Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Technical Note—Determining All Optimal and Near-Optimal Solutions when Solving Shortest Path Problems by Dynamic Programming

Thomas H. Byers, Michael S. Waterman,

To cite this article:
Thomas H. Byers, Michael S. Waterman, (1984) Technical Note—Determining All Optimal and Near-Optimal Solutions when Solving Shortest Path Problems by Dynamic Programming. Operations Research 32(6):1381-1384. https://doi.org/10.1287/opre.32.6.1381

Determining All Optimal and Near-Optimal Solutions when Solving Shortest Path Problems by Dynamic Programming

THOMAS H. BYERS
Digital Research Inc., Monterey, California

MICHAEL S. WATERMAN
University of Southern California, Los Angeles, California

(Received May 1982; accepted October 1983)

This paper presents a new algorithm for finding all solutions with objective function values in the neighborhood of the optimum for certain dynamic programming models, including shortest path problems. The new algorithm combines the depth-first search with stacking techniques of theoretical computer science and principles from dynamic programming to modify the usual backtracking routine and list all near-optimal policies. The resulting procedure is the first practical algorithm for a variety of large problems that are of interest.

THE ALGORITHM presented in this paper was motivated by a study of the evolutionary distance problem in molecular biology. In this context, dynamic programming methods are used to investigate evolutionary relationships between two DNA sequences (Smith et al. [1981]). The specific sequences studied implied a network of approximately 2,200 nodes and 110,000 arcs so that analysis by Kth shortest path methods was not practical. Details of this study have been published elsewhere (Waterman [1983]).

Consider a directed acyclic network or, more generally, a network with no cycle whose length is nonpositive. A simple method is presented for finding all paths from node 1 to node N whose lengths are within a prescribed distance $e(e \geq 0)$ of the length of the shortest path(s) from node 1 to node N. The algorithm uses a push-down (last-in, first-out) stack and has modest memory requirements. This new method is easy to understand and to code, which, with the memory requirements, accords it a special advantage over Kth shortest path calculations. See Dreyfus [1969] for a review of shortest path algorithms.

To describe the new method let $t(x, y)$ denote the length of arc (x, y) in the network. With $f(N) = 0$, let $f(x)$ denote the length of the shortest

Subject classification: 111 near-optimal policies.

Operations Research
Vol. 32, No. 6, November–December 1984

0030-364X/84/3206-1381 $01.25

path(s) from node $x (x \neq N)$ to node N. For methods that compute f-values, see Dreyfus and Law [1976] or Denardo [1982]. Consider a node $x \neq N$. Some path P of length d led us from node 1 to node x. Arc (x, y) is now said to *enter* if

$$d + t(x, y) + f(y) \leq f(1) + e. \quad (1)$$

Hence, the arcs that enter are those on paths from node 1 to node N having path length within e of the shortest path length.

The depth-first procedure lets one use the same path P for each entry

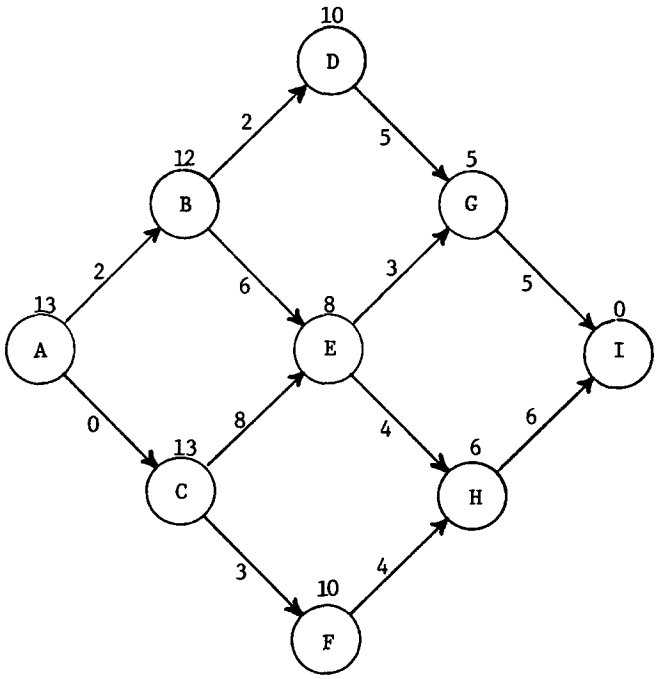

Figure 1. Example network.

in the stack (push-down list). This approach keeps the data in the stack itself small. Each *entry* in the stack has three attributes:

x = a next-to-last node
y = a last-node
c = the length of the path $(1, \cdots, x, y)$ having a subpath $(1, \cdots, x)$ in P.

The algorithm is as follows:

1. Set $P = (1)$ and $x = 1$. Then for each arc $(1, y)$ that satisfies $t(1, y) + f(y) \leq f(1) + e$, create an entry $(1, y, c)$ in the stack with $c = t(1, y)$.

2. Stop if the stack is empty.
3. Remove (POP) the topmost entry (x, y, c) in the stack. Replace $P = (1, \cdots, x)$ by $P = (1, \cdots, x, y)$. If $y = N$, go to Step 4. If $y \neq N$, let $x \leftarrow y$ and $d \leftarrow c$. Then for each arc (x, y) satisfying (1), create an entry (x, y, c) in the stack with $c = d + (t(x, y)$. Go to Step 2.
4. Output P and c. Go to Step 3.

In the case of an acyclic network, the number of elements in the stack at any one time is at most the number $|A|$ of arcs in the network. Since (x, y, c) in the stack implies arcs leaving y are not associated with the stack, the actual stack size is much smaller than this bound indicates. In a cyclic network whose shortest cycle has length $L > 0$, the number of elements in the stack is at most $|A| \lceil e/L \rceil$, where $\lceil z \rceil$ is the smallest integer a that satisfies $a \geq z$.

In Figure 1, the shortest path from node A to node I has a length of 13; path lengths are shown above the nodes. The method is illustrated by computing all paths from node A to node 1 whose lengths are within 2.6 units ($e = 2.6$ which is 20% of 13) of the shortest path length.

Step	Entries			Path P
	x	y	c	
1	A	B	2	(A)
	A	C	0	
2	B	D	4	(A, B)
	A	C	0	
3	D	G	9	(A, B, D)
	A	C	0	
4	G	I	14	(A, B, D, G)
	A	C	0	
5	A	C	0	Output (A, B, D, G, I), 14
6	C	F	3	(A, C)
7	F	H	7	(A, C, F)
8	H	I	13	(A, C, F, H)
9				Output (A, C, F, H, I), 13

ACKNOWLEDGMENT

The second author received support from the System Development Foundation. The authors are grateful to the referees for their very useful comments and suggestions.

REFERENCES

DENARDO, E. 1982. *Dynamic Programming: Models and Applications.* Prentice-Hall, Englewood Cliffs, N.J.

DREYFUS, S. 1969. Appraisal of Some Shortest Path Algorithms. *Opns. Res.* **17,** 395–412.

DREYFUS, S., AND A. LAW. 1976. *The Art and Theory of Dynamic Programming.* Academic Press, New York.

SMITH, T., M. WATERMAN AND W. FITCH. 1981. Comparative Biosequence Metrics. *J. Mol. Evol.* **18,** 38–46.

WATERMAN, M. 1983. Sequence Alignments in the Neighborhood of the Optimum. *Proc. Natl. Acad. Sci. U.S.A.* **80,** 3123–3124.