Shortest paths in almost acyclic graphs

Donald K. Wagner*

Mathematical, Computer, and Information Sciences Division, Office of Naval Research, Arlington, VA 22217-5660, USA

Received 12 August 1999; received in revised form 11 July 2000; accepted 2 August 2000

Abstract

This paper presents an algorithm for the shortest-path problem on a directed graph having arbitrary arc weights. One feature of the algorithm is its ability to exploit a certain type of structure. Two examples of this feature are highlighted. The first example is when the given graph is “almost” acyclic in the sense that there exists a small subset $T$ of nodes, the deletion of which yields an acyclic graph. In this case, a version of the algorithm solves the shortest-path problem in $O(|T|m)$ time; this bound is at least as good as the $O(mn)$ time bound of Bellman (Quart. Appl. Math. 16 (1958) 87–90). The second example is when the weight vector is “almost” nonnegative in the sense that only a small subset $F$ of arcs of the given graph have negative weight. In this case, the algorithm runs in $O(\min(|F|,n)(m + n \log n))$ time. © 2000 Elsevier Science B.V. All rights reserved.

1. Introduction

This paper presents an algorithm for the shortest-path problem (SPP). For the purposes of this paper, the SPP is defined as follows. The input is a directed graph $G = (N,A)$, a weight function $w : A \to \mathbb{R}$, and a distinguished source node $s$. The output is either the conclusion that $(G,w)$ has a negative-weight directed cycle, or in the absence of such a cycle, a shortest (i.e., minimum-weight) directed path from $s$ to every other node if such a path exists. In this paper, paths are considered to be simple; that is, no nodes are repeated.

The SPP is solvable in $O(mn)$ time, a bound that dates back to Bellman [2], where $m$ and $n$ are the number of arcs and nodes, respectively. In some cases, the SPP can be solved faster. One such case is when the underlying graph is acyclic. In this case, a very simple algorithm solves the problem in $O(m)$ time. Another well-known case is when all of the weights are nonnegative. In this case, the Fibonacci-heap version of Dijkstra’s algorithm runs in $O(m + n \log n)$ time. The book by Ahuja et al. [1] provides a good description of these and several other shortest-path algorithms.

* E-mail address: wagnerd@onr.navy.mil (D.K. Wagner).

0167-6377/00/$-see front matter © 2000 Elsevier Science B.V. All rights reserved.
PII: S 0167-6377(00)00054-7
The algorithm presented here is based on a general framework for exploiting structure in the SPP. The basic idea is as follows. Suppose that there exists a set \( F \) of “complicating” arcs; that is, the SPP on \( G \setminus F \) is easier to solve than that on \( G \). For instance, this is true if \( G \setminus F \) is acyclic (and \( G \) is not), or if \( G \setminus F \) has only nonnegative arc weights (and \( G \) does not). The main result of this paper shows that the SPP on \( G \) can be solved as a sequence of min\( \{ |F| + 1, n \} \) SPPs, each on a certain auxiliary graph, called here the surrogate graph. The surrogate graph is constructed from \( G \) by replacing each arc \( f \) of \( F \) with a surrogate arc, the tail and head of which are the source node of \( G \) and the head of \( f \), respectively. The weights on the surrogate arcs are modified after solving each problem in the sequence. In order for this approach to be computationally viable, it must be the case that the addition of the surrogate arcs to \( G \setminus F \) does not reintroduce the complicating structure that was eliminated by the removal of the arcs in \( F \). This is the case in both of the examples cited above. Specifically, in the case that \( G \setminus F \) is acyclic, then so is the corresponding surrogate graph, from which it follows that the SPP on \( G \) can be solved in \( O(\min\{ |F|, n \} \) time. Similarly, in the case that the weight vector of \( G \setminus F \) is nonnegative, then so is each one of the weight vectors in the sequence of SPPs on the corresponding surrogate graph; it follows that the SPP on \( G \) can be solved in \( O(\min\{ |F|, n \} (m + n \log n) \) time.

