MATHEMATICS OF OPERATIONS RESEARCH
Vol. 17, No. 1, February 1992
Printed in U.S.A.

APPROXIMATION SCHEMES FOR THE RESTRICTED SHORTEST PATH PROBLEM*

REFAEL HASSIN

This note contains two fully polynomial approximation schemes for the shortest path problem with an additional constraint. The main difficulty in constructing such algorithms arises since no trivial lower and upper bounds on the solution value, whose ratio is polynomially bounded, are known. In spite of this difficulty, one of the algorithms presented here is strongly polynomial. Applications to other problems are also discussed.

1. Introduction. This note describes fully polynomial approximation schemes for the restricted shortest path problem. In this problem a directed graph $G = (V, E)$ with a vertex set $V = \{1, \ldots, n\}$ and an edge set E is given. Each edge $(i, j) \in E$ has a *length* c_{ij} and a *transition time* t_{ij}. These numbers are assumed here to be integers and, in order to simplify the presentation, positive. The length $c(S)$ and transition time $t(S)$ of a set of edges S is defined as the sum of the lengths and transition times, respectively, of the edges in this set. A directed path from vertex i to vertex j (an *i-j path*) is called a *t-path* if its transition time is less than or equal to t. The restricted shortest path problem is to compute, for a given value T, a shortest $1 - n$ T-path.

Fully polynomial approximation schemes (FPAS) are known already for many NP-complete problems that can be solved by pseudopolynomial algorithms. These algorithms usually apply rounding and scaling to the data (for definitions and a description of the basic ideas, see Sahni 1977 and Garey and Johnson 1979). In order to select an appropriate scaling factor, and guarantee that the resulting approximation will be polynomial, these algorithms require knowledge of upper and lower bounds on the optimal solution whose ratio is bounded by a polynomial function of the size of the input.

For example, Ibarra and Kim (1975) obtained a FPAS for the 0-1 knapsack problem, and Lawler (1979) improved their algorithm. The algorithm uses the fact that a variant of the greedy algorithm gives a lower bound that is at least half the value of an optimal solution.

A main difficulty in the development of a FPAS for the restricted shortest path problem is that no trivial upper and lower bounds are known such that their ratio is bounded by a polynomial function of the input. In spite of this difficulty, Warburton (1987) developed a FPAS for the problem, and as a matter of fact also for the broader class of problems of identifying the Pareto optimal paths of multiple objective problems. The ideas presented here can be combined with his results and extended to yield a strongly polynomial approximation procedure for the approximation of Pareto optima in multiple objective path problems. When applied to the restricted shortest path problem in acyclic graphs, Warburton's algorithm requires $O(n^3\epsilon^{-1} \log n \log B)$ time. Here B is an upper bound on the optimal solution value, equal to $n - 1$ times the maximum edge-length, and the algorithm produces a path

*Received July 11, 1988; revised April 6, 1990.
AMS 1980 subject classification. Primary: 05C38.
IAOR 1973 subject classification. Main: Graphs.
OR/MS Index 1978 subject classification. Primary: 489 Networks (graphs) theory.
Key words. Strongly polynomial approximation scheme, restricted shortest path.

0364-765X/92/1701/0036/$01.25

with length at most $1 + \epsilon$ times the length of an optimal path. The computation time is measured here by the number of real additions, multiplications, divisions, and comparisons.

Following Warburton (1987), we assume that the underlying graph is acyclic. This assumption simplifies the presentation while the extension of the algorithms to general graphs is straightforward.

In the next section we outline two exact pseudopolynomial algorithms. In §3 and §4 we present an $O(\log\log B(|E|(n/\epsilon) + \log\log B))$ ϵ-approximation algorithm (B is an upper bound on the solution). The principles of this algorithm are close to those of Warburton's algorithm, using the basic technique of rounding and scaling. The above complexity, while polynomial, depends on the size of the input numbers. In §5 and §6 we present an alternative FPAS whose complexity depends only on the number of input variables and $1/\epsilon$. The number of arithmetic operations required is $O(|E|(n^2/\epsilon)\log(n/\epsilon))$. We conclude the paper with an application to a scheduling problem in §7.

