Transpn Res.-B Vol. 16B, No. 1, pp. 45–55, 1982
Printed in Great Britain.

0191-2615/82/010045-11$03.00/0

APPROXIMATE ALGORITHMS FOR THE DISCRETE NETWORK DESIGN PROBLEM†

HOSSAIN POORZAHEDY
Isfahan University of Technology, Isfahan, Iran

and

MARK A. TURNQUIST
School of Civil and Environmental Engineering, Cornell University, Ithaca, NY 14853, U.S.A.

(*Received* 9 *August* 1980; *in revised form* 21 *January* 1981)

Abstract—The discrete network design problem is one of finding a set of feasible actions (projects) from among a collection of possible actions, that when implemented, optimizes some objective function(s). This is a combinatorial optimization problem that is very expensive to solve exactly. This paper proposes two algorithms for obtaining approximate solutions to the discrete network design problem with much less computational effeort. The computational savings are achieved by approximating the original problem with a new formulation which is easier to solve. The first algorithm proposed solves this approximate problem exactly, while the second is even more efficient, but provides only a near-optimal solution to the approximate problem. Experience with test problems indicates that these approximations can reduce the computational effort by a factor of 3–5, with little loss in solution accuracy.

1. INTRODUCTION

Investment planning in transportation networks is a complicated problem. In general, the problem is to find a set of feasible actions (investments) from among a collection of possible actions, that when implemented, optimizes some objective function(s). The feasibility of a set of actions is determined by resource, physical and environmental constraints.

There is a substantial history of attempts to address this problem using optimization methods. However, these efforts have generally proved to be either too computationally demanding to be useful for "real" problems, or too simplified to produce useful results. They also have typically provided little in the way of sensitivity information, particularly with respect to changes in the allowable total level of investment. Such information is of vital importance to transportation planners, for budgets are seldom rigidly fixed.

The objective of this paper is to suggest alternative algorithms for obtaining an approximately optimal solution to the discrete network design problem. These algorithms are much faster than previous methods, and provide budget sensitivity information very readily. The improvements are achieved by solving an approximation to the original problem. Thus, an optimal solution to the original problem cannot be guaranteed, but in cases tested to date, the approximation appears quite reasonable.

We proceed as follows. Section 2 presents the formal mathematical formulation of the network design problem, and discusses previous attempts at solution. Section 3 describes an approximation to this problem, and two algorithms designed to solve it. Computational results on a particular network are presented in Section 4, and Section 5 contains conclusions.

2. PROBLEM FORMULATION AND PRIOR RESEARCH

Consider a network $\eta(V, A)$, where V is the set of vertices with $|V| = N$; and $A = A_1 \cup A_2 \cup A_3$ is the set of arcs, with $A_1 = \{i: i = 1, 2, \ldots, n\}$ the set of existing arcs which will not be modified, $A_2 = \{i: i = n+1, \ldots, n+m\}$ the set of existing arcs for which there are candidate projects, and $A_3 = \{i: i = n+m+1, \ldots, n+2m\}$ the set of *new* arcs. The arcs in A_2 and A_3 are numbered such that if $i \in A_2$ is chosen as an arc whose project is to be undertaken, arc $i+m \in A_3$ will replace i, otherwise $i+m \in A_3$ will be discarded. In other words, i and $i+m$ are the old and new versions of the candidate project arc i, $i \in A_2$.

†This research was conducted while both authors were at Northwestern University, Evanston, Illinois.

Let $P = \{v: v = 1, 2 \ldots, P \leq N\}$ be the set of vertices which are origins and/or destinations. Let x_{is} denote the flow along arc i with destination s, $s \in P$, and let x_i denote the total flow along arc i, $i \in A$. The vector of arc flows, x_i, $i = 1,2 \ldots, n+2m$, will be denoted x. Led d_j^s denote the demand for the (j, s) origin–destination (O–D) pair ($j \in P$, $s \in P$ and $j \neq s$). Define the set

$$Y = \{y = (y_{n+m+1}, y_{n+m,+2}, \ldots, y_{n+2m}) \,|\, y_i = 0 \text{ or } 1\}.$$

This is the set of possible combinations of projects. The binary variables y_i take on the value 1 if project i is chosen, and are 0 otherwise. Let $h_i(x_i, y_i)$ denote the average (generalized)cost of travel over link i. This cost depends both on the flow, x_i, and on whether or not a project is implemented on link i. For a given value of y_i(0 or 1), we will assume that $h_i(x_i, y_i)$ is a convex, Reimann integrable, single-valued and differentiable function of x_i, for $x_i > 0$. Let B and c_i, $i \in A_3$, denote the total available budget and the cost of project i, respectively.

