MATHEMATICS OF OPERATIONS RESEARCH
Vol. 4, No. 4, November 1979
Printed in U.S.A.

COMBINATORIAL OPTIMIZATION WITH RATIONAL OBJECTIVE FUNCTIONS*

NIMROD MEGIDDO

Tel Aviv University

Let A be the problem of minimizing $c_1x_1 + \ldots + c_nx_n$ subject to certain constraints on $x = (x_1, \ldots, x_n)$, and let B be the problem of minimizing $(a_0 + a_1x_1 + \ldots + a_nx_n)/(b_0 + b_1x_1 + \ldots + b_nx_n)$ subject to the same constraints, assuming the denominator is always positive. It is shown that if A is solvable within $O[p(n)]$ comparisons and $O[q(n)]$ additions, then B is solvable in time $O[p(n)(q(n) + p(n))]$. This applies to most of the "network" algorithms. Consequently, minimum ratio cycles, minimum ratio spanning trees, minimum ratio (simple) paths, maximum ratio weighted matchings, etc., can be computed withing polynomial-time in the number of variables. This improves a result of E. L. Lawler, namely, that a minimum ratio cycle can be computed within a time bound which is polynomial in the number of bits required to specify an instance of the problem. A recent result on minimum ratio spanning trees by R. Chandrasekaran is also improved by the general arguments presented in this paper. Algorithms of time-complexity $O(|E| \cdot |V|^2 \cdot \log|V|)$ for a minimum ratio cycle and $O(|E| \cdot \log^2|V| \cdot \log\log|V|)$ for a minimum ratio spanning tree are developed.

1. Introduction. Numerous combinatorial optimization problems can be formulated as linear minimization problems subject to certain constraints. Let us denote

Problem A:

$$\begin{aligned} &\text{Minimize} && c_1x_1 + \cdots + c_nx_n \\ &\text{s.t.} && x = (x_1, \ldots, x_n) \in D. \end{aligned}$$

As examples we might mention the problems of the shortest (simple) path, the minimum spanning tree, the maximum weighted matching, the minimum cut, the traveling salesman, the Chinese postman, and a variety of scheduling problems.

In view of the examples given above, the following generalization of A is interesting both from the applicative and the theoretical aspects. We denote

Problem B:

$$\begin{aligned} &\text{Minimize} && (a_0 + a_1x_1 + \ldots + a_nx_n)/(b_o + b_1x_1 + \ldots + b_nx_n) \\ &\text{s.t.} && x \in D \end{aligned}$$

(assuming the denominator is always positive).

One example of a practical ratio minimization is that of minimizing cost-to-time ratio. Dantzig, Blattner and Rao [5] and Lawler [8] introduced the problem of the minimum cost-to-time ratio cycle in a graph. This problem applies to finding optimal ship routing.

Our general result in this paper relates the time complexity of problem B to that of problem A. It turns out that whenever A has a good algorithm, then the same is true for B.

An algorithm for the minimum ratio cycle problem is given by Lawler [9, Chapter 3, §13]. The time bound for Lawler's algorithm depends on the numerical values of

*Received November 18, 1977; revised October 20, 1978.
AMS 1970 subject classification. Primary 90C30. Secondary 90C35.
IAOR 1973 subject classification. Main: Fractional programming. Cross reference: Network programming.
OR/MS Index 1978 subject classification. Primary: 623 Programming, fractional. Secondary: 481 Networks/graphs; 652 Programming, equivalence/transformations.
Key words. Fractional programming, computational complexity, polynomial running time; minimum-ratio cycles, spanning trees, paths, weighted matchings.

0364-765X/79/0404/0414$01.25

the parameters, namely, "the number of computational steps is bounded by a polynomial function in the number of bits required to specify an instance of the problem." The theorem proved in the present paper provides improved algorithms, in the sense that the time bound is a polynomial function in the number of the variables and does not depend on the numerical values of the parameters. Karp [7] solves the minimum cycle mean problem (i.e., the special case of the minimum cost-to-time ratio cycle, when all times b_{ij} are equal) in time $O(|E| \cdot |V|)$, where E is the edge set and V is the vertex set of the graph. The theorem proved in this paper implies an $O(|E|^2 \cdot |V|^2)$ for the general minimum ratio cycle. This is reduced further to $O(|E| \cdot |V|^2 \log|V|)$ by considering special features of the negative cycle algorithms.

In a recent paper, Chandrasekaran [3] computes a minimum ratio spanning tree in time $O(|E|^2 \log|V|)$. His argument is specific to the spanning tree problem, even though it can be extended to finding bases in matroids, minimizing ratio functions. Our theorem improves this bound, yielding $O[|E|^2 \cdot (\log\log|V|)^2]$. This is reduced further to $O(|E|\log^2|V|\log\log|V|)$ by considering special features of minimum spanning tree algorithms.