2. Exact algorithms. Some authors constructed practical algorithms for the restricted shortest path problem (Joksch 1966, Lawler 1976, Handler and Zang 1980, Aneja, Aggarwal and Nair 1983, Henig 1985). None of these algorithms are polynomial, and as a matter of fact the problem is NP-hard (cf., Garey and Johnson 1979). It is well known that the problem can be solved by pseudopolynomial algorithms, and thus if the input numbers are uniformly bounded it is polynomially solvable. In fact as we show in this section, it suffices that the edge lengths **or** transit times are bounded. We now outline the pseudopolynomial procedures. We will assume that the vertices are numbered in a way such that $(i, j) \in E$ implies $i < j$. This numbering can be done in $O(|E|)$ time. We denote by OPT the length of a shortest $1 - n$ T-path. We first describe the standard dynamic programming procedure (e.g., Joksch 1966, Lawler 1976):

why?

Algorithm A. Denote by $f_j(t)$ the length of a shortest $1 - j$ t-path. Then:

$$f_1(t) = 0, \qquad t = 0, \ldots, T,$$

$$f_j(0) = \infty, \qquad j = 2, \ldots, n,$$

$$f_j(t) = \min\left\{f_j(t-1), \min_{k|t_{kj} \leq t}\left\{f_k(t - t_{kj}) + c_{kj}\right\}\right\}, \qquad j = 2, \ldots, n,\ t = 1, \ldots, T.$$

Warum nich O(|V| T)?

The algorithm computes a shortest $1 - n$ T-path of length $OPT = f_n(T)$ in $O(|E|T)$ time. Joksch (1966) presented an improvement to this algorithm, noting that $f_j(t)$ is a step function and thus it suffices to locate its steps. However, the worst case complexity of this improvement is not better than that of the basic application. For our purposes another recursive algorithm is more suitable:

Algorithm B. Denote by $g_j(c)$ the time of a quickest $1 - j$ path whose length is at most c. Then:

$$g_1(c) = 0, \qquad c = 0, \ldots, OPT,$$

$$g_j(0) = \infty, \qquad j = 2, \ldots, n,$$

$$g_j(c) = \min\left\{g_j(c-1), \min_{k|c_{kj} \leq c}\left\{g_k(c - c_{kj}) + t_{kj}\right\}\right\},$$

$$j = 2, \ldots, n,\ c = 1, \ldots, OPT.$$

Note that OPT is not known a priori, but it satisfies $OPT = \min\{c | g_n(c) \leqslant T\}$. Thus $g_j(c)$ is computed first for $c = 1$ and $j = 2, \ldots, n$, then for $c = 2$ and $j = 2, \ldots, n$ and so on, until the first value of c for which $g_n(c) \leqslant T$. Then OPT is set to this value of c. The complexity of this algorithm is $O(|E|OPT)$.

Warum nicht O(IVI OPT)?

3. A testing procedure. Let V be a given value, and suppose we want to test whether $OPT \geqslant V$ or not. A polynomial procedure that answers this query can be extended to a polynomial algorithm for finding the exact value of OPT. One simply performs binary search on $\{0, \ldots, B\}$, where B is an upper bound on OPT such as the one mentioned in the introduction. The exact value of OPT is then found after $O(\log B)$ applications of the polynomial procedure. As the problem is NP-hard it comes that we will have to be satisfied with a weaker test.

Let ϵ be fixed, $0 < \epsilon < 1$. We now show how to construct a polynomial time ϵ-approximation test. This test has the following properties: If it outputs a positive answer then definitely $OPT \geqslant V$. If it outputs a negative answer, then all we know is that $OPT < V(1 + \epsilon)$.

The test rounds the edge-lengths c_{ij}, replacing them by

$$\left\lfloor \frac{c_{ij}}{V\epsilon/(n-1)} \right\rfloor \frac{V\epsilon}{n-1}.$$

Falls c_ij = k div + 0.9999 div

This decreases each edge-length by at most $V\epsilon/(n-1)$, and each path-length by at most $V\epsilon$. Now the problem can be solved by applying Algorithm B to a network with the scaled edge-lengths $\lfloor c_{ij}/(V\epsilon/(n-1)) \rfloor$. The values $g_j(c)$, $j = 2, \ldots, n$, are first computed for $c = 1$, then for $c = 2, 3, \ldots$ until either $g_n(c) \leqslant T$ for some $c = \hat{c} < (n-1)/\epsilon$, or $c \geqslant (n-1)/\epsilon$.