The discrete network design problem may be stated as follows:

(NDP)
$$\min_{y \in Y} \sum_{i \in A} x_i^* h_i(x_i^*, y_i)$$
$$\text{s.t.} \quad \sum_{i \in A_3} c_i y_i \leq B$$

where x_i^*, $i \in A$, is the solution of the following network equilibrium problem with y fixed.

(NEP)
$$\min \sum_{i \in A} \int_0^{x_i} h_i(z, y_i)\, dz$$
$$\text{s.t.} \sum_{i \in \alpha(j)} x_{is} - \sum_{i \in \beta(j)} x_{is} = d_j^s, \; j \in V, \; s \in P, \; s \neq j$$
$$\sum_{s \in P} x_{is} - x_i = 0, \quad i \in A$$
$$x_{is}, \; x_i \geq 0, \quad i \in A, \; s \in P.$$

The sets $\alpha(j)$ and $\beta(j)$ denote arcs originating at j, and terminating at j, respectively.

The choice of projects, y_i, will affect the equilibrium flow pattern x, and hence the objective function value in (NDP), which is a measure of total user cost. The planner may choose the y_i, and then users of the network will distribute themselves on the network to achieve an equilibrium flow pattern, x. The planner cannot dictate a particular flow pattern that may be more efficient in an aggregate sense.

The departure from economically efficient use of the road network (system optimal flow pattern) is a primary source of mathematical difficulty in the attempt to minimize total travel cost in the design process. The total user cost is not necessarily a decreasing function of the decision variables (network improvements). This is at the heart of what has come to be known as Braess' paradox (see Murchland, 1970). In theory at least, some capacity additions to a network can *increase* total user cost.

Ochoa-Rossa and Silva (1968) described a branch-and-bound algorithm for solving (NDP), but the bounding step was dependent upon assumption that additional link improvements would always reduce total user cost. LeBlanc (1975) presented a similar algorithm, but with an important difference in the bound computation. If the objective function in (NEP) is replaced by:

$$\min \sum_{i \in A} x_i h_i(x_i, y_i)$$

we have what is often termed the "system-optimal" problem, as opposed to the "user equilibrium" problem represented by (NEP). Let us denote this "system-optimal" problem as (SOP).

LeBlanc (1975) showed that the objective function value for (NDP) when the x_i^* are from (SOP) is monotonically decreasing as further link improvements are made, in addition to being a lower bound on the objective function using flows from (NEP). Thus, his algorithm uses flows

derived from (SOP) to compute true bounds on the objective function in (NDP), avoiding the potential problems caused by Braess' Paradox. Unfortunately, however, this algorithm is relatively inefficient, at least in part because the bounds are relatively "loose", and becomes computationally prohibitive for large networks.

Other authors have used other methods to simplify (NDP) for solution. For example, Boyce *et al.* (1973) assumed no congestion on network arcs. This eliminates the differences between (NEP) and (SOP), and makes either much easier to solve. This assumption is quite unrealistic, however, because the basic rationale for considering network improvements is often to reduce congestion.

Steenbrink (1974), Abdulaal and LeBlanc (1979) and Dantzig *et al.* (1979) have assumed project investment levels are continuous. This clearly simplifies the problem because it removes the combinatorial aspects, and makes the problem amenable to a number of nonlinear programming algorithms. The computational benefits of using a continuous approximation to the discrete problem can be substantial. For example, Abdulaal and LeBlanc (1979) report computational results which indicate that a continuous approximate solution can be obtained at about 20–25% of the cost of obtaining an exact discrete solution.

The disadvantage to this approach, however, is that it may be difficult to translate the continuous solution into particular projects which can be implemented. This is especially difficult if the continuous solution indicates relatively small improvements on a large number of links. Because projects often incur substantial start-up costs, such a solution may not really be practical.

We have chosen an alternative approach to finding an approximate solution to the discrete network design problem. We retain the discrete nature of the projects, but develop an approximation to the objective function of (NDP) which allows much more efficient solution. Experience with test problems indicates that the computational savings are approximately the same as those attained by using a continuous investment approximation. Thus, this method offers an alternative means of constructing an approximate solution, without forcing the analyst to ignore the discrete nature of alternative network improvement projects. However, this approach must be viewed as a heuristic; since it involves solving an approximation to the real problem, an optimal solution to the real problem cannot be guaranteed. The approach, the algorithms to implement it, and test results for an example problem are discussed in the following sections.