Additional references for ratio minimization can be found in Lawler's book [9, p. 107]. This paper is organized as follows. In §2 we prove a basic theorem, providing a general algorithm for problem B. An example of the general idea is presented in §3. In §4 we discuss ways of accelerating the algorithm, and the case where problem A has a comparisons algorithm is discussed in §5. In §§6 and 7 we provide algorithms for the minimum ratio spanning tree and the minimum ratio cycle, respectively, which are based on the acceleration techniques. In an appendix we provide an algorithm for finding the minimum of n linear functions in time $O(n \log n)$. This algorithm turns out to be useful in accelerating the general algorithm for ratio minimization.

2. The general algorithm. Our main result here is the following theorem.

THEOREM. *If problem* A *is solvable within* $O(p(n))$ *comparisons and* $O(q(n))$ *additions then problem* B *is solvable in time* $O(p(n)(q(n) + p(n)))$.

PROOF. A rather standard trick for solving ratio minimization problems is as follows. Given problem B, pick a real number t and solve problem A with $c_i = a_i - tb_i$ $(i = 1, \ldots, n)$ under the same constraints.

Suppose that v is the optimal value of problem A. If v turns out to be equal to $tb_0 - a_0$ then t is the optimal value of problem B and the optimal solution x of problem A (with respect to t) is also an optimal solution of problem B. On the other hand, if $v < tb_0 - a_0$ then a smaller t should be tested and if $v > tb_0 - a_0$ then a greater t should be tested. This procedure continues until the "correct" value t^* (t^* is characterized by the property that the optimal value $v(t^*)$ of problem A w.r.t. t^* equals $t^*b_0 - a_0$) is found. The key question is how many values of t have to be tested before the "correct" one t^* is found. We shall prove that the number of these tests is not greater than the number of comparisons made by the algorithm which is available for problem A. We refer to this algorithm as the A-algorithm.

Given the data for problem B, consider the different paths, depending on the value of t, that the A-algorithm may follow when solving problem A with $c_i = a_i - tb_i$, $i = 1, \ldots, n$. These paths form a directed tree where branching points correspond to comparisons made by the algorithm.

At the start the data are linear functions (possibly constants) of t, defined over the entire real line. Additions generate some more linear functions of t. Consider the first comparison made by the algorithm. Since the algorithm compares two linear functions of t, the outcome of the comparison may depend on t. However, in any case there will be at most one critical value t_1, say, such that for all $t \leqslant t_1$, one of the

functions compared is greater than or equal to the other, and for $t \geqslant t_1$ the other function is greater than or equal to the first one. Thus, the comparison may partition the real line into two rays of values which are equivalent from the point of view of the outcome of the comparison. The comparison corresponds to a branching point with an outgoing degree of 2, and the two subtrees correspond to the rays $[-\infty, t_1]$ and $[t_1, \infty]$. All the functions previously generated by the algorithm are linear over these rays. Even if a function $g(t) = \text{Min}(f_1(t), f_2(t))$ is generated at the point of comparison, then $g(t)$ is still linear over $[-\infty, t_1]$ and $[t_1, \infty]$. Additions made after the first comparison but prior to the second one will generate functions which are linear over the two rays.

By induction, every branching point corresponds to a comparison between two functions of t which are linear over some interval $[e, f]$ $(-\infty \leqslant e < f \leqslant \infty)$ which is associated with the branching point. Thus, there is at most one critical value t' in the interval. The outgoing degree of the branching point is 2 and the two complementary subintervals $[e, t']$ and $[t', f]$ correspond to the two subtrees rooted at the branching point. All functions generated prior to the next comparison (in either subtree) are linear over the respective subinterval. Furthermore the endpoints are also associated with intervals and the optimal solution given at any endpoint is in fact a linear function of t over the interval associated with the endpoint.

Our observation gives rise to the following algorithm for problem B. Essentially, the algorithm solves problem A parametrically over an interval, which reduces throughout the computation, searching for the "correct" value t^*. At each branching point reached in the tree of the A-algorithm, the corresponding critical value of t is tested by running the A-algorithm with t fixed at the critical value. Then the appropriate branch is selected and the next branching point is considered. At the end, the optimal value of problem A will be given in the form of a linear function $v(t)$ defined over an interval $[e, f]$ which contains t^*. The value t^* is then calculated by solving $v(t^*) = t^* b_0 - a_0$.

Following is a detailed description of an algorithm for problem B.

The description is quite general. However, we will illustrate particular cases in which it will be clear how the general principle is implemented. For the convenience of notation we write $[-\infty, \infty] = \{x : -\infty < x < \infty\}$ and, similarly, if $d \in [-\infty, \infty]$ then $[-\infty, d] = \{x : -\infty < x \leqslant d\}$ and $[d, \infty] = \{x : d \leqslant x < \infty\}$.

The B-algorithm.

0. Initialize with $[e, f] = [-\infty, \infty]$ and define $c_i(t) = a_i - tb_i$, $i = 1, \ldots, n$.

1. Follow the A-algorithm for minimizing $c_1(t)x_1 + \ldots + c_n(t)x_n$ $(x \in D)$, simultaneously for all t in $[e, f]$, from the start or from the recent point of resumption, to the next point of comparison. If there are no more comparisons and the A-algorithm terminates then go to 5; otherwise, let the A-algorithm make a pause at the point of comparison and go to 2.

