OPTIMAL CYCLES IN GRAPHS AND THE MINIMAL COST-TO-TIME RATIO PROBLEM

by

Eugene L. Lawler

Memorandum No. ERL-M343

10 May 1972

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

OPTIMAL CYCLES IN GRAPHS AND THE MINIMAL COST-TO-TIME RATIO PROBLEM

Eugene L. Lawler

Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

Let $G = (N,A)$ be a directed graph, and let c_{ij} and t_{ij} be a "cost" and a "transit time" assigned to each arc (i,j). We consider three problems. The Negative Cycle Problem: Does G contain a directed cycle for which the sum of the costs is strictly negative? The Minimal-Cost Cycle Problem: Assuming G contains no negative cycles, find a directed cycle for which the sum of the costs in minimum. The Minimal Cost-to-Time Ratio Cycle Problem: Find a directed cycle for which the sum of the costs divided by the sum of the transit times is minimum.

The first two problems can be solved by adaptations of well-known shortest path algorithms which require $O(n^3)$ computational steps, where n is the number of nodes of G. The minimal cost-to-time ratio problem can be solved by a procedure which uses a negative cycle algorithm as a subroutine. This procedure is essentially $O(n^3 \log n)$ in complexity.

Various generalizations of these problems, in the form of side constraints on the cycles, are indicated.

Research sponsored by the Naval Electronic Systems Command, Contract N00039-71-C-0255.

To be presented at Conference on Periodic Optimization, CISM, Udine, Italy, June 1972.

1. Introduction

Let G = (N,A) be a directed graph where N is the set of nodes and A is the set of directed arcs. Let c_{ij} be a (positive or negative) "cost" and t_{ij} be a "transit time" assigned to arc (i,j).

We consider three problems:

Negative Cycle Problem: Does G contain a directed cycle for which the sum of the costs is strictly negative? And if so, find such a cycle.

Minimal-Cost Cycle Problem: Assuming G contains no negative cycles, find a directed cycle for which the sum of the costs is minimum.

Minimal Cost-to-Time Ratio Cycle Problem: Find a directed cycle for which the sum of the costs divided by the sum of the transit times is minimum.

The first two problems can be solved by adapting well-known shortest-path algorithms for the purpose. We review the algorithms of Bellman-Ford and Floyd-Warshall and indicate their application to these problems.

The minimal cost-to-time ratio problem can be solved by a procedure which uses a negative cycle algorithm as a subroutine. We show that the ratio problem can be solved in a number of computational steps that is a relatively low-order polynomial function of the dimension of the problem. (Essentially $O(n^3 \log n)$ for an n-node graph, compared with $O(n^3)$ for a single shortest path computation.)

We conclude by indicating various generalizations of these problems in the form of side constraints on the cycles.

Remark:

(1.1) In this paper we deal exclusively with directed networks. Analogous results and computational methods can be obtained for undirected networks. However, the theory and the computational procedures are

necessarily much more sophisticated. E.g. shortest path problems in undirected networks are solved by the matching algorithms of Edmonds [5].

2. Bellman-Ford Shortest Path Algorithm

Let c_{ij} be the "cost" of arc (i,j) if there is such an arc, and $+\infty$ if no such arc exists. Let $c_{ij} = 0$, if $i = j$.

Suppose we seek to find a least-costly path from node 1 to each of the other nodes, $i = 2, 3, \ldots, n$. Let

$$c_i^* = \text{the cost of a least-costly path from node 1 to node } i.$$

If there are no negative cycles, the values c_i^*, $i = 1,2,\ldots, n$, must satisfy the following equations:

$$(2.1)\quad \begin{cases} c_1^* = 0, \\ c_i^* = \min\limits_{k \neq i} \{c_k^* + c_{ki}\}. \end{cases}$$

The nonlinear equations (2.1) imply implicit functional relationships between the variables c_i^*. Bellman [1] and Ford [8] have proposed solving these equations by a method of "successive approximations:"

$$(2.2)\quad \begin{cases} c_i^{(1)} = c_{1i}, \\ c_i^{(m+1)} = \min\limits_{k} \{c_k^{(m)} + c_{ki}\}. \end{cases}$$

Because $c_{ii} = 0$, for all i, it is easy to see that $c_i^{(1)} \geq c_i^{(2)} \geq c_i^{(3)} \geq \ldots \geq c_i^{(m)} \geq c_i^{(m+1)}$.