3. AN APPROXIMATE FORMULATION AND RESULTING ALGORITHMS

The network design problem (NDP) can be approximated by the following formulation, which we denote (NDP'):

$$\text{(NDP')} \qquad \min_{x,y\in Y} \sum_{i\in A} \int_0^{x_i} h_i(z,y_i)\,dz$$

$$\text{s.t.} \sum_{i\in\alpha(j)} x_{is} - \sum_{i\in\beta(j)} x_{is} = d_j^s,\ j\in V,\ s\in P, \quad s\neq j$$

$$\sum_{s\in P} x_{is} - x_i = 0,\ i\in A$$

$$\sum_{i\in A_3} c_i y_i \leq B$$

$$x_{is},\ x_i \geq 0,\ i\in A,\ s\in P.$$

This approximation has replaced the total user cost, $\sum_{i\in A} x_i^* h_i(x_i^*, y_i)$ in the objective function of (NDP) by the sum of integrals under the average cost functions, as used in (NEP). Such a replacement accomplishes two things. First, it makes the objective function of (NDP') monotonic in the investment variables, y. This overcomes the problem of potential local optima associated with Braess' Paradox. Second, it suggests ways in which algorithms normally applied to solve the (NEP) can be modified to make the solution of (NDP') easier.

Before proceeding to these algorithms, however, it is necessary to discuss the nature of

(NDP') as an approximation to (NDP). Why should we believe that solving (NDP') is a useful exercise in achieving an optimal or near-optimal solution to (NDP)?

First, note that in two particular cases, (NDP') is exactly equivalent to (NDP). In the case of congestion-free links, so that $h_i(x_i, y_i) = a_i$, a link-specific constant, the solutions will be identical. More generally, if $h_i(x_i, y_i) = b_i x_i^{\eta}$, where b_i is a link-specific constant (which may depend on y_i) and η is a constant common to all links, the solutions will be identical. This is because the system-optimal and user equilibrium flow patterns for any selection of projects will coincide when the link cost functions are of this form. This has been shown by Dafermos and Sparrow (1969). These cases are of interest because the widely-used volume-delay functions suggested by the Federal Highway Administration (1973) are of the form $a_i + b_i x_i^4$. Thus, in situations where congestion is insignificant, the a_i terms dominate and the approximation will be good. In cases of severe congestion, the quartic terms dominate, and again the approximation is good. However, in situtions where the constant and quartic terms are of comparable size, the approximation may be less accurate. In order to investigate this question further, empirical tests have been conducted on the network described by LeBlanc (1975).

This network contains 24 nodes and 76 one-way links. The link volume-delay functions used in the tests are of the form $a_i + b_i x_i^4$. The level of congestion varies across the network. Some parts of the network are relatively lightly loaded, while other parts exhibit more significant congestion. No links are extremely congested however. 128 different network structures were formed by adding and deleting various combinations of links. For each resulting network, an equilibrium assignment was performed, and the values of:

$$\gamma_1 = \sum_{i \in A} x_i^* \, h_i(x_i^*, y_i)$$

and

$$\gamma_2 = \sum_{i \in A} \int_0^{x_i} h_i(z, y_i) \, dz$$

were measured. These data were then used to estimate a regression of γ_1 on γ_2.

The resulting estimated regression is:

$$\hat{\gamma}_1 = -6.36 \times 10^4 + 3.18 \, \gamma_2$$
$$(-27.94) \quad (66.80)$$
$$r^2 = 0.973.$$

The numbers in parenthesis below the estimated coefficients are t-statistics. The regression is highly significant, and the coefficient of variation in the estimate of γ_1 is about 0.01. Thus, there appears to be a strong linear relationship between γ_1 and γ_2, and γ_2 can be used to predict γ_1 quite accurately. This suggests that minimization of γ_2 is likely to be essentially equivalent to minimization of γ_1. With this motivation, we can proceed to a discussion of methods for solving (NDP').

3.1 *Algorithm* 1

The first algorithm we discuss provides an exact solution to (NDP'). It is of the branch-and-backtrack category, and involves three phases. In the first phase (branching), potential solutions are sorted (implicitly) into three subsets: feasible non-dominated solutions, feasible dominated solutions and infeasible solutions. A solution is feasible if it satisfies the budget constraint; otherwise it is infeasible. Feasible solutions which are subsets of other feasible solutions (i.e. contain only projects included in other feasible solutions) are dominated.

In the second phase, (backtracking), the non-dominated feasible solutions are evaluated to find the optimal solution for a given budget. This phase involves the solution (or partial solution) of equilibrium assignment problems (NEP) for given networks. The efficiency of this process is increased by use of bounds which can be computed easily, and serve to eliminate wasted effort in the computation of equilibrium flow patterns for suboptimal network configurations.

The third phase is sensitivity analysis with respect to changes in the available budget. This phase can be omitted if desired, but often produces very valuable additional information.