2. If $g_1(t)$ and $g_2(t)$ are the two linear functions that have to be compared over $[e, f]$, then solve the equation $g_1(t) = g_2(t)$. If there is no unique solution in $[e, f]$ then resume the A-algorithm and go to 1 (the outcome of the comparison is independent of t over $[e, f]$); otherwise go to 3.

3. Let t' denote the unique solution of $g_1(t) = g_2(t)$ over $[e, f]$. Solve the problem: minimize $c_1(t')x_1 + \ldots + c_n(t')x_n$ subject to $x \in D$ by employing the regular A-algorithm (all data are constant). If the optimal value v equals $t'b_0 - a_0$ then terminate (x is an optimal solution for B and $t^* = v$); otherwise set $f = t'$ if $v < t'b_0 - a_0$ and set $e = t'$ if $v > t'b_0 - a_0$.

4. Resume the A-algorithm for the parametric problem over $[e, f]$ and go to 1.

5. The optimal value of problem A is a linear function $v(t)$ over $[e, f]$. The correct value t^* may be found by solving $v(t^*) = t^* b_0 - a_0$. Optimal solutions $x(t)$ are given

in the form of a linear function over $[e, f]$. An optimal solution for problem B will be $x^* = x(t^*)$.

Obviously, the number of critical values of t tested by the A-algorithm is bounded by the total number of comparisons made during one run of the A-algorithm itself. This completes the proof of the theorem.

3. An example: Finding a minimum ratio cycle. The general B-algorithm will be demonstrated by the problem of the minimum cost-to-time ratio in a directed graph.

Let a_{ij} and b_{ij} denote the cost and time, respectively, of traversing an arc (i, j) ($b_{ij} > 0$ for $i \neq j$, $a_{ii} = b_{ii} = 0$, $i = 1, \ldots, n$). We wish to find a directed cycle such that the ratio of the total cost to the total time of traversing the cycle is minimized. The corresponding A-problem is to find the shortest simple cycle in a directed graph with distances c_{ij}. The distances may take negative values.

This problem could be formulated as minimizing $\sum c_{ij}x_{ij}$ where $x = (x_{ij})$ is a zero-one matrix and minimization is taken with respect to the set D of all zero-one matrices, such that the arcs (i, j) for which $x_{ij} = 1$ form a simple directed cycle. Floyd's algorithm for all-pairs shortest paths (see [9]) could be employed as the A-algorithm in this case. Essentially Floyd's algorithm will be used as a negative cycles detector. Negative cycles can be detected in time $O(|E| \cdot |V|)$ (see [6], [7], [9]) and hence an upperbound of $O(|E|^2 \cdot |V|^2)$ for the minimum ratio cycle is implied. However, we will later derive an $O(|E| \cdot |V|^2 \log |V|)$ bound.

Define $u_{ij}^{(m)}(t)$ to be the length (with respect to the distances $c_{kl}(t) = a_{kl} - tb_{kl}$) of a shortest simple path from i to j allowing only nodes in the set $\{1, \ldots, m-1\}$ to serve as intermediate nodes. The corresponding B-algorithm is the following (all bookkeeping with respect to the actual paths and cycles found is omitted).

An $O(n^6)$ algorithm for minimum ratio cycles:

0. Initiate with $[e, f] = [-\infty, \infty]$ and $u_{ij}^{(1)}(t) = a_{ij} - tb_{ij}$ $(1 \leqslant i, j \leqslant n)$. Set $i = j = m = 1$.

1. Solve $u_{ij}^{(m)}(t) = u_{im}^{(m)}(t) + u_{mj}^{(m)}(t)$.

2. If there is a unique solution t' in $[e, f]$ then go to 3; otherwise go to 4.

3. Test the graph with distances $c_{kl}(t')$ for negative cycles. If there is a zero cycle and no negative cycles then terminate (the zero cycle is a minimum ratio cycle); otherwise, let $f = t'$ if there is a negative cycle and let $e = t'$ if all cycles are positive.

4. Set $u_{ij}^{(m+1)}(t) = \text{Min}[u_{ij}^{(m)}(t), u_{im}^{(m)}(t) + u_{mj}^{(m)}(t)]$. (This is *effective* and $u_{ij}^{(m+1)}(t)$ will be *linear* over $[e, f]$.)

5. If $j < n$ set $j = j + 1$ and go to 1; if $j = n$ and $i < n$ set $j = 1$, $i = i + 1$, and go to 1; if $i = j = n$ and $m < n$ set $i = j = 1$, $m = m + 1$, and go to 1; if $i = j = n$ and $m = n$ then go to 6.

6. Find k such that $u_{kk}^{(n+1)}(f) < 0$ and terminate. (The minimum ratio cycle is the shortest (w.r.t. $c_{ij}(f)$) simple cycle which contains k; the "correct" t^* is precisely e.)

The claim in step 6 follows from the fact that for each i, $u_{ii}^{(n+1)}(t)$ is linear over $[e, f]$. Moreover, $u_{ii}^{(n+1)}(e) = 0$, $u_{ii}^{(n+1)}(f) \leqslant 0$ $(i = 1, \ldots, n)$ and for some k $(1 \leqslant k \leqslant n)$ $u_{kk}^{(n+1)}(f) < 0$. Thus, the maximum value of t for which there is no negative cycle is precisely e. The cycle found by the algorithm has zero length (w.r.t. $c_{ij}(t)$) if $t = e$.