We can understand the equations (2.2), and demonstrate convergence

to c_i^*, by applying the following interpretation:

$c_i^{(m)}$ = the cost of a least-costly path from node 1 to node i, subject to the constraint that the path contains no more than m arcs.

In the absence of negative cycles, there exists an optimal path to any node i with no more than n-1 arcs. (If a path contains n or more arcs, at least one node is repeated, and the path contains a cycle.) It follows that $c_i^{(m)} = c_i^*$, for all i and all $m \geq n-1$.

The complexity of the computation can be estimated as follows. It is necessary to compute $c_i^{(m)}$ for i = 2,3,..., n and for m = 2,3,..., n-1. Each such computation, by (2.2), requires n additions and n-1 comparisons. Thus, approximately n^3 additions and n^3 comparisons all required in all.

3. Negative Cycle Computation

We propose to use the Bellman-Ford algorithm to detect the existence of negative cycles. Although the results we state do not strictly require it, it is reasonable to suppose that the graph G is strongly connected. That is, for any pair of nodes i,j there exists a path from i to j and a path from j to i. If G is not strongly connected, it is best to identify its strongly connected components and analyze the components separately.

We leave the proof of the theorem below as an exercise for the reader.

Theorem 3.1

Let G have a path from node 1 to each of the other nodes. Then G contains a negative cycle if and only if, in (2.2), $c_i^{(n)} < c_i^{(n-1)}$, for at least one i = 1,2,..., n.

Theorem 3.1 suggests that to detect the existence of negative cycles, all that is necessary is to carry out the computation of (2.2) for one

additional iteration. Thus, the complexity of the computation remains at approximately n^3 additions and n^3 comparisons.

It may be possible to halt the computation earlier by testing for other conditions which are sufficient to indicate the existence or non-existence of negative cycles. One condition is that $c_1^{(m)} < 0$, for any m (there is a negative cycle containing node 1). Another such condition is that $c_i^{(m+1)} = c_i^{(m)}$, for all i, and for any m (there are no negative cycles). Still another set of such conditions is given by the following theorem.

Theorem 3.2

G contains a negative cycle if, in (2.2), $c_i^{(m+1)} < c_i^{(m)}$, for some $m = 1,2,3,\ldots, n-1$, and for at least n-m distinct nodes i.

These sufficient conditions may well enable the computation to be ended earlier, but they do not affect the worst-case bound of n^3 additions and n^3 comparisons.

Remarks:

(3.1) We have made no attempt to find a "most negative" negative cycle. Any procedure capable of solving that problem could be applied to solve the traveling salesman problem as well. (Subtract a sufficiently large number from the length of each arc in the traveling salesman problem. A minimum-length Hamiltonian tour is then identified with a most-negative cycle.)

(3.2) Any procedure for solving the assignment problem can be used to solve the negative cycle problem. Solve the assignment problem for the matrix of arc costs, with $c_{ii} = 0$. Then an optimal solution to the assignment problem yields a most negative union of disjoint negative

cycles, if negative cycles exist. Note: Assignment algorithms are generally $O(n^3)$ in complexity.

(3.3) We have not described how to construct a negative cycle, when such a cycle is detected. In the tradition of dynamic programming, we simply assert that by appropriate record-keeping (particularly of the choices of k that give rise to minimum values in (2.2)), one can construct such cycles relatively easily as a byproduct of the computation.