To present the algorithm more formally, we require additional definitions. A *partial* solution is a vector y such that

$$y_i = \begin{cases} 0, \text{if } i \in P_0, \\ 1, \text{if } i \in P_1; \end{cases}$$

where $P_0 \cap P_1 = \emptyset$, and $P_0, P_1 \subseteq P = (i : i = n + m + 1, \ldots, n + 2m\}$. P_0 is the set of projects in A_3 which we have decided not to implement in a particular partial solution, and P_1 is the set of projects which will be implemented. The set $P \backslash (P_0 \cup P_1)$ will be called the set of free projects. These are projects for which we have not yet reached a decision on implementation. A partial solution is called *augmented* if

$$y_i = \begin{cases} 0, \text{for } i \in P \backslash P_1, \\ 1, \quad \text{if } i \in P_1; \end{cases}$$

and will be denoted by a superscript "a", such as y^a. An augmented solution is one in which all "free" projects are assumed not to be implemented. A partial solution will be called feasible (infeasible) if the corresponding augmented solution is (is not) feasible.

Consider a rooted tree T, where each node k of T is associated with a partial solution y^k. Let N, Q and F be, respectively, the set of nodes in T associated with (partial) solutions which are feasible and may accept at least one more project; feasible but may not accept any more projects; and budget infeasible.

Order the projects in accordance with the increase in their construction costs. For notational simplicity, we denote y_{n+m+i} by y_i, $i = 1,2, \ldots, m$. Let B be a budget level and B_0 its initial value. Let $\gamma_L(x, y)$ denote a lower bound on the value of the objective function (NEP) for a given project set y (the computation of $\gamma_L(x, y)$ will be described later). Include the root of the tree T, node 0, in N with the null solution $y^0 = \{y_i = 0, i \in A_3\}$—the original network. Finally, let C be the set of nodes in T for which a conditional (NEP) problem must be solved. The algorithm is as follows.

Phase 0. (*Initialization*)

Find x^0, the flow pattern from the (NEP) problem for $y = y^0$. Set $B = B_0$, $C = N = \{0\}$, $Q = F = \emptyset$, and $y^* = y^0$. Initialize $\gamma_2(x^*, y^*) = \gamma_2(x^0, y^0)$.

Phase I. (*Branching*)

1. If $N = \emptyset$, go to Step 3; otherwise remove the node with maximum index, k from N. For the associated (partial) solution $y^k = (y_1^k, y_2^k, \ldots, y_{i(k)}^k, -, \ldots, -)$, where $i(k)$ is the index if the last project variable in a solution y^k which has been assigned a value, accept projects $i(k) + 1, \ldots, i^*$ such that i^* is the first index at which $\sum_{i=1}^{i^*} c_i y_i^k > B$. Denote the resulting partial solution as y^f, and connect it to y^k by what will be termed an *extension branch.* Include this in solution F.

2. Create a set of solutions associated with node y^f as follows:

(i) Set $y_{i^*}^f = 0$, and include the resulting solution, y^{i^*-1}, in Q. Attach it to y^f by what will be termed a *covering branch.*

(ii) Set $r \leftarrow i^* - 1$.

(iii) If $r \leq k + 1$, return to step 1; otherwise continue.

(iv) Create a new solution, y^{r-1}, from y^r by setting $y_r^r = 0$. Include y^{r-1}, in N, and attach it to y^f by a covering branch.

(v) $r \leftarrow r - 1$; go to (iii).

3. Remove all solutions y^k which are subsets of a solution $y \in C \cup Q$, and store them in a set D of dominated solution. Then proceed to phase II.

Phase II. (*Backtracking*)

1. If $Q = \emptyset$, go to Step 3; otherwise, remove the node k from Q, where k is the minimum index in Q, and include it in C.

2. Solve the (NEP) for $y = y^k$, using the Frank–Wolfe algorithm (FWA) (Frank and Wolfe, 1956), adopted for network assignment by LeBlanc, *et al.* (1975), with the following modifications:

(i) At iteration l of FWA, find x^l from x^{l-1} in the normal fashion, by solving an "all-or-nothing" assignment problem and then a one-dimensional search.

(ii) Find a lower bound to the value of the objective function of the problem $\gamma_L(x^l, y^k)$. If $\gamma_L(x^l, y^k) \geq \gamma_2(x^*, y^*)$, the value of the objective function of the current best solution, stop FWA and return to Step 1 with y^k discarded. Otherwise, continue.

(iii) If the stopping rule of FWA is met, compare $\gamma_2(x^*, y^k)$ with $\gamma_2(x^*, y^*)$; if $\gamma_2(x^*, y^*) > \gamma_2(x^*, y^k)$, set $\gamma_2(x^*, y^*) = \gamma_2(x^*, y^k)$ and return to step 1; otherwise, return to step 1 with the incumbent solution remaining the best. If the FWA is not to be stopped, set $l \leftarrow l + 1$, and go to (i) above.