Since Floyd's algorithm runs in time $O(n^3)$, it follows that the B-algorithm based on it runs in time $O(n^6)$.

4. Accelerating the B-algorithm. The general idea of the B-algorithm could be summarized as follows. The A-algorithm is employed parametrically over an interval $[e, f]$. Whenever a comparison between two linear functions $c_1(t)$, $c_2(t)$ has to be made, an updating of the interval is considered. The result of the comparison could be a piecewise linear function, $c_3(t) = \text{Min}\{c_1(t), c_2(t)\}$, say, over $[e, f]$. The function $c_3(t)$ could have one breaking-point at most. If there is such a point, then it is tested by

running the regular A-algorithm at that point, and the appropriate subinterval of linearity is selected. Thus, the number of test-runs of the A-algorithm may be as large as the number of comparisons performed by the A-algorithm.

As a matter of fact, in many cases we do not need so many test-runs of the A-algorithm. A test at a breaking point could be postponed for a while in the following manner. The parametric A-algorithm could be decomposed into stages. At the beginning of a stage all the functions are linear over some interval $[e, f]$. During the stage some more linear functions may be generated, and also some piecewise linear functions over $[e, f]$ are generated. The parametric A-algorithm should make a pause when, for the first time, a piecewise linear function, generated previously during the present stage, has to be compared to some other function.

If $e = t_0 < t_1 < \ldots < t_k = f$ are all the breaking-points of piecewise linear functions generated during the present stage, then all of our functions are linear over each subinterval $[t_{i-1}, t_i]$ $(i = 1, \ldots, k)$. It will not be necessary to test all of these points, provided that they are already sorted, since the subinterval $[t_{l-1}, t_l]$ that contains t^* (see §3) can be found by a binary search. This amounts to only $O(\log k)$ test-runs of the A-algorithm. Then the interval is updated: $[e, f] = [t_{l-1}, t_l]$ and the following stage starts. Similarly, when the parametric A-algorithm terminates, the appropriate subinterval $[t_{l-1}, t_l]$ of the final $[e, f]$, which contains t^*, is found by means of a binary search.

The idea of storing critical values is especially simple to implement when the regular A-algorithm itself has the structure of stages or iterations. For example, Floyd's algorithm for all-pairs shortest paths, which we used to detect negative cycles, has an adequate structure of n iterations. The piecewise linear functions generated during one iteration do not have to be compared to any function before the following iteration starts. Thus, an accelerated B-algorithm for a minimum ratio cycle could operate as follows.

Start with $[e, f] = [-\infty, \infty]$ and $u_{ij}^{(1)}(t) = a_{ij} - tb_{ij}$. Assume, by induction, that for some m $(1 \leqslant m < n)$ all the functions $u_{ij}^{(m)}(t)$ $(i = 1, \ldots, n, j = 1, \ldots, n)$ are linear over an interval $[e, f]$ which is known to contain t^* $(t^* < f)$. Compute all solutions of the equations $u_{ij}^{(m)}(t) = u_{im}^{(m)}(t) + u_{mj}^{(m)}(t)$ over $[e, f]$. This requires $O(n^2)$ time. Sort the set of solutions to form a sequence $e = t_0 < t_1 < \ldots < t_k = f$, spending no more than $O(n^2 \log n)$ time. Search for the first l $(1 \leqslant l \leqslant k)$ such that $t^* < t_l$. This requires $O(\log n)$ test-runs of Floyd's algorithm and hence $O(n^3 \log n)$ time. Obviously, the functions $u_{ij}^{(m+1)}(t)$ are linear over $[t_{l-1}, t_l]$, which will serve as the interval $[e, f]$ during the next iteration. Since there are altogether n iterations, the overall time bound is $O(n^4 \log n)$.

We have been using Floyd's algorithm just for demonstrating the general idea. An upper-bound of $O(|E| \cdot |V|^2 \cdot \log |V|)$ for the minimum ratio cycle problem will be derived later.

5. Comparison algorithms. Another way of accelerating the B-algorithm applies to cases where the A-algorithm uses only comparisons of input elements. In such a case, we start from n linear functions, and all of the breaking-points of piecewise linear functions that might be generated by the algorithm are contained in the set of at most $n(n-1)/2$ intersection points of the original functions. Thus, one could compute all of these intersection points beforehand, sort them, and then search for the interval which contains t^* and over which all of the functions will be linear during the entire computation. The computation and sorting of the breaking-points require $O(n^2 \log n)$ time, while the search requires $O(\log n)$ test-runs of the A-algorithm. If $T(n)$ is a time bound for the A-algorithm, then an overall time bound is $O(\text{Max}[T(n), n^2] \cdot \log n)$.