For the acyclic case, a variation of the algorithm is presented that handles complicating nodes. That is, suppose \( G \setminus T \) is acyclic for some subset \( T \) of nodes. Then using a simple node-splitting technique, \( T \) can be “converted” to a complicating set of arcs. Applying the above-mentioned result now yields an \( O(|T|m) \) algorithm for solving the SPP on \( G \). This bound, as well as the previously mentioned \( O(\min\{ |F|, n \} m) \) bound, are of interest because they are both at least as good as the \( O(mn) \) bound of Bellman. (It might be worth pointing out that both these bounds are general in the sense that one can always find a subset of arcs \( F \) or a subset of nodes \( T \) the deletion of which yields an acyclic graph.)

2. A shortest-path algorithm

Consider an instance of the SPP defined on a graph \( G \). Let \( F \) be a subset of arcs of \( G \) such that deleting \( F \) results in an instance of the SPP that is relatively easy to solve. For example, \( G \setminus F \) is acyclic or has only nonnegative arc weights. Obviously, the arcs in \( F \) cannot just be deleted and ignored. There needs to be a mechanism that accounts for them. To this end, surrogate arcs are introduced. Specifically, each arc \( (i, j) \in F \) is deleted from \( G \) and replaced by the surrogate arc \( (s, j) \), where \( s \) is the source node of \( G \). By convention, if two arcs from \( F \) have the same head, they are replaced by a single surrogate arc. The resulting graph \( H \) is called the surrogate graph with respect to \( F \). The set of surrogate arcs of \( H \) is denoted by \( S \). An initial weight vector \( w^H \) is defined on \( H \) as follows: the surrogate arcs are given infinite weight and all other arcs have the same weight as they had in \( G \).

Consider the following algorithm, called the Surrogate algorithm. The input is as described above, and the output is proved to be an optimal distance vector \( d \) for the SPP on \( G \). That is, \( d(v) \) is the weight of a shortest path from \( s \) to \( v \) in \( G \). The Surrogate algorithm operates by solving a sequence of at most \( |F| + 1 \) SPPs on the surrogate graph \( H \). The weights on the surrogate arcs are modified after solving each problem in the sequence. As is usual with shortest-path algorithms, the Surrogate algorithm maintains an approximate distance vector \( d' \) which gets refined during the course of the algorithm. The interpretation of \( d' \) (the correctness of which is proved below) is as follows: \( d'(v) \) is the weight of a shortest path from \( s \) to \( v \) in \( G \) subject to the constraint that the path contains at most \( t \) arcs from \( F \).

**algorithm** Surrogate;  
begin  
\( t := 0; \)  
\( w^0 := w^H; \)  
while \( t \leq \min\{ |F|, n - 1 \} \) do  


Notate that the algorithm only computes shortest-path distances, and not the actual shortest paths. To compute the actual shortest paths, one can use standard predecessor-function techniques; see, for example, Ahuja et al. [1]. These techniques also provide a way of detecting the existence of a negative-weight directed cycle.

Denote by SG(G,F,m,n) the time required to solve the SPP on the surrogate graph of G with respect to F, where G has m arcs and n nodes.

**Theorem 1.** Let (G,w,s) be an instance of the SPP having no negative-weight directed cycles, and let F be a nonempty subset of arcs of G. Then the Surrogate algorithm correctly computes an optimal distance vector d for (G,w,s) in O(min{|F|,n}(SG(G,F,m,n)+|F|)) time.

**Proof.** First consider the correctness of the algorithm. Let g' be defined as follows: for node v of G, g'(v) is the weight of a shortest path from s to v in G subject to the constraint that the path contains at most t arcs from F. Since a path of G can have at most n−1 arcs, it suffices to restrict t to be less than or equal to min{|F|,n−1}. It is claimed that d't = g' for 0 ≤ t ≤ min{|F|,n−1}, from which the correctness follows. The proof is by induction on t. Clearly the claim is true for t = 0.