3. y^* is the optimal solution. Continue to Phase III if budget sensitivity analysis is desired; otherwise stop.

Phase III (*Sensitivity Analysis*)

Find solution(s) $y^k \in C \cup F$, if any, such that the the least amount of additional budget, ΔB, is required to accept an additional project. *Set* $B \leftarrow B + \Delta B$, remove this solution from the set to which it belongs and include it in N. Return to Phase I. If no such y^k exists, stop.

Discussion. In Phase I, the algorithm produces (implicitly) all alternative solutions which are feasible at the current budget level. For any partial solution to which some projects may be added without violating the budget constraint, the algorithm chooses as many projects (among those which are free in the partial solution) as possible such that the last project makes the solution infeasible. At this point, the algorithm resolves the infeasibility problem of the solution by branching it into a set of successor solutions which collectively preserve the following property of their preceding solution: if the partial feasible solution under consideration contains the optimal solution for the current budget level, so does the set of solutions succeeding it. The process continues in this manner until there is no partial solution which can accept any more projects without violating the current budget constraint.

In Phase II, the algorithm then identifies the best solution among the alternative solutions produced in Phase I. A key aspect of Phase II is the use of computed lower bounds on the objective function value to avoid needless iterations of the FWA, converging to an equilibrium flow pattern for a sub-optimal network. Such a lower bound can be derived using a procedure similar to that used to prove convergence of FWA (Frank and Wolfe, 1956). The derivation is quite lengthy, but leads to the following simple result

$$\gamma_L(x^l, y) = \frac{l+2}{l}\,\gamma_2(x^l, y) - \frac{2}{l}\,\gamma_2(x^0, y). \tag{1}$$

In the interests of space, the derivation is omitted here. The interested reader is referred to Poorzahedy (1980) for details. Note that this lower bound is a simple weighted combination of the starting solution and the current solution.

Poorzahedy (1980) proves that this algorithm is finite and provides an exact solution to (NDP'). However, this is not really a central point, because (NDP') is itself an approximation to (NDP). Thus, the proof is omitted here. We are more concerned with the quality of the approximate solution and with the computational cost of obtaining it, relative to solving (NDP) exactly.

3.2 *Algorithm* 2

Algorithm 1 appears to provide an efficient method to obtain an exact solution to (NDP'), as an approximation to solving (NDP). Since this is already an approximate solution to the original problem, it may be of interest to settle for a near-optimal solution to (NDP'), as this might also be a reasonable approximation to solving (NDP). This would be particularly advantageous if such a near-optimal solution could be obtained even more cheaply.

A very efficient procedure for obtaining an approximate solution to (NDP') can be developed by making one minor change in Algorithm 1, involving the lower bound computation.

The lower bound in eqn (1) is a true lower bound, but in many cases is relatively "loose", at least in the early iterations of the FWA. It is possible to construct a much tighter criterion which will be violated only rarely. This approximate lower bound is:

$$\gamma_L(x^l, y) = 3\,\gamma_2(x^l, y) - 2\,\gamma_2(x^{l-2}, y). \tag{2}$$

It must be emphasized that (2) is not a true lower bound, and thus by using it, there is some risk of cutting off a branch of the tree that contains the optimal solution. However, it is a much tighter criterion than (1), and improves computational efficiency substantially. This is discussed further in Section 4.

3.3 *Remarks on contingency planning*

Often in practice one is interested in having a contingency plan for the actual implementation of optimal strategies. In other words, one would like to be prepared for the case where construction of the optimal set of projects faces serious unexpected obstacles (for example, the construction of one or more projects in the set turns out to be politically objectionable).

One obvious solution for such problems, is, of course, to reject the troubled project(s) and rerun the models for the remaining set of projects. However, one can prepare a contingency plan by minor modifications of the algorithms discussed here, such that a set (including a prespecified number) of near-optimal solutions always accompanies the optimal solution. For example, in the case of Algorithm 1, one may easily retain n-1 near-optimal solutions which are not dominated, and are not rejected in Phase II, Step 2 (iii) of the algorithm.

4. TESTING THE ALGORITHMS

4.1 *The test network*

To compare the efficiency of the procedures outlined above with previous algorithms, the test network of LeBlanc (1975) has been chosen. The network and data represent an aggregate model of the city of Sioux Falls, South Dakota. The original network includes 24 vertices and 76 links as shown in Fig. 1. Links in the network are in pairs representing two-way traffic movements. There are a total of ten candidate projects, of which five are improvements to existing links and five involve construction of new links. These links are numbered in Fig. 1.