We might note further that the set S of intersection points does not have to be

sorted. If a linear-time median-finding procedure (see [2]) may be used, then the appropriate interval could be found as follows. Find the median t' of S and run the A-algorithm at t'. Then, according to the result of the test-run, drop a half of S and find the median of the remaining half. Repeat this process until only two elements of S are left. This procedure requires no more than $O(\log n)$ test-runs of the A-algorithm. The rest of the computation consists of computing the intersection points and median-findings in sets of sizes smaller than n^2, $\frac{1}{2}n^2$, $\frac{1}{4}n^2$, etc. All these require $O(n^2)$ time, and hence an overall time bound is $O(\text{Max}[n^2, T(n)\log n])$.

As an example for the application of the acceleration technique presented in this section, we might mention the problem of finding a minimum ratio spanning tree [3]. Here we start with the linear functions $a_{vu} - tb_{vu}$ associated with the edges $(v, u) \in E$ of an undirected connected graph. The number of intersection points is bounded by $|E|^2$. The minimum spanning tree problem has many algorithms using only comparisons (see [4], [10]). Since $O(|E|\log|V|)$ is an upper bound for the minimum spanning tree, the time bound for the minimum ratio problem that follows from our technique in the present section is $O(|E|^2)$. The algorithm which supports this time bound coincides essentially with Chandrasekaran's algorithm [3], the only difference being the idea of using linear-time median-finding repeatedly, instead of presorting the set of intersection points. The latter yields an $O(|E|^2\log|V|)$ bound. However, in the following section we beat these bounds down to $O(|E|\log^2|V|\log\log|V|)$.

6. Minimum ratio spanning trees. An efficient algorithm for a minimum ratio spanning tree is obtained if Sollin's algorithm [1, p. 179] is employed as the parametric A-algorithm. Test-runs could be made by any $O(|E|\log\log|V|)$ minimum spanning tree algorithm (see [4], [10]).

Sollin's algorithm consists of at most $O(\log|V|)$ iterations and in each one of them the minimum weight edge incident upon a vertex has to be found for each vertex. When employed parametrically, Sollin's algorithm generates at each vertex v a function which is equal to the minimum of an edge-weight in E_v (E_v is the set of edges incident upon v) linear functions. The amount of time required for computing all intersection points of these linear functions, i.e., $O(|E|^2)$, dominates the relatively low upper-bound of the minimum spanning tree problem. However, we show in the Appendix that the calculation of all breaking-points of the minimum of n linear functions could be carried out in time $O(n \log n)$. This implies that one can find all breaking-points of the functions, representing the minimum weight edges incident upon all vertices, in time $O(|E|\log|V|)$. Thus, a minimum ratio spanning tree can be found as follows.

For every edge $(v, u) \in E$ let $c_{vu}(t) = a_{vu} - tb_{vu}$. At the first stage find all breaking-points of the functions $g_v(t) = \text{Min}\{c_{vu}(t) : (v, u) \in E\}$ $(v \in V)$. Merge all sequences of breaking-points associated with the different vertices, to form a sequence $-\infty = t_0 < t_1 < \ldots < t_k = \infty$. Search for the first l such that the minimum spanning tree with respect to the weights $c_{vu}(t_l)$ has a negative total weight, and set $e = t_{l-1}$, $f = t_l$. For each v identify an edge (v, u) such that $g_v(t) = c_{vu}(t)$ for every t, $t_{l-1} \leqslant t \leqslant t_l$. These edges (except some that might have to be removed in order to prevent possible cycles) will belong to the minimum ratio spanning tree. Identify the groups of vertices that are connected by these edges. By induction, during the computation we have a forest consisting of subtrees $T_1, \ldots, T_m$. If $m = 1$ then the forest is the tree sought. For every component T_i, let $g_i(t)$ $(e \leqslant t \leqslant f)$ denote the minimum weight (with respect to $c_{vu}(t)$) of an edge linking T_i to another component. Find all breaking-points of the $g_i(t)$'s $(i = 1, \ldots, m)$ over $[e, f]$ and merge the m sequences to a sequence $e = t_0 < \ldots < t_k = f$. Search for the appropriate subinterval $[t_{l-1}, t_l]$, identify the corresponding edges, and update $[e, f] = [t_{l-1}, t_l]$. From the set of new edges delete one per each cycle (of new edges) and add the rest to the forest. The number of

components of the new tree will be at most a half times the number of components of the previous one. The process is repeated until the forest has a unique component, in which case it is the minimum spanning tree with respect to the weights $c_{vu}(t)$, for every t in the final interval $[e, f]$. Thus, this tree has zero weight with respect to $c_{vu}(t^*)$, and hence minimizes the given rational objective function.