4. Improvement in Efficiency of Bellman-Ford Algorithm

A close examination of the computations implied by equations (2.2) reveals that these computations may not be as efficient as they could be. They do not make use of the best information available at each iteration. For example, $c_i^{(m+1)}$ is computed as a function of $c_1^{(m)}$, $c_2^{(m)}$,..., $c_n^{(m)}$, even though $c_1^{(m+1)}$, $c_2^{(m+1)}$,..., $c_{i-1}^{(m+1)}$ may already have been computed. Making use of these (m + 1)st order approximations, when available, might accelerate convergence. This is indeed possible, as has been suggested by J. Y. Yen [2]

Suppose we call an arc (i,j) rightward if i < j and leftward if i > j. A path is said to contain a change in direction whenever a rightward arc is followed by a leftward arc, or vice-versa. Note that because node 1 is the first node of any path, the first arc is rightward and the first change in direction (if any) must be from right to left.

Let us assign a new interpretation to $c_i^{(m)}$. Let

$c_i^{(m)}$ = the cost of a least-costly path from node 1 to node i, subject to the constraint that the path contains no more than m-1 changes in direction.

The appropriate equations for $c_i^{(m)}$ are:

$$(4.1)\quad \begin{cases} c_i^{(0)} = c_{1i}, \quad i = 1,2,\ldots, n \\ c_i^{(m+1)} = \min \{c_i^{(m)}, \min_{k<i} \{c_k^{(m)} + c_{ki}\}\}, \ m \text{ even.} \\ c_i^{(m+1)} = \min \{c_i^{(m)}, \min_{k>i} \{c_k^{(m)} + c_{ki}\}\}, \ m \text{ odd.} \end{cases}$$

There exists an optimal path to any node i with no more than n-1 arcs and hence no more than n-2 changes in direction. It follows that $c_i^{(n-1)} = c_i^*$. Each equation is solved by a minimization over about $\frac{n}{2}$ alternatives, on the average, instead of n, as in (2.2). Accordingly, the computational complexity has been reduced by a factor of approximately two: about $\frac{1}{2} n^3$ additions and $\frac{1}{2} n^3$ comparisons are required.

Another, possibly less important, advantage is that storage requirements are also reduced by a factor of approximately two, since not both $c_i^{(m+1)}$ and $c_i^{(m)}$ must be stored. (As soon as $c_i^{(m+1)}$ is computed, it replaces $c_i^{(m)}$.)

Theorems 3.1 and 3.2 apply to Yen's modification (with (4.1) substituted for (2.2) in the statement of the theorems). Interestingly, Theorem 3.2 is the key to a still further improvement in efficiency. Yen [15] has shown how the length of the computation can be reduced to approximately $\frac{1}{4} n^3$ additions and $\frac{1}{4} n^3$ comparisons by exploiting the fact that, at each iteration, one additional value $c_i^{(m)}$ becomes equal to c_i^* and therefore affects the calculations no more thereafter.

5. Floyd-Warshall Shortest Path Algorithm [7]

Now suppose we see to find a least-costly path from node i to node j, for all ordered pairs i,j. Let

$$c_{ij}^{*} = \text{the cost of a least-costly path from node i to node j.}$$

We propose to compute c_{ij}^{*} by a method of successive approximations, as follows:

$$(5.1)\begin{cases} c_{ij}^{(1)} = c_{ij} \\ c_{ij}^{(m+1)} = \min \{c_{ij}^{(m)}, c_{im}^{(m)} + c_{mj}^{(m)}\} \end{cases}$$

We give $c_{ij}^{(m)}$ the following interpretation:

$$c_{ij}^{(m)} = \text{the cost of a least-costly path from node i to node j, subject to the constraint that the path does not pass through any of the nodes m, m+1,..., n (i,j excepted).}$$

The equations (5.1) call for minimization over two alternatives. It is now clear that the first of these alternatives ($c_{ij}^{(m)}$) is that the desired path does not pass through node m and the second alternative ($c_{im}^{(m)} + c_{mj}^{(m)}$) is that it does. With this insight, it becomes easy to justify the equations (5.1) and to see that $c_{ij}^{(n+1)} = c_{ij}^{*}$.