In the first case, a T-path of length at most

$$\frac{V\epsilon}{n-1}\hat{c} + V\epsilon < V(1+\epsilon)$$

Wtf, why? Soll das die Decreases von oben ausgleichen? Einmal für jede Kante + Ve für den Gesamten Pfad? Unklar, die Ungelichung gilt aber auch ohne eins der beiden

has been found. In the second case, every T-path has $c' \geqslant (n-1)/\epsilon$ or $c \geqslant V$, so that $OPT \geqslant V$. Thus the test performs as required. The test is summarized as following:

Procedure TEST(V).

1. Set $c \leftarrow 0$.
   For all $(i, j) \in E$:
   If $c_{ij} > V$, set $E \leftarrow E \setminus \{(i, j)\}$.
   Else, set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$.
2. If $c \geqslant (n-1)/\epsilon$ output YES.
   Else use Algorithm B to compute $g_j(c)$ for $j = 2, \ldots, n$.
   If $g_n(c) \leqslant T$ output NO.
   Else set $c \leftarrow c + 1$ and repeat Step 2.

Wo kommen diese (n - 1)/e her?
Kanten mit Kosten > V werden entfernt (s.u.). Hat eine Kante Kosten V, hat sie nach der Skalierung Kosten V (n-1) / Ve = (n-1)/e

Taking the integer part of a nonnegative number A which is known to be bounded from above by an integer U can be done in $O(\log U)$ time using binary search on $\{0, \ldots, U\}$. Step 1 of $TEST(V)$ requires $O(E)$ such operations. Since we scale only costs c_{ij} that are less than or equal to V, then all of the numbers to which we apply floor operations are smaller than $(n-1)/\epsilon$. Thus Step 1 requires $O(|E|\log(n/\epsilon))$ time. Step 2 uses $O(n/\epsilon)$ iterations of Algorithm B, so that its time is $O(|E|n/\epsilon)$. The latter is therefore also the complexity of the whole $TEST$ procedure.

4. First FPAS: rounding and scaling. To approximate OPT we first need easily computable upper and lower bounds. An upper bound, denoted UB, can be set to the sum of the $n-1$ longest edge-lengths, or to the length of the quickest $1-n$ path. (If this path is not a T-path then clearly no feasible solution exists.) A lower bound, denoted by LB, can be set to the length of a shortest (unrestricted) $1-n$ path, or simply to 1. Another possibility is to relax the integer programming formulation of the problem by replacing the binary constraints by nonnegativity constraints. The resulting linear program can be solved to obtain a lower bound on OPT. However, this may involve relatively high time complexity and does not supply a feasible solution associated with the bound.

If $UB \leqslant (1+\epsilon)LB$ then UB is an ϵ-approximation to OPT. Suppose now that $UB > (1+\epsilon)LB$. Let V be a given value $LB < V < UB(1+\epsilon)^{-1}$. TEST($V$) can be applied now to improve the bounds on OPT. Specifically either LB is increased to V or UB is decreased to $V(1+\epsilon)$. By performing a sequence of tests the ratio UB/LB can be reduced. Once this ratio reduces below some predetermined constant, say 2, then an ϵ-approximation can be obtained by applying Algorithm B to the scaled costs $\lfloor c_{ij}/(LB\epsilon/(n-1))\rfloor$. The error introduced is at most $\epsilon LB < \epsilon OPT$ and the time complexity is $O(|E|n/\epsilon)$, as for a single application of TEST(V).

Reduction of the ratio UB/LB is best achieved by performing binary search on the interval (LB, UB) in a logarithmic scale. After each test we modify the bounds. To guarantee fast reduction in the bounds' ratio we execute the first test in the point x such that $UB/x = x/LB$, that is $x = (UB \cdot LB)^{1/2}$. The number of tests required to reduce the ratio below 2 is therefore $O(\log\log(UB/LB))$:

Rounding Algorithm.

0. Set LB and UB to their initial values.
   (For example, set $LB = 1$, UB = sum of $(n-1)$ longest edge-lengths.)
1. If $UB \leqslant 2LB$ go to Step 2.
   Let $V = (LB \cdot UB)^{1/2}$.
   If TEST(V) = YES, set $LB = V$.
   If TEST(V) = NO, set $UB = V(1+\epsilon)$.
   Repeat Step 1.
2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB\rfloor$.
   Apply Algorithm B to compute an optimal T-path.
   Output this path.