In procedure MRST, the set T collects the edges of the final spanning tree. The set VS contains the vertex sets corresponding to the connected components of the spanning tree found so far. $E(W)$ is the set of edges linking the vertex set W to some other vertex sets. ES is the collection of the $E(W)$'s. MST(t) is a procedure that returns the weight of a minimum spanning tree with respect to $c_{vu} = a_{vu} - tb_{vu}$.

```
Procedure MRST;
  begin T←∅; VS←∅; ES←∅;
   for each vertex v ∈ V do
    begin
     add the singleton set {v} to VS;
     add the edge set E({v})
     = {all the edges incident upon v} to ES;
    end
   e← −∞; f←∞;
   while |VS| > 1 do
    begin
     for each vertex set W ∈ VS do
      begin
       for each edge (v, v') ∈ E(W) if v' ∈ W
       then delete (v, v') from E(W);
       construct the sequence BP(W) of
       breaking-points of the function
       g_W(t) = Min{a_vu − tb_vu : (v, u) ∈ E(W)}
       in the interval [e, f] (procedure PMIN);
      end
     merge the sequences BP(W) (W ∈ VS) into
     a sequence BP;
     search BP for the first t' such that
     MST(t') < 0;
     f←t'; e←the predecessor of t' in BP;
     T'←∅;
     for each vertex set W ∈ VS do
      begin
       find the edge (v, v') in E(W) such
       that g_W(t) = a_vv' − tb_vv' for every
       t in [e, f];
       add (v, v') to T';
      end
     for each edge (v, v') ∈ T' do
      begin
       if the set W ∈ VS containing v and
       the set W' ∈ VS containing v' are
       distinct then do
        begin
         in VS, replace W and W' by W ∪ W';
         in ES, replace E(W) and E(W') by
```

```
        E(W) ∪ E(W');
        add (v, v') to T;
      end
    end
  end
 output T;
end MRST.
```

The number of iterations is bounded by $O(\log|V|)$. In every iteration the computation of all breaking-points, as well as merging the sequences, does not require more than $O(|E| \cdot \log|V|)$ time. The search involves $O(\log|V|)$ test-runs of a minimum spanning tree procedure, and hence amounts to $O(|E| \cdot \log|V| \cdot \log\log|V|)$ computational steps. Thus, the overall time bound is $O(|E| \cdot \log^2|V| \cdot \log\log|V|)$.

To summarize, procedure MRST calls MST(t) $O(\log^2|V|)$ times. Chandrasekaran's algorithm calls MST(t) $O(\log|V|)$ times, but on the other hand has to spend at least $O(|E|^2)$ time for some other steps.

7. Minimum ratio cycles. Procedure MRC for finding a minimum ratio cycle is based on Karp [7]. The given graph may be assumed to be strongly connected, since otherwise the strongly connected components can be found in time $O(|V| + |E|)$, which is dominated by the overall time bound obtained in any case, and then each component could be handled independently.

A vertex s is chosen arbitrarily, and for every vertex $v \in V$ and k ($i = 0, 1, \ldots, n = |V|$) we denote by $F_k(v; t)$ the minimum weight (with respect to $c_{vu}(t) = a_{vu} - tb_{vu}$) of an edge progression of length k from s to v ($F_k(v; t) = \infty$ if no such edge progression exists). It is proved in [7] that the graph with the weights $c_{vu}(t)$ contains a negative cycle if and only if there is a vertex v such that for every k ($0 \leqslant k \leqslant n-1$) $F_n(v; t) < F_k(v; t)$. If this is the case then the minimum-weight edge progression of length n from s to v contains a negative cycle.

If t is confined to an interval over which $F_k(v; t)$ is a linear function of t, then we represent $F_k(v; t) = G_k(v) + t \cdot H_k(v)$ Also, if the edge progression corresponding to $F_k(v; t)$ is independent of t over the interval under consideration then we denote by $I_k(v)$ the predecessor of v in that edge progression.

The values $F_k(v; t)$ can be computed by $F_k(v; t) = \text{Min}\{F_{k-1}(u; t) + a_{uv} - tb_{uv} : (u, v) \in E\}$ ($k = 1, \ldots, n$), starting from $F_0(s; t) = 0$ and $F_0(v; t) = \infty$ for $v \neq s$. Whenever $F_k(v; t)$ will have to be computed, t will have already been confined to an interval over which each $F_{k-1}(u; t)$ ($(u, v) \in E$) is linear. Then, all breaking-points of the functions $F_k(v; t)$ over that interval of t could be found in time $O(|E|\log|V|)$.