We estimate the complexity of the Floyd-Warshall algorithm as follows. It is necessary to compute $c_{ij}^{(m)}$ for i = 1,2,..., n, for j = 1,2,..., n and for m = 2,3,..., n+1. Each such computation, by (5.1), requires one addition and one comparison. Thus, exactly n^3 additions and n^3 comparisons are required overall.

6. Min-Cost Cycle Problem

The theorem below requires no proof.

Theorem 6.1

G contains a negative cycle if and only if, in (5.1), $c_{ii}^{(m)} < 0$, for some i = 1,2,..., n and some m = 2,3,..., n+1.

The Floyd-Warshall algorithm has essentially the same upper bound on the number of computational steps as the (unmodified) Bellman-Ford algorithm. However, in practice it appears that the Bellman-Ford algorithm is much more likely to terminate early, and should be preferred for the negative cycle problem.

The Floyd-Warshall method is more useful, for our purpose, to solve the minimal-cost cycle problem.

Theorem 6.2

If G does not contain a negative cycle, then the cost of a least-costly cycle is given by

$$\min_i \left\{ c_{ii}^{(n+1)} \right\}$$

where $c_{ii}^{(n+1)}$ is determined by (5.1), with $c_{ii} = +\infty$, for all i.

7. Minimal Cost-to-Time Ratio Problem

Recall that in addition to a cost c_{ij}, each arc (i,j) of G is assigned a transit time t_{ij}. We now wish to find a directed cycle C for which the cost-to-time ratio,

$$\frac{\sum_C c_{ij}}{\sum_C t_{ij}} \tag{7.1}$$

is minimal.

As an example of this type of problem, consider the following

situation suggested by Dantzig [3].

A tramp steamer is free to choose its ports of call and the order in which it calls on them. A voyage from port i to port j earns p_{ij} dollars profit, and requires t_{ij} days time (including time for loading cargo at i and unloading at j). What ports should the steamer visit, and in what order, so as to maximize its mean daily profit?

The system can be represented by a labelled directed graph, with nodes identified with ports (states) and arcs with possible voyages (state transitions). Each arc has associated with it a profit (output) $p_{ij} = - c_{ij}$ and a transit time t_{ij}.

A solution to the optimization problem is found by identifying a directed cycle reachable from the initial node for which the ratio of total profit to total travel time is maximized. The trap steamer then sails from its starting point to any port in this cycle, and then continues to sail around this cycle idefinitely.

In order to avoid complications, and to avoid having to examine a multiplicity of cases, we assume that the sum of the transit times around any cycle is strictly positive. I.e., for all C,

$$\sum_{C} t_{ij} > 0. \tag{7.2}$$

Let us guess a minimum value λ for the ratio (7.1). Suppose we give each arc (i,j) in G a new cost $\bar{c}_{ij} = c_{ij} - \lambda\, t_{ij}$. There are three situations that may exist with respect to these modified cost coefficients $\bar{c}_{ij}$:

Case 1. There is a negative cycle in G.

Case 2. There is no negative cycle, and the cost of a minimal-cost cycle is exactly zero.

Case 3. There is no negative cycle, and the cost of a minimal-cost cycle is strictly positive.

Suppose that Case 1 holds. Let C be a negative cycle.

Then

$$\sum_C \bar{c}_{ij} = \sum_C (c_{ij} - \lambda\, t_{ij}) < 0.$$

By assumption (7.2), it follows that

$$\frac{\sum_C c_{ij}}{\sum_C t_{ij}} < \lambda.$$

In other words, the guessed value of λ is too large and C is a cycle for which the cost-to-time ratio is strictly smaller than λ.

