: If it was oriented in the x-to-i direction then $STATUS(i)$ is *up*,

otherwise $STATUS(i)$ is *down*. Note that $STATUS(i)$ may change between *up* and *down* as many times as the number of children of i in T minus one. The following is a pseudo-Pascal description of procedure *VISIT* which, when called with $VISIT(1)$, will produce the desired orientation of G. Recall that k is the vertex which is to become the sink in the resulting digraph D, that 1 is to become the root of D, and that the 1-to-k path in T consists of the vertices $1,2,\cdots,k$.

```
 Procedure VISIT(i);
 begin
1    if 2≤i≤k then
           begin
2           orient edge (i,F(i)) in the F(i)-to-i direction;
3           STATUS(F(i)) := down
           end
     else
          if i>k then
            begin
4           if STATUS(LOW(i))=down then
                  begin
5                 orient edge (i,F(i)) in the i-to-F(i) direction;
6                 STATUS(F(i)) := up
                  end
               else
                  begin
7                 orient edge (i,F(i)) in the F(i)-to-i direction;
8                 STATUS(F(i)) := down
                  end
           end;
9    for every j∈U(i) do
                  begin
10                 if STATUS(j)=down then
11                     orient edge (i,j) in the j-to-i direction
12                 else orient edge (i,j) in the i-to-j direction
                  end;
13   for every j∈T(i) do VISIT(j)
 end;
```

It is easy to see that $VISIT(1)$ runs in time $O(n+m)$. To prove correctness, we must show that the resulting digraph is acyclic, has vertex 1 as a root, and has vertex k as it unique sink. From now on, we use the shorthand $i \rightarrow j$ to stand for "edge (i,j) was oriented in the i-to-j direction".

Observation 1 Let x and y $(x=F(y))$ be consecutive vertices on the path in T from the root (vertex 1) to vertex z $(z \neq y)$. When z is visited, the value of

$STATUS(x)$ depends only on the way edge (x,y) was oriented: It is *up* if $y \rightarrow x$ and *down* if $x \rightarrow y$.

Proof: Follows from the definition of $STATUS(x)$. ▪

Corollary 2 As the subtree of k in T is visited, $STATUS(i)$ $(1 \leq i \leq k-1)$ remains *down*.

Proof: Follows from Observation 1 and the fact that $i \rightarrow i+1$ for $1 \leq i < k$ (lines 1-3). ▪

From now on we use $D=(V,A)$ to denote the digraph resulting from orienting the biconnected undirected graph $G=(V,E)$, using the algorithm described above.

Lemma 3 D is acyclic.

Proof: Suppose not, and let $C=w_0 \cdots w_l w_0$ be a directed cycle in D. Let $w_i = Min\{w_0, \cdots, w_l\}$. Note that w_{i-1} and w_{i+1} (subscripts are modulo $l+1$) must be descendants of w_i in T since if one of them (say, w_{i-1}) is not a descendant of w_i then the fact that $(w_{i-1},w_i) \in E$ would contradict the fact that T is depth-first. In addition, w_{i-1} and w_{i+1} are descendants of the same child of w_i because otherwise there is no way to complete the cycle C without passing through a proper ancestor of w_i, and this proper ancestor of w_i would have a number less than w_i, contradicting our choice of w_i as the lowest-numbered vertex on C. Let s be the child of w_i whose subtree contains both w_{i-1} and w_{i+1}. At least one of the two edges (w_{i-1},w_i) and (w_{i+1},w_i) is in $E-T$; without loss of generality, assume $(w_{i+1},w_i) \in E-T$. Then we distinguish two cases:

Case 1: $w_{i-1}=s$

In this case Observation 1 and the fact that $w_{i-1} \rightarrow w_i$ imply that when w_{i+1} was visited we had $STATUS(w_i)=up$. Since $w_i \in U(w_{i+1})$, lines 9-12 then imply that $w_{i+1} \rightarrow w_i$, a contradiction.

Case 2: $w_{i-1} \neq s$

In this case $STATUS(w_i)$ is the same when w_{i-1} is visited as when w_{i+1} is visited (this follows from Observation 1), and since $w_i \in U(w_{i-1})$ and $w_i \in U(w_{i+1})$, lines 9-12 imply that either $w_i \rightarrow w_{i-1}$ and $w_i \rightarrow w_{i+1}$ (this occurs if $STATUS(w_i)$ is *down* when w_{i-1} and w_{i+1} are visited, i.e. if $w_i \rightarrow s$), or $w_{i-1} \rightarrow w_i$ and $w_{i+1} \rightarrow w_i$ (this occurs if $s \rightarrow w_i$). This contradicts the fact that $w_{i-1} \rightarrow w_i$ and $w_i \rightarrow w_{i+1}$.

