Appl. Math. Lett. Vol. 6, No. 3, pp. 3–6, 1993
Printed in Great Britain

0893-9659/93 \$6.00 + 0.00
Pergamon Press Ltd

A HEURISTIC IMPROVEMENT OF THE BELLMAN-FORD ALGORITHM

ANDREW V. GOLDBERG*
Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A.
TOMASZ RADZIK†
SORIE, Cornell University, Ithaca, NY 14853, U.S.A.

(Received and accepted March 1993)

Abstract—We describe a new shortest paths algorithm. Our algorithm achieves the same $O(nm)$ worst-case time bound as Bellman-Ford algorithm but is superior in practice.

1. INTRODUCTION

The Bellman-Ford algorithm [1–3] is a classical algorithm for the single-source shortest paths problem. The algorithm runs in $O(nm)$ time on a graph with n nodes and m arcs. This is the best currently known strongly polynomial bound for the problem (see [4] for the best weakly polynomial bound).

In practice, however, the Bellman-Ford algorithm is usually out-performed by the *deque* algorithm of D'Escopo-Pape [5][1] and by the *two-queue* algorithm of Pallottino [7]. The work-case time bounds for these algorithms, however, are worse than those for the Bellman-Ford algorithm. The deque algorithm may take exponential time in the worst case [8] and the two-queue algorithm may take $\Omega(n^2m)$ time.

We propose a new *topological-scan* algorithm for the shortest paths problem. The algorithm combines, in a natural way, the ideas of the Bellman-Ford algorithm with those for finding the shortest paths in acyclic graphs (see, e.g., [9]). The algorithm achieves the same $O(nm)$ time bound as the Bellman-Ford algorithm, but out-performs it on most problem instances. A preliminary study shows that the algorithm is competitive with the deque and the two-queue algorithms.

2. DEFINITIONS AND NOTATION

The input to the single-source shortest paths problem is (G, s, ℓ), where $G = (V, E)$ is a directed graph, $\ell : E \to \mathbb{R}$ is a length function, and $s \in V$ is the source node. The goal is to find the shortest paths from s to all other nodes of G or to find a negative length cycle in G. If G has a negative length cycle, we say that the problem is *infeasible*. We assume, without loss of generality, that all nodes are reachable from s in G and that G has no multiple arcs. The latter assumption allows us to refer to an arc by its endpoints without ambiguity. We denote $|V|$ by n and $|E|$ by m.

A *potential function* is a real-valued function on nodes. Given a potential function d, we define a *reduced cost function* $\ell_d : E \to \mathbb{R}$ by

$$\ell_d(v, w) = \ell(v, w) + d(v) - d(w).$$

*A.V. Goldberg was supported in part by ONR Young Investigator Award N00014-91-J-1855, NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T, DEC, and 3M, and a grant from Powell Foundation.

†T. Radzik was supported by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF Grant DMS-8920550, and by the Packard Fellowship of Éva Tardos.

[1]This algorithm was discovered independently by Levit [6].

Given a potential function d, we say that an arc a is *admissible* if $\ell_d(a) \leq 0$, and denote the set of admissible arcs by E_d. The *admissible graph* is defined by $G_d = (V, E_d)$.

A *shortest paths tree* of G is a spanning tree rooted at s such that for any $v \in V$, the reversal of the v to s path in the tree is a shortest path from s to v.

3. THE LABEL-CORRECTING METHOD

Our algorithm is based on the general *label-correcting* method for solving the shortest paths problem. In this section, we briefly outline the method. (See, e.g., [9,10] for more detail.)

For every node v, the method maintains its potential $d(v)$, parent $\pi(v)$, and status $S(v) \in \{unreached, labeled, scanned\}$. Initially for every node v, $p(v) = \infty$, $\pi(v) =$ nil, and $S(v) \leftarrow$ unreached. The method starts by setting $d(s) \leftarrow 0$ and $S(s) =$ labeled, and applies the SCAN operation to labeled nodes until none exists, in which case the method terminates.

The SCAN operation applies to a labeled node v. The operation is described in Figure 1. Note that if v is labeled, then $d(v) < \infty$ and $d(v) + \ell(v, w)$ is finite.

```
procedure SCAN($v$);
    for all $(v, w) \in E$ do
        if $d(v) + \ell(v, w) < d(w)$ then
          $d(w) \leftarrow d(v) + \ell(v, w)$;
          $S(w) \leftarrow$ labeled;
          $\pi(w) \leftarrow v$;
          $S(v) \leftarrow$ scanned;
        end.
```

Figure 1. The scan operation.

The method terminates if and only if G does not have negative length cycles. If the method terminates, the parent pointers define a correct shortest paths tree and, for any $v \in V$, $d(v)$ is the shortest path distance from s to v. The label-correcting method can be easily modified so that if G has negative cycles, the method finds such a cycle and terminates.

Different strategies for selecting labeled nodes to be scanned lead to different algorithms. The Bellman-Ford algorithm maintains the set of labeled nodes in a FIFO queue. The next node to be scanned is removed from the head of the queue, a node that becomes labeled is added to the tail of the queue.

The D'Escopo-Pape algorithm uses a deque (a queue that allows insertions at either end) to maintain the labeled nodes. The next node to be scanned is removed from the head of the queue. A node that becomes labeled is added to the tail of the queue if this is the first time this node became labeled, and to the head of the queue otherwise (i.e., if it has been scanned previously).

Pallottino's two-queue algorithm uses two FIFO queues, one of low priority and another of high priority. Each labeled node appears on exactly one of the queues. The next node to be scanned is removed from the head of the high priority queue if the queue is not empty and from the head of the low priority queue otherwise. A node that becomes labeled is added to the tail of the low priority queue if this is the first time this node became labeled, and to the tail of the high priority queue otherwise.