By similar analysis, we see that in Case 2 the guessed value of λ is exactly equal to the minimal cost-to-time ratio, and in Case 3 λ is too small.

These observations suggest a search procedure based on the testing of successive trial values of λ. There are two principal types of searches we might conduct: "monotonic" and "binary."

One can organize a monotonic search by choosing an initial value $\lambda^{(0)}$ at least as large as the minimum cost-time ratio, and then, by successive applications of Case 1, obtain $\lambda^{(0)} > \lambda^{(1)} > \lambda^{(2)} \ldots$, until some $\lambda^{(p)}$ is found for which Case 2 holds. This must occur in a finite number of steps, because G contains only a finite number of cycles. Hence there are only a finite number of possible cost-time ratios.

In fact, it can be shown by the analysis in Section 8 that a monotonic search requires only $O(n^3)$ trial values of λ. Since each trial value requires an $O(n^3)$ negative cycle computation, the monotonic search procedure is $O(n^6)$ overall.

Binary search [10] provides a much better bound and is probably much more effective in practice. The binary search proceeds as follows. Suppose we know that the optimum cost-time ratio is contained in the interval (a,b). We first try the trial value $\lambda = \frac{a+b}{2}$. If Case 1 holds, we know that the optimum ratio is contained in the interval $(a, \frac{a+b}{2})$. If Case 3 holds, the optimum ratio is in the interval $(\frac{a+b}{2}, b)$. If Case 2 holds, we have, of course, found the optimum value of the cost-time ratio at the first try.

We continue in this way, halving the remaining interval with each trial value. After k trial values the length of the remaining interval can be no greater than $(b-a)/2^k$. Or, to put it another way, the number of trial values required to reduce the length of the interval to ε is $\log_2 (b-a) - \log_2 \varepsilon$.

We continue the interval-halving procedure until the remaining interval is so small that only one distinct cost-time ratio can be contained within the interval. Then one additional trial value of λ is sufficient to find the cycle with this minimal ratio. (The final value of λ is chosen to be equal to the largest value in the ε-interval; either Case 1 or Case 2 must hold.)

In the next section we indicate how to determine the values a,b, and ε, and estimate the length of the computation.

Remark:

(7.3) It is not difficult to formulate the ratio cycle problem as a "fractional" linear programming problem. I.e. a problem with linear

constraints and an objective function that is the ratio of two linear forms. However, the standard techniques for solving such problems, e.g. [2], are unworkable, because they do not yield integer solutions.

8. Complexity of Cost-to-Time Ratio Algorithm

We suppose that all the parameters c_{ij} and t_{ij} are integers and that $|c_{ij}| \leq \gamma$ and $|t_{ij}| \leq \tau$, for all i,j. Each cycle contains at most n arcs, and the smallest possible value for the sum of the transit times around any cycle is unity. Hence the minimum and maximum attainable cost-to-time ratios are $a = -n\gamma$ and $b = n\gamma$, respectively.

Furthermore, if the cost-to-time ratios for two cycles are unequal, those ratios must differ by at least $\varepsilon = 1/n^2\tau^2$.

It follows that the binary search procedure outlined in Section 7 requires no more trial values of λ than

$$\log_2(b-a) - \log_2 \varepsilon = \log_2(2n\gamma) - \log_2(1/n^2\tau^2) = 1 + 3\log_2 n + \log_2 \gamma + 2\log_2 \tau$$

Thus, the number of negative-cycle problems which must be solved is $O(\log n + \log \gamma + \log \tau)$.

Suppose we are concerned with graphs of various sizes, but with similar cost and time parameter values, if we assume that γ and τ are invariant with n(or even if γ and τ increase as polynomial functions of n) then the number of negative cycle computations is simply proportional to $\log_2 n$ and the overall computation is $O(n^3 \log_2 n)$.