Consider a node v, and let P be a shortest path from s to v in (H,w') for some t ≥ 1. If P does not use any arc from S, then P is a path of G that does not use any arcs from F, and so g'(v) ≤ d't(v). Now suppose that P does use an arc from S, say (s,j). Then, w'(s,j) = d't−1(i) + w(i,j) for some (i,j) ∈ F. By induction, there exists a directed path Q1 from s to i in G of weight d't−1(i) that contains at most t − 1 arcs from F. Let Q2 be the subpath of P from j to v. Then Q2 is also a path of G that contains no arcs from F. It follows that the concatenation of Q1, (i,j), and Q2, is a directed walk in G that contains at most t arcs from F and the weight of which is equal to that of P. Since (G,w) contains no negative-weight directed cycles, this walk contains a directed path in G from s to v of weight no more than that of P. (To see this, note that if the walk contains a cycle, this cycle is of nonnegative weight, and deleting this cycle yields another directed walk from s to v. Repeating this process eventually yields the desired path.) Thus, g'(v) ≤ d't(v).

To prove the reverse inequality, now let P be a shortest path from s to v in G subject to the constraint that P has at most t arcs from F. If P does not use any arc from F, then P is a path of H, and so d't(v) ≤ g'(v). Otherwise, let (i,j) be the last arc of P from F. Let Q1 be the subpath of P from s to i. Then, Q1 has at most t − 1 arcs from F, and so by induction, the weight of Q1 equals d't−1(i). Now, for the arc (s,j) ∈ S, w'(s,j) ≤ d't−1(i) + w(i,j). Let Q2 be the subpath of P from j to v. Then, the concatenation of (s,j) and Q2 is a directed path from s to v in (H,w') of weight less than or equal to that of P. Thus, d't(v) ≤ g'(v).

Finally, consider the running time of the algorithm. The while loop executes min{|F|,n−1} times. Each execution of the loop requires solving a SPP on the surrogate graph plus updating the weight vector. By definition, the former requires SG(G,F,m,n) time. The latter can be accomplished in O(|F|) time since it requires the examination of each arc in F exactly once.

One possible choice for F was suggested above. Namely, let F be the subset of arcs of G that have negative weight. Note, in this case, that the weight of a surrogate arc might become negative during the course of the algorithm. But it is well known that Dijkstra’s algorithm can solve a SPP that has negative-weight arcs if all
such arcs are incident to the source node. (To see this, note that an equivalent problem is obtained by adding any positive constant to all arcs emanating from the source node.) Thus, \( \text{SG}(G,F,m,n) = O(m + n \log n) \), and the Surrogate algorithm runs in time \( O(\min\{m + n\log n\}) \). In the worst case, this is slower than Bellman’s algorithm by a factor of \( \log n \). But if the graph has relatively few negative-weight arcs, then it is considerably faster.

A second possible choice for \( F \) is that of all of the arcs in \( G \). In this case, \( \text{SG}(G,F,m,n) = O(n) \). Thus, the running time of the Surrogate algorithm is \( O(mn) \), which is the same as Bellman’s. In fact, it is not hard to see that, in this case, the surrogate algorithm is a version of Bellman’s algorithm.

At the other end of the spectrum is choosing \( F = \emptyset \). In this case, Theorem 1 seemingly does not apply. It is easy to verify, however, that the correctness part of the proof is still valid, while the stated running time is incorrect. Note that \( F = \emptyset \) implies that the Surrogate algorithm just reduces to solving the SPP directly on \( G \). Thus, the running time of the Surrogate algorithm is equal to that of whatever algorithm is used to solve the SPP on \( G \). For example, if \( G \) happens to be acyclic, then the Surrogate algorithm runs in \( O(m) \) time.

A fourth possible choice for \( F \) is that of a feedback arc set, which is defined to be a subset of arcs the deletion of which yields an acyclic graph. (Note finding a feedback arc set is easy. Just order the nodes arbitrarily and let \( F \) be the set of arcs that go from “right” to “left”.) In this case, \( \text{SG}(G,F,m,n) = O(m) \) since the surrogate graph is acyclic. To see this, first observe that it can be assumed, without loss of generality, that the source node of \( G \) has indegree zero. Thus, adding the surrogate arcs to \( G \setminus F \) does not create any directed cycles. It follows that the running time of the Surrogate algorithm is \( O(\min\{|F|,n\}m) \). In the worst case, this is the same as Bellman’s algorithm. But if the graph is “almost” acyclic in the sense that the feedback arc set \( F \) is relatively small, then it is considerably better. For convenience, call this version of the Surrogate algorithm the Acyclic-Surrogate algorithm.