Computing $(LB \cdot UB)^{1/2}$ to high precision may be time consuming. Instead, we can find in $O(\log\log(LB/UB))$ time an integer V' satisfying $(UB/LB)^{1/4} < V' \leqslant (UB/LB)^{1/2}$. This is done by generating the sequence $a_i = 2^{(2^i)}$, $i = 1, 2, \ldots$, till for the first time we have $a_i > UB/LB$. Then $V' = a_{i-2}$ satisfies the above inequalities. Now let the test point be $V = V' \cdot LB$. Note that the ratio of the new bounds, after applying TEST(V), is at most $(UB/LB)^{3/4}$. (This is true if $(1+\epsilon)^4 < 2$. Else, the first line of Step 1 of the procedure can be modified: If $UB < (1+\epsilon)^4 LB$ go to Step 2.) The total number of tests may increase as a result of this approximation, but it is still $O(\log\log(UB/LB))$. Each test requires $O(|E|n/\epsilon)$ and together with the time needed to compute the test point we obtain a total of $O(\log\log(UB/LB)(|E|(n/\epsilon) + \log\log(UB/LB)))$.

REMARK. The idea of reducing a ratio by performing binary search in a logarithmic scale is not limited to the present model. We now mention two examples. Hochbaum and Shmoys (1988) developed a polynomial approximation scheme for the minimum makespan problem on uniform parallel machines. Lenstra, Shmoys, and Tardos (1987) achieved such a result for scheduling a bounded number of unrelated

parallel machines (improving the space requirement of a polynomial approximation scheme developed by Horowitz and Sahni 1976). To obtain these results a procedure similar to the one developed by Warburton (1987) was used (cf. Theorem 1 of Hochbaum and Shmoys 1988 and Lemma 1 of Lenstra, Shmoys, and Tardos 1987). Replacing the binary search by a search procedure as in Step 1 of Rounding Algorithm will reduce the number of tests required by the algorithms from $O(\log(UB/LB))$ to $O(\log\log(UB/LB))$.

5. Partitioning. Suppose we are given a set $Q = \{p_1, \ldots, p_m\}$ and a number X such that $p_1, \ldots, p_m, X > 0$. We are interested in partitioning Q into subsets $R_1, \ldots, R_{k+1}$ such that $p_i \in R_j$ if and only if $X(j-1)/k < p_i \leqslant Xj/k$ for $j = 1, \ldots, k$, and $p_i \in R_{k+1}$ if and only if $p_i > X$. This task is easily performed in $O(m \log k)$ operations by letting $p_i \in R_j$ for $j = \min\{k+1, \lceil kp_i/X \rceil\}$. Suppose now that X is of unknown value, but a test is available to check for any number V whether $X \geqslant V$ or $X < V$. The following procedure partitions Q in $O(\log mk)$ tests, and $O(mk \log mk)$ additional operations.

Procedure PARTITION(Q, X, k). Let $P = \{p_{ij} | p_{ij} = kp_i/j,\ j = 1, \ldots, k,\ i = 1, \ldots, m\}$.

1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
   2. Perform binary search on π to locate the interval $[\pi_g, \pi_{g+1})$ containing X.
3. For $i = 1, \ldots, m$: Insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geqslant p_{ij}$, $j = 1, \ldots, k$.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$.

To explain Step 3 we note that if $p_{i,(j-1)} > \pi_g \geqslant p_{ij}$ then also $p_{i,(j-1)} \geqslant \pi_{g+1} > X \geqslant p_{ij}$ implying $X(j-1)/k < p_i \leqslant Xj/k$. If $\pi_g < p_{ik}$ then also $\pi_{g+1} \leqslant p_{ik} = p_i$ implying $X < p_i$.

A reduction in the number of additional operations, while maintaining the number of tests, is possible if we exploit the special structure of P. This is done by applying techniques for searching over matrices with sorted rows (see, for example, Frederickson and Johnson 1982). Basically, one finds in $O(m)$ time an element of P which is known to be greater than or equal to a quarter of the elements and also smaller than or equal to a quarter of them (for the first iteration this is just the median of the $(k/2)$th column). Then this element is tested and at least a quarter of the elements can be discarded. After $O(\log(mk/m))$ iterations only $O(m)$ elements are left. In additional $O(\log m)$ tests and $O(m)$ other operations the search is completed. Note that $O(m \log k)$ is also the complexity of Step 3. Thus, in total, PARTITION(Q, X, k) requires $O(\log mk)$ tests and $O(m \log k)$ other operations.