Even if we make no assumptions about the nature of the parameters c_{ij} and t_{ij}, it is clear that the number of computational steps is bounded by a polynomial function in the number of bits required to specify an instance of the problem. In this sense, the minimal cost-to-time ratio problem can be considered to be a well-solved combinatorial optimization problem.

Remark:

(8.1) The cost-to-time ratios for two cycles, if unequal, must differ by at least $1/n^2\tau^2$. However, it is possible that the ratios for two closed paths (not cycles) may differ by less than this amount. Thus, the final interval of length ε may contain a multiplicity of realizable ratios, with respect to closed paths other than cycles. However, the minimum ratio is realized by a cycle, and the negative-cycle computation enables us to find such a cycle.

(8.2) The present author [10] and Shapiro [12] had previously propsed proposed an $O(n^4)$ algorithm for solving the ratio problem, in the special case that $t_{ij} = 1$, for all i,j. This was based on the recursion

$$c_{ij}(1) = c_{ij}, \text{ where } c_{ii} = +\infty$$

$$c_{ij}^{(m+1)} = \min_k \{c_{ik}^{(m)} + c_{kj}\}.$$

One then computes

$$\min_i \min_m \left\{ \frac{c_{ii}^{(m)}}{m} \right\}$$

to obtain the minimum cost-to-time ratio. However, except in very special circumstances it appears that this method is inferior to the binary search method.

9. Side Constraints

It is interesting to consider variations of the cost-to-time ratio problem. For example, suppose there are only m ports, $m < n$, at which the tramp steamer may refuel. A closed path must be found which is optimal, subject to the constraint that the travel time between two successive refulings does not exceed a certain time T. This version of

the ratio problem can be solved as follows.

We first observe that, because of the refueling restrictions, an optimal closed path may not be a cycle. That is, it can pass through a given refueling port no more than once, but it may pass through a given nonrefueling port as many as m times, but not more than once between successive refueling ports. It follows that the minimum and maximum attainable cost-to-time ratios are $a = -m(n-m+1)\gamma$ and $b = m(n-m+1)\gamma$, and that the cost-to-time ratios of two closed paths can be assumed to differ by at least $\varepsilon = 1/m^2(n-m+1)^2\tau^2$. A small bit of algebra shows that, although the number of trial values of λ is greater than in the unconstrained problem, this number is $O(\log n + \log \tau + \log \gamma)$, as before.

For each trial value of λ, we must test for the existence of a negative closed path, subject to refueling constraints. This can be done in two stages. First $c_{ij}(T)$ is computed, for all refueling ports i and j, where

$c_{ij}(t)$ = the cost of a least-costly path from node i to node j, subject to the constraint that the path requires no more than t units of time.

Then a conventional negative cycle computation is carried out for the m-node network of refueling ports, with arc lengths $c_{ij}(T)$. We assert that there is a negative closed path in the original n-node network, subject to refueling constraints, if and only if there is a negative cycle in the m-node network.

One way to carry out the computation of $c_{ij}(T)$ is as follows.

Assume that each t_{ij} is a positive integer. Then we have

$$(9.1)\quad \begin{cases} c_{ij}(t) = +\infty, & \text{if } t < 0, \\ \quad = \min\limits_{k}\{c_{ik}(t-t_{kj}) + c_{kj}\}, & \text{if } t \geq 0, \end{cases}$$

where we assume a self loop with $c_{ii} = 0$, $t_{ii} = 1$, at each node i. Equations (9.1) can be viewed as a generalization of equations (2.2), and require $O(mn^2T)$ computational steps. (Incidentally, note that c_{ij} is actually $\bar{c}_{ij} = c_{ij} - \lambda\, t_{ij}$, where λ is the current trial value.)