Procedure NCD(t) is a negative cycle detector that returns YES if the graph with the weights $c_{vu}(t)$ contains a negative cycle, and NO otherwise. Procedure MRC computes the functions $F_k(v; t)$ ($k = 0, 1, \ldots, n$, $v \in V$) in an interval $[e, f]$ over which each one of them is linear, and which is also known to contain the critical number $t^* = \text{Max}\{t : \text{NCD}(t) = \text{NO}\}$. Moreover, the edge progression corresponding to $F_k(v; t)$ is independent of t over $[e, f]$. This interval is then narrowed further, using a similar type of technique, to an interval $[e, f]$ which contains t^* ($t^* < f$), and such that $F_n(v; t)$ does not intersect $F_k(v; t)$ in $[e, f]$ (excluding the possibility of coincidence over the entire interval) for every vertex v and k, $k = 0, 1, \ldots, n-1$. Since $t^* < f$, it follows that there is a vertex v^* such that $F_n(v^*; f) < F_k(v^*; f)$ for every k, $k = 0, 1, \ldots, n-1$. This implies that $F_n(v^*; t^*) \leqslant F_k(v^*; t^*)$ for every such k. However, the edge progression corresponding to $F_n(v^*; t^*)$ certainly contains a cycle. It follows that this cycle must be of weight zero with respect to $c_{vu}(t^*)$. Thus, this cycle is a minimum ratio cycle.

```
Procedure MRC;
  begin e ← −∞; f ← ∞; G_0(s) ← 0; H_0(s) ← 0;
   for each vertex v ≠ s do
    begin
     G_0(v) ← ∞; H_0(v) ← 0;
    end
   for k ← 1 until n do
    begin
     for each vertex v ∈ V construct the sequence
     BP(v) of all breaking-points of the function
     g_v(t) = Min{ G_{k−1}(u) + a_uv + t[H_{k−1}(u) − b_uv]:
     (u, v) ∈ E } in [e, f] (procedure PMIN);
     merge the sequences BP(v) (v ∈ V) into a
     sequence BP
     search BP for the first t' such that
     NCD(t') = YES;
     f ← t'; e ← the predecessor of t' in BP;
     for each vertex v ∈ V do
      begin
       identify the vertex u such that
       g_v(t) = G_{k−1}(u) + a_uv + t[H_{k−1}(u) − b_uv]
       for t ∈ [e, f];
       G_k(v) ← G_{k−1}(u) + a_uv;
       H_k(v) ← H_{k−1}(u) − b_uv;
       I_k(v) ← u;
      end
    end
   S ← {e, f}
   for each vertex v ∈ V do
    begin
     for k ← 0 until n − 1 do
      begin
       if H_n(v) ≠ H_k(v) then do
        begin
         t ← [G_k(v) − G_n(v)]/[H_n(v) − H_k(v)];
         if t ∈ [e, f] then add t to S;
        end
      end
    end
   sort S;
   search S for the first t' such that
   NCD(t') = YES; f ← t'; e ← the predecessor of
   t' in S;
   for each vertex v do
    begin
     while x = 0 and k ⩽ n − 1 do
      begin
       if G_n(v) + fH_n(v) ⩾ G_k(v) + fH_k(v)
       then x ← 1;
       k ← k + 1
      end
     if x = 0 then go to 1
```

```
    end
1 S←{v}; x←0; u_n←v; k←n+1;
  while x = 0 do
   begin k←k−1;
    u_{k−1}←I_k(u_k);
    if u_{k−1}∉S then add u_{k−1} to S else x←1;
   end
  identify l (k ⩽ l ⩽ n) such that u_{k−1} = u_l;
  output u_{k−1}, u_k, ..., u_l;
 end MRC.
```

The computation of $G_k(v)$ and $H_k(v)$ for all vertices and some fixed k requires $O(|E|\log|V|)$ time for finding the sequence of breaking-points, and then $O(\log|V|)$ test-runs of a negative cycle detector are required for narrowing the interval. Thus, all the quantities of the form $G_k(v)$ and $H_k(v)$ are computed in time $O(|E|\cdot|V|^2\cdot\log|V|)$. The terminal narrowing of the interval requires the computation of $O(|V|^2)$ values and $O(\log|V|)$ test-runs. Thus, the overall bound is $O(|E|\cdot|V|^2\cdot\log|V|)$.

Appendix: An $O(n\log n)$ algorithm for the minimum of n linear functions. Let $a_1,\ldots,a_n, b_1,\ldots,b_n$ be given real numbers and let $g(t)=\text{Min}\{g(t;i)\equiv a_it+b_i : i=1,\ldots,n\}$. Obviously, $g(t)$ is concave and piecewise linear with at most n linear pieces. In order to describe $g(t)$ completely, it suffices to specify all of its breaking-points $-\infty=t_0<t_1<\ldots<t_{j+1}=\infty$, together with indices $k_1,\ldots,k_{j+1}$, such that $g(t)=g(t;k_i)$ for all t and i $(i=1,\ldots,j+1)$ satisfying $t_{i-1}\leqslant t\leqslant t_i$.