Dijkstra's algorithm [11] maintains the set of labeled nodes in a priority queue and selects a labeled node with the minimum potential as the next node to be scanned. When the length function is nonnegative, this algorithm runs in $O(m + n \log n)$ time if Fibonacci heaps [12] are used to implement the priority queue. If the graph has negative length arcs, the algorithm can take exponential time.

4. THE TOPOLOGICAL-SCAN ALGORITHM

Suppose both v and w are labeled and $\ell_d(v, w) < 0$. Then it is better to scan v before w, since we know that when we scan v, $d(w)$ will impfove and w will become labeled. The topological-scan algorithm uses a generalization of this idea.

To simplify the algorithm description, we assume that G has no cycles of length zero or less, and therefore, for any d, G_d is acyclic. Later, we show how to get rid of this assumption.

The topological-scan algorithm maintains the set of labeled nodes in two sets, A and B. Each labeled node is in exactly one set. Initially $A = \emptyset$ and $B = \{s\}$. At the beginning of each *pass*, the algorithm uses the set B to compute the set A of nodes to be scanned during the pass, and resets B to the empty set. A is a linearly ordered set. During the pass, elements are removed according to the ordering of A and scanned. The newly created labeled nodes are added to B. A pass ends when A becomes empty. The algorithm terminates when B is empty at the end of a pass.

The algorithm computes A from B as follows.

1. For every $v \in B$ that has no outgoing arc with negative reduced cost, delete v from B and mark it as scanned.
2. Let A be the set of nodes reachable from B in G_d. Mark all nodes in A as labeled.
3. Apply topological sort to order A so that for every pair of nodes v and w in A such that $(v, w) \in G_d$, v precedes w and, therefore, v will be scanned before w.

THEOREM 4.1. *The topological-scan algorithm is correct.*

PROOF. Note that if a node v has no outgoing arc with negative reduced cost, then the only side-effect of SCAN(v) will be the change of the status of v to scanned. Note also that the label-correcting method remains correct, if we make a scanned node into a labeled node, and that all nodes reachable from B in G_d at Step 2 have finite potentials and, therefore, are either labeled or scanned. Finally, recall that we assumed that G_d is acyclic, and thus Step 3 is well-defined. These observations and the fact that the label-correcting method is correct imply that the algorithm is correct. ∎

Analysis of the topological-scan algorithm is very similar to that of the Bellman-Ford algorithm, as given for example in [9,10].

THEOREM 4.2. *The topological-scan algorithm runs in $O(nm)$ time.*

Now suppose G has cycles of zero or negative length. In this case G_d need not be acyclic. If, however, G_d has a negative length cycle, we can terminate the computation. If G_d has zero length cycles, we can contract such cycles and continue the computation. This can be easily done while maintaining the $O(nm)$ time bound. (See, e.g., [4].)

5. CONCLUDING REMARKS

To obtain a practical implementation of the algorithm, it is important to implement the procedure of constructing A from B efficiently. This construction can be done in one depth-first search computation. It uses stacks to implement A and B. The computation pops nodes from B one by one. For each removed node v, if v was already visited by the current depth-first search, we do nothing. If v was not visited and v has no outgoing arcs of negative reduced cost, we mark v as scanned. If v was not visited and has an outgoing arc of negative reduced cost, we visit in the depth-first order all nodes reachable from v in G_d and not visited previously. At the end of the depth-first search visit of a node, the node is marked labeled and pushed on A. Note that this procedure can be implemented so that it looks only at the nodes which were on B at the beginning of the computation or are on A at the end and each such node is examined exactly once. Thus, if the average number of labeled nodes at the beginning of every phase is small, the average time per phase is small.

An alternative way of dealing with potential cycles in G_d is to ignore some arcs of G_d and produce a topological ordering with respect to the (acyclic) graph G' obtained from G_d by deleting the ignored arcs. A standard way to implement a topological sort is by using depth-first search. If we ignore the arcs that depth-first search classifies as back arcs (see, e.g., [9]), then we obtain a topological ordering of nodes with respect to G'. To assure that the algorithm detects negative cycles, we can use standard techniques developed for the label-correcting method. The resulting algorithm still runs in $O(nm)$ time.

REFERENCES

1. R.E. Bellman, On a routing problem, *Quart. Appl. Math.* **16**, 87–90 (1958).
2. L.R. Ford, Jr. and D.R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, (1962).
3. E.F. Moore, The shortest path through a maze, In *Proc. of the Int. Symp. on the Theory of Switching*, pp. 285–292, Harvard University Press, (1959).
4. A.V. Goldberg, Scaling algorithms for the shortest paths problem, In *Proc. 4nd ACM-SIAM Symposium on Discrete Algorithms*, pp. 222–231, (1993).
5. U. Pape, Implementation and efficiency of Moore algorithms for the shortest root problem, *Math. Prog.* **7**, 212–222 (1974).
6. B.-Ju. Levit and B.-N. Livshits, *Neleneinye Setevye Transportnye Zadachi,* (in Russian), Transport, Moscow, (1972).
7. S. Pallottino, Shortest-path methods: Complexity, interrelations and new propositions, *Networks* **14**, 257–267 (1984).
8. D. Shier and C. Witzgall, Properties of labeling methods for determinimg shortest paths trees, *J. Res. Natl. Bur. Stand.* **86**, 317–330 (1981).
9. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, (1990).
10. R.E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, (1983).
11. E.W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.* **1**, 269–271 (1959).
12. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. Assoc. Comput. Mach.* **34**, 596–615 (1987).