The form of the average travel time (volume-delay) functions, denoted $h_i(x_i, y_i)$ for link i, is $a_i + b_i x_i^4$, as recommended by the U.S. Federal Highway Administration (1973), where a_i and b_i are two constants for link i (which may depend on y_i), and x_i is the corresponding flow. These parameters for the existing network are given in Table 1, and parameter values and the costs of construction of various projects are shown in Table 2. The O–D travel volumes used are the same as used by LeBlanc (1975), and thus the trip table is not reproduced here.

4.2 *Computational results*

Computation times for any given problem depend on four principal factors: (1) the number of projects to be considered, (2) the costs of those projects, (3) the available budget, and (4) whether or not sensitivity analysis to changes in the budget is performed. Clearly factors 2 and 3 are related, and a useful way to express this is by forming the ratio of available budget to the sum of costs of all projects. If this ratio is very small, there are relatively few feasible solutions, and the algorithms will identify the optimal (or near-optimal) solution quite rapidly. Computation time will increase as the ratio increases up to some point, and then decline again as the ratio approaches 1.0. For values near 1.0, there are many feasible solutions but most are dominated, since almost all projects can be selected. Hence, the optimal solution can be found relatively quickly, because only non-dominated solutions need be examined.

Comparison of computation times for the algorithms described here against other algorithms proposed by previous authors is difficult because each author has used a different computer, and in many cases, a different test network. However, one useful way of measuring computational effort is in terms of the number of FWA iterations performed, since this is the most time consuming part of the algorithm. Because the algorithm developed by LeBlanc (1975) also uses the FWA, such a measure at least facilitates comparison with his procedure.

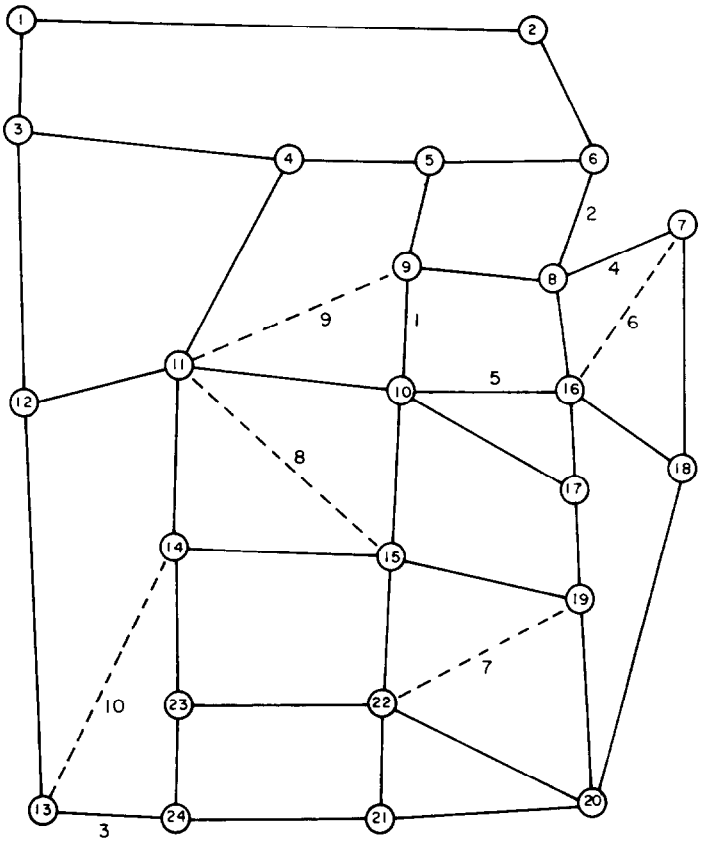

Fig. 1. The test network, with potential projects numbered. Solid lines indicate existing links, and dashed lines potential new links.

Table 1. Link parameters for test network (a in hours, b in hours/[1000 vehicles/day]^4)

Link*	a(x 10^{-2})	b(x 10^{-4})	Link	a(x 10^{-2})	b(x 10^{-4})
1,2	5.96	.00023	15,19	3.50	.00104
1,3	4.34	.00017	15,22	3.50	.00525
2,6	5.17	.12408	16,17	1.67	.04008
3,4	4.31	.00069	16,18	2.69	.00025
3,12	4.14	.00016	17,19	2.31	.05544
4,5	2.16	.00035	18,20	4.46	.00017
4,11	6.46	.15504	19,20	3.99	.09576
5,6	4.17	.10008	20,21	5.72	.13728
5,9	5.03	.00755	20,22	4.71	.11304
7,18	2.18	.00008	21,22	1.67	.04008
8,9	9.61	.23064	21,24	3.29	.07896
8,16	4.82	.11568	22,23	4.00	.09600
10,11	5.00	.00750	23,14	4.25	.10200
10,15	5.87	.00265	23,24	1.88	.04512
10,17	8.04	.19296	9,10	2.75	.00124
11,12	6.46	.15504	6,8	2.17	.05208
11,14	4.42	.10608	13,24	3.72	.08928
12,13	2.98	.00011	7,8	2.50	.01185
14,15	4.52	.10848	10,16	4.50	.10800

* Each link listing is for a pair of identical one-way arcs, one in each direction, with the parameter values indicated.