The following algorithm is based on the fact that the slopes of the linear pieces of $g(t)$ form a monotone decreasing sequence. The first step requires sorting the set $\{(a_1,b_1),\ldots,(a_n,b_n)\}$ according to the order $>$, defined by $(a_i,b_i)>(a_j,b_j)$ if and only if either $a_i>a_j$ or $a_i=a_j$ and $b_i<b_j$. Then we may assume that $a_1\geqslant a_2\geqslant\ldots\geqslant a_n$. Denote $g^{(m)}(t)=\text{Min}\{g(t;i):i=1,\ldots,m\}$, $m=1,2,\ldots,n$. Suppose that for some $m<n$ all the breaking-points $-\infty=t_0<t_1<\ldots<t_{j+1}=\infty$ of $g^{(m)}(t)$ are known, together with indices $k_1,\ldots,k_{j+1}$ such that $g^{(m)}(t)=g(t;k_i)$ for $t_{i-1}\leqslant t\leqslant t_i$. The graph of $g_{m+1}(t)$ either intersects the graph of $g^{(m)}(t)$ at most at one point, or coincides with its rightmost linear piece. If $g^{(m)}(t_j)=a_{k_j}t_j+b_{k_j}<g_{m+1}(t_j)$ then the intersection point t^* (if at all) of $g^{(m)}(t)$ and $g_{m+1}(t)$ is the rightmost breaking-point of $g^{(m+1)}(t)$. In fact, $g^{(m+1)}(t)$ coincides with $g^{(m)}(t)$ over $[-\infty,t^*]$, and with $g_{m+1}(t)$ over $[t^*,\infty]$. If $g^{(m)}(t_i)=g_{m+1}(t_j)$ then $g^{(m+1)}(t)$ coincides with $g^{(m)}(t)$ over $[-\infty,t_j]$ and with $g_{m+1}(t)$ over $[t_j,\infty]$. The remaining case is $g^{(m)}(t_j)>g_{m+1}(t_j)$. In this case a binary search over $1,\ldots,j$ finds the first i such that $g^{(m)}(t_i)>g_{m+1}(t_i)$. The intersection point t^* of $g^{(m)}(t)$ and $g_{m+1}(t)$ lies in this case in $[t_{i-1},t_i]$. Again $g^{(m+1)}(t)$ coincides with $g^{(m)}(t)$ over $[-\infty,t^*]$ and with $g_{m+1}(t)$ over $[t^*,\infty]$.

```
Procedure PMIN;
 begin
  sort {(a_1, b_1), ..., (a_n, b_n)} according to the
  order > defined by (a_i b_i) > (a_j, b_j) if
  either a_i > a_j or a_i = a_j and b_i < b_j;
  j←0; k←1;
  for m←2 until n do
   begin
    if j = 0 then do
     begin
      if a_m ≠ a_1 then do
       begin
        j←1; k_2←m; t_1←(b_m − b_1)/(a_1 − a_m);
```

```
        end
       else go to 1;
      end
     else if a_m ≠ a_{k_{j+1}} then do
      begin
       x ← a_{k_j} t_j + b_{k_j}; y ← a_m t_j + b_m;
       if x < y then do
        begin
         t_{j+1} ← (b_m − b_{k_{j+1}})/(a_{k_{j+1}} − a_m);
         k_{j+2} ← m; j ← j + 1;
        end
       else if x > y then do
        begin
         search for the first i such that
         a_{k_i} t_i + b_{k_i} > a_m t_i + b_m;
         j ← i; k_{j+1} ← m
        end
      end
1   end
  end PMIN.
```

Procedure PMIN runs in $O(n \log n)$ time. We note that the data structure for PMIN could simply be an array with a pointer, since all deletions are of terminal portions of a set and all insertions are made at the end of the set.

We also note that any algorithm for finding the minimum of n linear functions, using only comparisons, requires $O(n \log n)$ time. This is implied by the fact that the sorting problem is reducible to minimum finding problem in the following manner. Suppose that a set $S = \{a_1, \ldots, a_n\}$ of real numbers has to be sorted, and assume that all the elements of S are distinct. Let $g_i(t) = a_i t + a_i^2$, $i = 1, \ldots, n$. If $i \neq j$ then $g_i(t)$ intersects $g_j(t)$ at the point $-a_i - a_j$. This implies that $g(t) \equiv \text{Min}\{g_i(t) : i = 1, \ldots, n\}$ consists of precisely n linear pieces whose slopes constitute a monotone decreasing sequence. This sequence is in fact the sorted set S.

References

[1] Berge, C. and Ghouila-Houri, A. (1965). *Programming, Games and Transportation Networks*. Wiley, New York.

[2] Blum, M., Floyd, R. W., Pratt, R., Rivest, R. L. and Tarjan, R. E. (1972). Time Bounds for Selection. *J. Comput. System Sci.* 7 448–461.

[3] Chandrasekaran, R. (1977). Minimum Ratio Spanning Trees. *Networks*. 7 335–342.

[4] Cheriton, D. and Tarjan, R. E. (1976). Finding Minimum Spanning Trees. *SIAM J. Comput.* 5 724–742.

[5] Dantzig, G. B., Blattner, W. and Rao, M. R. (1967). Finding a Cycle in a Graph with Minimum Cost to Time Ratio with Application to a Ship Routing Problem. In *Theory of Graphs*, P. Rosenstiehl, ed. Dunod, Paris, and Gordon and Breach, New York. 77–84.

[6] Johnson, D. B. (1973). Algorithms for Shorest Paths. Ph.D. dissertation, Cornell University, Ithaca, New York.

[7] Karp, R. M. (June 1977). A Characterization of the Minimum Cycle Mean in a Digraph. Memorandum No. UCB/ERL M77/47, Electronic Research Laboratory, College of Engineering, University of California at Berkeley.

[8] Lawler, E. L. (1967). Optimal Cycles in Doubly Weighted Linear Graphs. In *Theory of Graphs*, P. Rosenstiehl, ed. Dunod, Paris, and Gordon and Breach, New York. 209–214.

[9] ———. (1976). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

[10] Yao, A. C. (1975). An $O(|E| \log \log |V|)$ Algorithm for Finding Minimum Spanning Trees. *Information Processing Lett.* 4 21–23.

TEL AVIV UNIVERSITY, TEL AVIV, ISRAEL