In either case we have a contradiction, and therefore D is acyclic. ▪

Lemma 4 Vertex 1 is a source in D, vertex k a sink, and every other vertex is neither a source nor a sink.

Proof (i) Vertex 1 is a source: The only tree edge incident on 1 is (1,2) and lines 1-2 imply that $1 \rightarrow 2$. As for nontree edges, let $(1,x)$ be one such edge. Observe that $STATUS(1)$ remains *down* throughout the algorithm, so that when vertex x is visited lines 9-11 imply that $1 \rightarrow x$.

(ii) Vertex k is a sink: We must show that for every $(x,k) \in E$, we have $x \rightarrow k$.

Case 1: $(x,k) \in T$

If $x = F(k)$ $(= k-1)$ then lines 1-2 imply that $x \rightarrow k$.

If $k = F(x)$ then, since $LOW(x) < k$ (because G is biconnected), it follows from Corollary 2 that $STATUS(LOW(x))$ is *down* when x is visited, and therefore lines 4-6 imply that $x \rightarrow k$. Note that this also implies that $STATUS(k)$ is set to *up* when its first child is visited and remains *up* throughout (this observation will soon be needed).

Case 2: $(x,k) \in E-T$

If $x < k$ (i.e. x is ancestor of k) then by Corollary 2 $STATUS(x)$ is *down* when k is visited and therefore lines 9-11 imply that $x \rightarrow k$.

If $x > k$ (i.e. x is descendant of k) then when x is visited $STATUS(k)$ is *up*

(as observed above) and therefore lines 9-12 imply that $x \to k$.

This completes the proof that k is a sink.

(iii) Every vertex other than 1 and k is neither source nor sink: We must show that for every $i \notin \{1,k\}$ we have $x \to i$ and $i \to y$ for some x and y.

If $1<i<k$ then lines 1-2 imply that $i-1 \to i$ and $i \to i+1$.

If $i>k$ then we distinguish two cases:

Case 1: $STATUS(LOW(i))$ is *down* when i is visited.

Lines 4-6 imply that $i \to F(i)$.

If $LOW(i) \in U(i)$ then lines 9-11 imply that $LOW(i) \to i$.

If $LOW(i) \notin U(i)$ then i is not a leaf and at least one of its children (say, j) has $LOW(j)=LOW(i)$. When j is visited, $STATUS(LOW(j))$ is still *down* and therefore lines 4-6 imply that $j \to i$.

Case 2: $STATUS(LOW(i))$ is *up* when i is visited.

Lines 4-6 imply that $F(i) \to i$.

If $LOW(i) \in U(i)$ then lines 9-12 imply that $i \to LOW(i)$.

If $LOW(i) \notin U(i)$ then i is not a leaf and at least one of its chidren (say, j) has $LOW(j)=LOW(i)$. When j is visited, $STATUS(LOW(j))$ is still *up* and therefore lines 4-8 imply that $i \to j$.

This completes the proof of Lemma 4. ▪

Lemmas 3 and 4 imply the following

Theorem 5 Let G be a biconnected undirected graph, and let s and t be distinct vertices of G. It is possible to orient G in time $O(n+m)$ such that the resulting digraph is acyclic, rooted at s, and has t as its only sink.

3. Orienting a Connected Graph

Let $G=(V,E)$ be a connected, undirected graph. In this section we give an $O(n+m)$ algorithm for orienting G so that the resulting digraph is acyclic, has a root and has as few sinks as possible. Before presenting the algorithm, we need a few preliminary definitions and observations.

Since G is connected, every one of its biconnected components (or *bicomponents*) contains one or more articulation points (or *cutnodes*). If a bicomponent contains exactly one cutnode then we say (as in [ET]) that it is *pendant*.

Theorem 6 Let G be a connected undirected graph and let p be the number of pendant bicomponents of G. Let D be a rooted acyclic digraph obtained by orienting the edges of G. Then D has at least $Max(1,p-1)$ sinks.