Table 2. Link parameters and costs for new projects

Link*	Old Values		New Values		
	$a(\times 10^{-2})$	$b(\times 10^{-4})$	$a(\times 10^{-2})$	$b(\times 10^{-4})$	Cost ($)
9,10	2.75	.00124	1.60	.00037	625,000
6,8	2.17	.05208	1.30	.01562	650,000
13,24	3.72	.08928	2.20	.02678	850,000
7,8	2.50	.01185	1.50	.00355	1,000,000
10,16	4.50	.10800	2.70	.03240	1,200,000
7,6	Potential New Links		3.00	.00321	1,500,000
19,22			1.00	.00042	1,650,000
11,15			1.50	.00411	1,800,000
9,11			2.14	.00028	1,950,000
13,14			1.00	.00160	2,100,000

* Each link listing is for a pair of identical one-way arcs, one in each direction, with the parameter values indicated.

As shown in Fig. 2, there is a strong linear relationship between CPU time and FWA iterations, at least for the algorithms described here. The CPU times are for a FORTRAN IV coding on a CDC 6600 computer.

Table 3 summarizes the computational results from a set of experiments involving varying numbers of projects and budget levels. Note that algorithm 2 is consistently more efficient than algorithm 1, although both are quite effective. For comparison, LeBlanc (1975) reported an experiment equivalent to the first line of Table 3, 5 projects and a budget of $3,000,000, for a budget/cost ratio of 0.462. His algorithm required 245 FWA iterations to identify the optimal solution. Thus, the procedures described here appear to be 3–5 times faster. It should also be noted that in each case reported in Table 3, the true optimal solution was identified by both algorithms.

It is interesting to note that this improvement in computational efficiency is approximately the same as reported by Abdulaal and LeBlanc (1979) by using a continuous variable approximation rather than discrete projects. The advantage of our approach is that it does not present the problems of interpretation resulting from the continuous variable solution.

Figure 3 shows an example of the sensitivity analysis information produced by the algorithms for the case of six potential projects ($n = 6$). The initial budget was assumed to be $2,000,000, and was incremented until all six projects could be constructed, at a budget of $5,825,000. The numbers next to each line segment on the graph indicate the projects selected. In each case, the solutions produced by the two algorithms were identical. Algorithm 1 required 395 FWA iterations (79 sec) and algorithm 2 required 301 FWA iterations (63 sec) to produce the entire set of solutions shown in Fig. 3. For details on additional computational experiments, the interested reader is referred to Poorzahedy (1980).

Based on our computational experience to date, it appears that these algorithms can produce a complete sensitivity analysis, of the sort shown in Fig. 3, at about the same computational

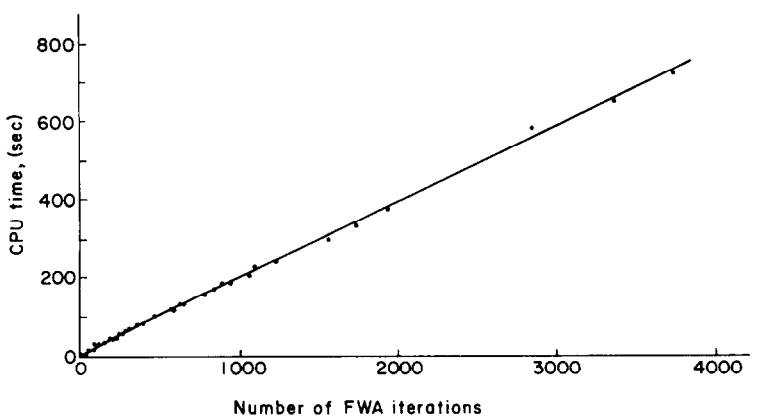

Fig. 2. Computation time as a function of the number of Frank–Wolfe iterations performed.