An alternative computation, for the case $T >> \tau$, is as follows.

$$(9.2)\quad \begin{cases} c_{ij}(t) = +\infty, & \text{if } t < 0, \\ \quad = \min\limits_{k\neq i,j} \min\limits_{0\leq\delta\leq\tau} \{c_{ik}(\lfloor \frac{t}{2} \rfloor - \delta) + c_{kj}(\lceil \frac{t}{2} \rceil + \delta)\}, & \text{if } 0 \leq t < t_{ij} \\ \quad = \min\{c_{ij}, \min\limits_{k\neq i,j} \min\limits_{0\leq\delta\leq\tau} \{c_{ik}(\lfloor \frac{t}{2} \rfloor - \delta) + c_{kj}(\lceil \frac{t}{2} \rceil + \delta)\}\}, & \text{if } t \geq t_{ij} \end{cases}$$

(The symbols "$\lfloor\ \rfloor$" and "$\lceil\ \rceil$" denote "greatest integer less than or equal to" and "least integer greater than or equal to," respectively.) An analysis of equations (9.2) shows that they imply $O(m\, n^2 \tau^2 \log T)$ computational steps to obtain $c_{ij}(T)$, for all m refueling ports i and j.

If we take the number of trial values of λ to be essentially $O(\log n)$, then the overall computation is either $O(mn^2T \log n)$ or $O(mn^2\tau^2 \log T \log n)$, depending upon whether equations (9.1) or (9.2) are used. If either T or τ is sufficiently small, the computation may not be too forbidding.

Remark:

(9.3) The author is indebted to S. Rinaldi for suggesting improvements in the steamer refueling computation, and for pointing out that the monotone search method for the ratio problem is $O(n^6)$ in complexity.

REFERENCES

[1] R. E. Bellman, "On a routing Problem," Quart. Appl. Math., XVI (1958) 87-90.

[2] A. Charnes and W. W. Cooper, "Programming with Linear Fractional Functions," Naval Research Logistics Quarterly, 9 (1962) 181-186.

[3] G. B. Dantzig, W. Blattner and M. R. Rao, "Finding a Cycle in a Graph with Minimum Cost to Time Ratio with Applications to a Ship Routing Problem," Theory of Graphs, International Symposium, Dunod, Paris, and Gordon and Breach, New York (1966) 77-83.

[4] S. E. Dreyfus, "An Appraisal of Some Shortest Path Algorithms," Operations Research, 17 (May 1969) 395-412.

[5] J. Edmonds, "Path, Trees, and Flowers," Can. J. Math., 17 (1965) 449-467.

[6] M. Florian and P. Robert, "A Direct Search Method to Locate Negative Cycles in a Graph," Management Science, 17 (1971) 307-310.

[7] R. W. Floyd, "Algorithm 97, Shortest Path," Comm. of the ACM, 5 (1962) 345.

[8] L. R. Ford, Jr., "Network Flow Theory," The RAND Corp., P-923 (Aug. 1956).

[9] B. L. Fox, "Finding Minimal Cost-Time Ratio Circuits," Operations Research, 17 (1969) 546-551.

[10] E. L. Lawler, "Optimal Cycles in Doubly Weighted Directed Linear graphs," Theory of Graphs, International Symposium, Dunod, Paris, and Gordon and Breach, New York (1966) 209-213.

[11] E. F. Moore, "The Shortest Path Through a Maze," Proc. Int. Symp. on the Theory of Switching, Part II, April 1957, The Annals of the Computation Laboratory of Harvard University, 30, Harvard University Press (1959).

[12] F. Shapiro, :Shortest Route Methods for Finite State Space Dynamic Programming Problems," SIAM J. Applied Math, 16 (1968) 1232-1250.

[13] I. L. Traiger and A. Gill, "On an Asymptotic Optimization Problem in Finite, Directed, Weighted Graphs," Information and Control, 13 (1968) 527-533.

[14] J. Y. Yen, "An Algorithm for Finding Shortest Routes from All Source Nodes to Given Destination in General Networks," Quart. of Appl. Math., 27, (Jan. 1970) 526-530.

[15] J. Y. Yen, "A Shortest Path Algorithm," Ph.D. dissertation, University of California, Berkeley, California (1970).