The Acyclic-Surrogate algorithm motivates the problem of finding a minimum-cardinality feedback arc set. This problem is known to be NP-hard; see Garey and Johnson [4]. Heuristic approaches have been developed; see, for example, Eades et al. [3]. It is easy to imagine cases where it might be worth the extra effort required to find a relatively small feedback arc set. For example, if one wants to solve several SPPs all on the same underlying graph, then savings gained by having a relatively small feedback arc set might offset the extra cost of computing such a set.

The complexity of the Acyclic-Surrogate algorithm can be improved slightly. Let \( T \) be a feedback node set of \( G \), which is defined to be a subset of nodes the deletion of which yields an acyclic graph. Then, it is shown below that the Acyclic-Surrogate algorithm runs in time \( O(|T|m) \). This is better than the above bound of \( O(\min\{|F|,n\}m) \) in the sense that for any feedback arc set \( F \), there exists a feedback node set \( T \) such that \( |T| \leq |F| \). To see this, just let \( T \) be the set of tails of the arcs in \( F \).

The idea of using the feedback node set \( T \) is simple. Imagine splitting each node of \( T \) into two nodes, and adding an arc joining these two nodes. If the splitting is done correctly, then the set of added arcs constitute a feedback arc set of the resulting graph. The Acyclic-Surrogate algorithm can then be run on this new graph using this feedback arc set. The node splitting can be done in at least a couple of different ways. Probably the simplest way is the following. Let \( v \) be a node in \( T \). Split \( v \) into two new nodes, say \( v^+ \) and \( v^- \), and add a new arc \((v^-,v^+)\); each arc \((j,v)\) is replaced by arc \((j,v^-)\), and each arc \((v,j)\) is replaced by arc \((v^+,j)\). Repeating this splitting process for each remaining node in \( T \) yields a graph \( H \). The set of new arcs (i.e., arcs of the form \((v',v^-)\)) constitute a feedback arc set \( F \) of \( H \) of cardinality \( |T| \). Thus, \( H \) has \( m + |T| \) arcs and \( n + |T| \) nodes. The SPP on \( G \) induces an equivalent instance on \( H \). Specifically, each arc in \( F \) is given a weight of zero, and each remaining arc of \( H \) inherits its weight from the corresponding arc of \( G \). The source node \( s \) of \( G \) can be assumed to have indegree zero, and thus can be assumed not to be in \( T \). Thus, \( s \) is also a node of \( H \). Now, it is straightforward to verify that solving the SPP on \( H \) with \( s \) as the source node is equivalent to solving it on \( G \). The Acyclic-Surrogate algorithm applied to \( H \) runs in time \( O(|T|m) \). This, of course, motivates the problem of finding a minimum-cardinality feedback node set. This problem is NP-hard; see Garey and Johnson [4]. Several heuristics are available; see, for example, Pardalos et al. [5].
One drawback of this feedback-node-set approach is that the resulting graph $H$ is bigger than $G$. One advantage, however, is that weight-updating step of the Surrogate algorithm becomes simpler. In particular, since no two of the feedback arcs of $H$ have a common head and since each arc in $F$ has weight zero, the for statement reduces to

$$\text{for each } (i,j) \in F \text{ do } w^{t+1}(s,j):=d'(i);$$

The above discussion is summarized in the following theorem.

**Theorem 2.** Let $(G,w,s)$ be an instance of the SPP. Let $F$ be a nonempty subset of arcs of $G$, and let $T$ be a nonempty subset of nodes of $G$. Then the Surrogate algorithm solves the SPP in

(i) $O(\min(|F|, n)(m + n \log n))$ time if $F = \{(i,j) | w(i,j) < 0\}$,
(ii) $O(mn)$ time if $F$ consists of all arcs of $G$,
(iii) $O(m)$ time if $G$ is acyclic,
(iv) $O(\min(|F|, n)m)$ time if $F$ is a feedback arc set, and
(v) $O(|T|m)$ time if $T$ is a feedback node set and node splitting is used.

**References**