Proof: Any acyclic digraph must have at least one sink. If G is not biconnected, let G_0 be a pendant bicomponent of G that does not contain the root r of D, let x be the cutnode in G_0, and let D_0 be the subdigraph of D induced by the vertices of G_0. D_0 is acyclic and therefore must contain a source y and a sink z. Since r is the only source in D, it must be the case that $y=x$ since otherwise y would be a source in D, a contradiction. This implies that z is also a sink in D. Since there are at least $p-1$ bicomponents like G_0, D has at least $p-1$ sinks. ▪

We now give a linear time algorithm which achieves the bound of Theorem 6:

Step 1: Identify the bicomponents of G.

Step 2: If there is just one bicomponent (i.e. G is biconnected) then choose arbitrarily two vertices s and t as the desired root and sink (respectively), and then use the linear time algorithm of Section 2 to produce the desired digraph D, then Halt.

Otherwise proceed to Step 3 (G has more than one bicomponent).

Step 3: Identify the pendant bicomponents of G and call them $G_1, \cdots, G_p$ (note that $p \geq 2$). Let w_i be the cutnode in G_i $(1 \leq i \leq p)$.

Step 4: Choose a vertex r in G_1 as the desired root of D $(r \neq w_1)$. Use the algorithm of Section 2 to orient G_1 so that r is the root and w_1 is the sink in the digraph D_1 resulting from the orientation of G_1.

Step 5: For every G_i $(2 \leq i \leq p)$, use the algorithm of Section 2 to obtain an acyclic digraph D_i in which w_i is the root and an arbitrarily chosen node (call it v_i) is the sink.

Step 6: Determine the distance between r and every other vertex in G (this can be done by a breadth-first search starting at r).

Step 7: For every non-pendant bicomponent of G (call it H), do the following: Let x be the cutnode in H which is closest to r, and let y be any other cutnode. Then use the algorithm of Section 2 to obtain an acyclic digraph in which x is the root and y is the sink.

(End of Algorithm)

The above algorithm can easily be implemented to run in $O(m+n)$ time using the techniques described in [T,AHU]. We still have to prove correctness, i.e. that the resulting digraph D is acyclic and has one root and $Max(1,p-1)$ sinks.

If G is biconnected then correctness follows from Theorem 5, so assume from now on that G has more than one bicomponent. First, note that D is acyclic because every one of the digraphs resulting from the orientation of a bicomponent of G is acyclic. It suffices to show that, in D, r is the only source (and

hence the root, since G is connected) and $v_2, \cdots, v_p$ the only sinks. Steps 4,5 and 7 of the algorithm imply that any additional source or sink in D is a cutnode in G, and therefore it suffices to show that no cutnode of G is a source or a sink in D. Let z be a cutnode, and assume that the distance between z and r is the l^{th} smallest among the distances between cutnodes and r (ties are broken arbitrarily). We now prove that z is neither a source nor a sink in D.

If $l=1$ then z is actually the cutnode in G_1, i.e. $z=w_1$. w_1 has at least one incoming arc in D because it is a sink in D_1 (Step 4). It also has at least one outgoing arc in D because, in every bicomponent other than G_1 and containing w_1, w_1 is a source in the oriented version of that bicomponent (by Step 5 or Step 7).

If $l>1$ then two of the bicomponents to which z belongs (say, bicomponents A and B) are such that A contains a cutnode x such that r is closer to x than to z, while in B there is no such cutnode (B may be pendant and contain no cutnode other than z). Now, Step 7 and the fact that r is closer to cutnode x than to z imply that z is not the root of the oriented version of A, and therefore z has an incoming arc in D. We still have to show that z has an outgoing arc in D. If B is pendant then Step 5 implies that z is the root of the oriented version of B, and therefore z has an outgoing arc in D. If B is not pendant then Step 7 and the fact that no other cutnode in B is closer to r imply that z is the root of the oriented version of B, and therefore z has an outgoing arc in D.

This completes the correctness proof. We therefore have shown the following

Theorem 7 Let $G=(V,E)$ be a connected, undirected graph. Then it is possible to orient the edges of G in linear time in such a way that the resulting digraph is acyclic, rooted, and has as few sinks as possible.

4. The Weighted Version of the Problem

In this Section we consider the weighted version of the problem, where a cost is associated with every vertex of G. The problem then is that of orienting the edges of G so that the resulting digraph is acyclic, rooted, and has the sum of the costs of its sinks as small as possible (from now on we refer to the sum of the costs of the sinks of an acyclic, rooted digraph simply as the *cost* of that digraph).