Suppose now that instead of an exact test for the query "$X \geqslant V$?" we have an ϵ-approximate test such as TEST(V). In this case we know that X is in the extended interval $[\pi_g, \pi_{g+1}(1+\epsilon))$. The sets generated by PARTITION$(Q, X, \lceil k(1+\epsilon) \rceil)$ have the property that $p_i \in R_j$ implies $X(j-1)/k < p_i \leqslant Xj/(k(1+\epsilon))$. Also, $p_i \in R_{k(1+\epsilon)+1}$ implies that $p_i > X$.

6. Second FPAS: interval partitioning. While the rounding algorithm is polynomial, its time complexity depends on the size of the edge-lengths, via the initial upper bound, UB. In this section we introduce a second FPAS whose complexity depends only on n and $1/\epsilon$. The underlying technique is related to the *interval partitioning* algorithm of Sahni (cf. Sahni 1977).

The algorithm consists of $n-1$ iterations, where in iteration i a set $S^{(i)}$ of $1-i$ paths is generated. All $1-j$ paths generated later for $j>i$, that use vertex i, will be restricted to have as their initial $1-i$ segment only members of $S^{(i)}$. The main achievement of the procedure will be in restricting the size of the sets $S^{(i)}$ so that the algorithm's time complexity will be kept small, while maintaining an error that is bounded by ϵOPT.

Partitioning Algorithm.

0. Let $S^{(1)}$ contain the path consisting of vertex 1.
Set $i=2$.

1. Set $Q^{(i)}=\{$*the lengths of* $1-i$ *paths constructed by appending edge* (j,i) *to a* $1-j$ *path of* $S^{(j)}$ *for some* $j<i\}$.
If $i=n$ set $c(P^*)=\min\{c(P)|P\in Q^*\}$ where $Q^*=\{P\in Q^{(n)}|t(P)\leqslant T\}$, and STOP.
(If $Q^*=\varnothing$ then no feasible solution exists. Else, P^* is a $1-n$ T-path with length $c(P^*)\leqslant(1+\epsilon)OPT$.)

2. Apply PARTITION$(Q^{(i)}, OPT, \lceil(n-1)(1+\epsilon)/\epsilon\rceil)$ using the TEST procedure for the binary search on π in Step 2.
Form $S^{(i)}$ by picking up a quickest subpath from each set R_j, $j=1,\ldots,\lceil(n-1)(1+\epsilon)/\epsilon\rceil$.
Set $i\leftarrow i+1$ and return to Step 1.

The set $S^{(i)}$ is formed in Step 2 of the algorithm from $Q^{(i)}$ by deleting paths that are known to be longer than $OPT(1+\epsilon)$, and other paths that are known to be slower than a path in $S^{(i)}$ whose length differs from the deleted path by at most $OPT\epsilon/(n-1)(1+\epsilon)$. Therefore the relative error accumulated during the execution of the algorithm increases by at most $\epsilon/(n-1)(1+\epsilon)$ per vertex and $\epsilon/(1+\epsilon)<\epsilon$ in total. The size of the set $S^{(i)}$ is $\lceil(n-1)(1+\epsilon)/\epsilon\rceil=O(n/\epsilon)$. The size of $Q^{(i)}$ is at most $\sum_{j<i}|S^{(j)}|=O(n^2/\epsilon)$. Thus each application of PARTITION requires $O((n^2/\epsilon)\log(n/\epsilon))$ elementary operations and

$$O\left(\log\frac{n(1+\epsilon)}{\epsilon}\frac{n^2}{\epsilon}\right)=O(\log(n/\epsilon))$$

tests. Each test is $O(|E|(n/\epsilon))$ so testing amounts in total to $O(|E|(n/\epsilon)\log(n/\epsilon))$. Forming the sets $S^{(i)}$ takes $O(|Q^{(i)}|)=O(n^2/\epsilon)$. Thus for a fixed i Step 2 requires $O(|E|(n/\epsilon)\log(n/\epsilon))$ and altogether the algorithm is $O(|E|(n^2/\epsilon)\log(n/\epsilon))$.

7. Concluding remarks. The techniques presented in this paper can also be applied to other minimization problems for which a dynamic programming algorithm is available, but good lower and upper bounds are not known.