Table 3. Summary of computational effort for various experiments

Number of projects	Ratio of budget to sum of project costs	Algorithm	No. of F-W iterations	CPU secs.
5	.462	1	75	20
		2	51	15
	.925	1	60	17
		2	48	15
6	.343	1	85	22
		2	72	19
	.687	1	104	26
		2	78	21
7	.401	1	119	29
		2	103	26
	.803	1	88	23
		2	69	19
8	.431	1	115	28
		2	133	31
	.755	1	158	36
		2	132	31
9	.535	1	317	67
		2	295	63
	.713	1	312	66
		2	218	48
10	.450	1	473	97
		2	469	96
	.675	1	565	115
		2	477	98

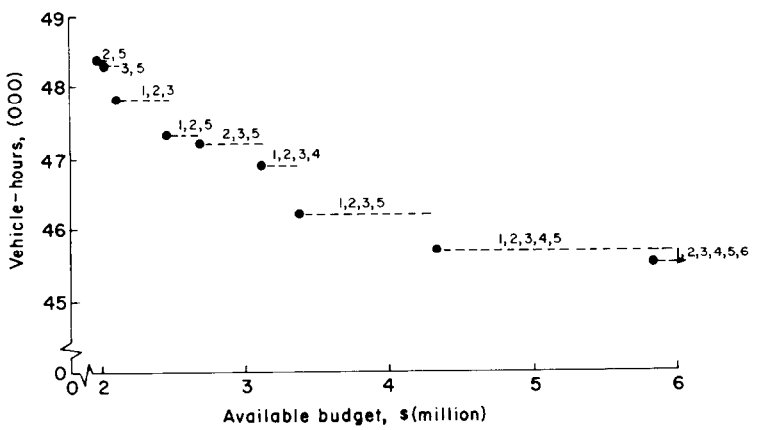

Fig. 3. Optimal solutions and resulting objective function value as budget changes.

cost as previous algorithms would require to find an optimal solution for a single budget level. This is a very appealing feature of these procedures, because the information represented by Fig. 3 is likely to be much more valuable to a decision-maker than is a single optimal solution.

5. CONCLUSIONS

Two approximate algorithms for solving the discrete network design problem have been discussed. They are approximate in the sense that they solve a problem which is an approximation of the real problem but which is easier to solve. The first algorithm yields an exact

solution to this approximate problem, while the second, in general, will produce an approximate solution, but with even less computation.

The algorithms appear to perform 3–5 times faster than previous exact algorithms for the discrete network design problem, and in the cases tested to date have almost always identified the true optimal solution. The increase in computational speed appears comparable to that achievable by using a continuous approximation to the discrete problem, but does not engender the sorts of difficulties in interpretation of the solution that arise from a continuous approximation.

Furthermore, these algorithms have been designed to enable sensitivity analysis to changes in the available budget to be carried out easily. This is also a major advantage over previous methods, because it provides vital information for informed decision-making on network investments. It is also quite easy to retain several near-optimal solutions at any stage so as to allow contingency planning on the part of the decision-makers.

Acknowledgements—The authors wish to thank Joseph Schofer, Frank Koppelman and Alex Anas, all of Northwestern University, for helpful comments in the course of this research, and a referee for useful suggestions which improved the paper.

REFERENCES

Abdulaal M. and LeBlanc L. J. (1979) Continuous equilibrium network design models. *Transpn Res.* **13B**, 19–32.

Boyce D. E., Farhi A. and Weischedel R. (1973) Optimal network problem: a branch-and-bound algorithm. *Envnmt Plan.* **6**, 519–533.

Dafermos S. C. and Sparrow F. T. (1969) The traffic assignment problem for a general network. *J. Res. NBS* **73B**(2) 91–118.

Dantzig G. D., Harvey R. P., Lansdowne Z. F., Robinson D. W. and Maier S. F. (1979) Formulating and solving the network design problem by decomposition. *Transpn Res.* **13B**, 5–17.

Federal Highway Administration, U.S. Department of Transportation (1973), *Traffic Assignment*, Wasington, D.C.

Frank M. and Wolfe P. (1956) An algorithm for quadratic programming. *Naval Res. Logistics Quart.* **3**, 95–110.

LeBlanc L. J. (1975) An algorithm for the discrete network design problem. *Transpn Sci.* **9**, 183-199.

LeBlanc L. J., Morlok E. K. and Pierskalla W. P. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transpn Res.* **9**, 309–318.

Murchland J. D. (1970), Braess' paradox of traffic flow. *Transpn Res.* **4**, 391–394.

Ochoa-Rosso F. and Silva A. (1968) *Optimum Project Addition in Urban Transportation Networks via Descriptive Traffic Assignment Models*, MIT Department of Civil Engineering, *Res. Rep. R*68–44.

Poorzahedy H. (1980)*Efficient Algorithms for Solving the Network Design Problem*, Ph.D. Diss., Northwestern University, Department of Civil Engineering , Evanston, Illinois.

Steenbrink P. A. (1974) Transportation network optimization in the Dutch integral transportation study. *Transpn Res.* **8**, 11–27.