4.1 Nonnegative Costs

Suppose that the cost of every vertex is a nonnegative number, and let c_i denote the cost of a smallest-cost vertex in $G_i - \{w_i\}$ $(1 \le i \le p)$. The algorithm of Section 3 can be modified to produce an optimal orientation, as follows:

In Step 2, if G is biconnected, rather than choosing an arbitrary vertex as the desired sink, select instead the lowest-cost vertex.

In Step 3, G_1 is such that $c_1 = \underset{1 \le i \le p}{Max}\ c_i$.

In Step 5, rather than choosing v_i arbitrarily, select v_i to have cost equal to c_i.

The modifications outlined above do not change the time complexity of the algorithm, which still results in an acyclic, rooted digraph. To see that the resulting digraph has minimum cost, note that in any acyclic, rooted digraph D resulting from the orientation of G, every $G_i - \{w_i\}$ must either contain the root of D or contain a sink of D (see the proof of Theorem 6). Therefore the lowest cost we can hope to achieve is $c_1 + \cdots c_p - \underset{1 \le i \le p}{Max}\ c_i$, which is precisely the cost of the digraph resulting from the modified algorithm. This completes the proof of the following

Theorem 8 Let G be a connected, undirected graph. If a nonnegative cost is associated with every vertex of G, then the problem of orienting the edges of G so that the resulting digraph is acyclic, rooted, and has minimum cost can be solved in linear time.

4.2 Negative Costs

We now show that if the costs of vertices are allowed to be negative then the problem is NP-hard.

Theorem 9 Let G be a connected, undirected graph. A (possibly negative) cost is associated with every vertex of G. Given G and a number α, the problem of determining whether it is possible to orient the edges of G so that the resulting digraph is acyclic, rooted, and has cost no greater than α is NP-complete.

Proof: It is easy to see that the problem is in *NP*. We now show that the *INDEPENDENT SET* problem [GJ] is polynomially reducible to this problem. Given an undirected graph H and an integer α, whether H contains an independent set of size $\geq\alpha$ can be determined by solving the following instance of this problem: Create undirected graph G by adding a vertex v_0 to H and joining v_0 to every vertex of H. Assign to every vertex of G a cost of -1. We claim that H has an independent set of size $\geq\alpha$ iff G has an orientation whose cost is $\leq -\alpha$. To prove this claim, let x be the size of the largest independent set in H, and let y be the cost of the optimal directed version of G. It clearly suffices to show that $y=-x$.

If G has an optimal orientation of cost y then the resulting digraph has $-y$ sinks. If v_0 is a sink then $y=-1$ (i.e. v_0 is the only sink) and in this case $x\geq -y$, since H trivially has an independent set of size 1. If v_0 is not a sink then the $-y$ sinks form an independent set in H, and again we have $x\geq -y$.

Now, let S denote the largest independent set in H ($|S|=x$). Note that $G'=G-S$ is connected (since v_0 is joined to all the other vertices) and therefore can be oriented so that the resulting digraph is acyclic and rooted, say, at v_0. Note also that every vertex of G' is joined in G to at least one vertex of S. Now, orient the edges of G so that those edges that also belong to G' are oriented as in the

above-mentioned orientation of G'. The remaining edges are assigned a direction *into* the set S. It is easy to see that the digraph resulting from this orientation of G is acyclic and rooted at v_0, and that its sinks are precisely the vertices in S (i.e. it has cost $=-x$). This implies that $y \leq -x$, and since we have already shown that $y \geq -x$ it follows that $y = -x$. ∎

5. Conclusion

We gave an linear time algorithm for orienting the edges of a connected undirected graph so that the resulting digraph is acyclic, rooted, and has as few sinks as possible. We also considered the weighted version of this problem, where a cost is associated with every vertex and we want to minimize the sum of the costs of the sinks in the resulting digraph. We showed that a linear time solution is possible if the weights are nonnegative, and that the problem is NP-hard if negative weights are allowed.

References

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

[ET] K.P. Eswaran and R.E. Tarjan, "Augmentation Problems," *SIAM Journal on Computing*, pp. 653-665, 1976.

[GJ] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[S] J. Spinrad, "Transitive Orientation in $O(n^2)$ time," *Proc. 15th Annual ACM Symposium on Theory of Computing*, pp. 457-465, 1983.

[T] R.E. Tarjan, "Depth-First-Search and Linear Graph Algorithms", *SIAM Journal on Computing*, pp. 146-160, 1972.