As an example consider the following scheduling problem solved in Sahni (1977): Each of n jobs needs processing on a given machine. Job i has a processing time requirement t_i, a deadline d_i, and a profit p_i which is earned only if the processing of the job is completed by d_i. The objective is to find a schedule that maximizes the profit. An exact algorithm is presented in Sahni (1977). It solves the problem by first ordering the jobs by their deadlines and then constructing partial solutions by determining at the ith stage whether Job i will be processed in time or not. The algorithm's complexity is $O(\min\{2^n, n\sum_{j=1}^n p_j, n\sum_{j=1}^n t_j, \sum_{j=1}^n d_j\})$. Sahni applies interval partitioning and scaling to yield ϵ-approximation schemes of time $O(n^2/\epsilon)$. The idea of the partitioning heuristic is to use the highest value obtained in the ith stage, say B_i, as a lower bound on the solution value OPT. Then the interval $[1, B_i]$ is partitioned to $O(n/\epsilon)$ subintervals of size at most $\epsilon B_i/n$, and in each subinterval all

but one solution are discarded. Since $B_i \leqslant OPT$ the total error accumulated in the n stages is at most ϵOPT.

Suppose now that the problem is posed in a slightly different way, where the only change is that the objective is to minimize the lost profit due to jobs that are not completed in time. In other words, p_i can be viewed as a fixed penalty (not a function of the size of the delay) associated Job i if it is not completed by its deadline. Clearly the two objectives are identical as long as exact computation is concerned and, as a matter of fact, their sum is constant for all possible solutions. However, an ϵ-approximation scheme for the first may give large errors relative to the latter. This is so since B_i may be very large relative to OPT and the subintervals may be too large. Our approach yields an FPAS as follows: We use the approximation scheme of Sahni as our TEST(V) where V, the trial value, is used instead of B_i to construct the subintervals. One possibility is to apply this test $O(\log\log\sum_i d_i)$ times, as described above in §4. The other is to use the partitioning algorithm of §6. Here both m and k are $O(n/\epsilon)$ since $|Q^{(i)}| \leqslant 2|S^{(i-1)}|$. Therefore, the overall complexity of performing $O\log(mk)$ tests and $O(m \log k)$ other operations in each stage amounts to $O((n^3/\epsilon) \log(n/\epsilon))$.

References

Aneja, Y. P., Aggarwal, V., and Nair, K. P. K. (1983). Shortest Chain Subject to Side Constraints. *Networks* **13** 295–302.

Frederickson, G. N. and Johnson, D. B. (1982). The Complexity of Selection and Ranking in $X + Y$ and Matrices with Sorted Columns. *J. Comput. System Sci.* **24** 197–208.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP Completeness*. W. H. Freeman, San Francisco.

Handler, G. and Zang, I. (1980). A Dual Algorithm for the Constrained Shortest Path Problem. *Networks* **10** 293–310.

Henig, M. (1985). The Shortest Path Problem with Two Objective Functions. *European J. Oper. Res.* **25** 281–291.

Hochbaum, D. and Shmoys, D. (1988). A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approach. *SIAM J. Comput.* **17** 539–551.

Horowitz, E. and Sahni, S. (1976). Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J. Assoc. Comput. Mach.* **23** 317–327.

Ibarra, O. and Kim, C. (1975). Fast Approximation Algorithms for the Knapsack and Sum of Subsets Problems. *J. Assoc. Comput. Mach.* **22** 463–468.

Joksch, H. C. (1966). The Shortest Route Problem with Constraints. *J. Math. Anal. Appl.* **14** 191–197.

Lawler, E. L. (1976). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

_____ (1979). Fast Approximation Algorithms for Knapsack Problems. *Math. Oper. Res.* **4** 339–356.

Lenstra, J. K., Shmoys, D. B. and Tardos, E. (1987). Approximate Algorithms for Scheduling Unrelated Parallel Machines. *Proc. 28th Annual Sympos. Foundations of Computer Science*, 217–224.

Sahni, S. (1977). General Techniques for Combinatorial Approximations. *Oper. Res.* **25** 920–936.

Warburton, A. (1987). Approximation of Pareto Optima in Multiple-Objective, Shortest Path Problems. *Oper. Res.* **35** 70–79.

STATISTICS DEPARTMENT, TEL-AVIV UNIVERSITY, TEL-AVIV 69978, ISRAEL