# A FLOW ALGORITHM FOR NETWORK SYNCHRONIZATION

**REFAEL HASSIN**

*Tel Aviv University, Tel Aviv, Israel*

(Received November 1992; revisions received October 1993, June 1994; accepted November 1994)

The problem we treat is defined on a graph where each node is associated with a variable and there are loss functions defined on the arcs, depending on the difference between the corresponding node variables. The objective is to compute values for the node variables so as to minimize the sum of losses. We exploit the relation between this problem and network flows optimization and use it in developing an approximation algorithm for the problem. A main application of the problem is the synchronization of fixed cycle traffic signals.

This paper describes an application of network flow algorithms to a class of scheduling problems that involve synchronization of node variables. The *network synchronization* problem is formulated as follows: Let $G = (N, A)$ be a graph with a node set $N$ and an arc set $A$. For each $i \in N$ define a variable (a *potential*) $u_i$. In some cases of interest these variables may be constrained to be discrete and in others they may be continuous. For each arc $(i, j) \in A$ a *loss function* $f_{ij}(\Delta_{ij})$ is given, where $\Delta_{ij} = u_j - u_i$ denotes the *offset* (or, *tension*) at $(i, j)$. The problem is to determine a set of values $u_i$ $i \in N$ minimizing $\Sigma_{(i,j)\in A} f_{ij}(\Delta_{ij})$.

An especially interesting case of this general problem is when the loss functions are *periodic* with a common period (which we denote $c$). Thus, $f_{ij}(\Delta_{ij} + c) = f_{ij}(\Delta_{ij})$. Without loss of generality, it can be assumed in this case that $u_i$ is restricted to $[0, c)$ for every $i \in N$ and that $\Delta_{ij}$ is defined as $u_j - u_i (\text{mod } c)$ so that it is also in $[0, c)$. Network synchronization with periodic loss functions can model *periodic scheduling* problems. Several such problems are discussed in Serafini and Ukovich (1989a, b) and their references.

In a geometric interpretation, the periodic synchronization problem is that of locating points on a circle so as to minimize the sum of loss functions depending on the distances between pairs of points along the circle in a given orientation. A more general problem allows locating the points on a general graph rather than on a single arc of a circle. The *p-median problem with mutual communication* is obtained when the loss is proportional to the distance between the points (see Tamir 1993 for recent results).

Several other applications of the network synchronization problem are described in Rockafellar's book (1984, Chapter 7F). When the loss functions are convex, Rockafellar, and Karazanov and McKormick (1993) contain solution algorithms. In particular, Karazanov and McKormick describe a strongly polynomial algorithm when the functions are convex and piecewise linear. However, this is often not the case in interesting applications, and in particular periodic loss functions are not convex.

A main area of application of the problem with periodic loss functions is that of traffic signal synchronization: traffic signals are set at the nodes, with a common period of length $c$. At each node the cycle is partitioned into subintervals, and each subinterval is characterized by a fixed set of allowed driving possibilities (straight-through driving or turns). Our task is to decide how much time after the beginning of a given green interval at a node should another given green interval at another node start.

To formulate this signal synchronization problem in terms of our opening paragraph, we associate with each $i \in N$ a reference point of time, such as a beginning of some green interval at the node. Then, $u_i$, $i \in N$, can be set to the time difference from $i$th reference point to the previous occurrence of node 1's reference point (thus setting $u_1 = 0$). It is often assumed, as we do here, that the objective function can be well approximated by an *arc separable* function (Allsop 1986, Gartner and Little 1975, Gartner et al. 1975, Importa and Sforza 1982, Hillier 1967). This assumption greatly simplifies the solution approach, and it is often claimed to be sufficiently accurate under heavy traffic conditions. (Nonseparable functions are more realistic but also harder to evaluate and are usually computed through a simulation subroutine.)

Importa and Sforza and Serafini and Ukovich (1989a, b) developed branch-and-bound algorithms for network synchronization problems. Such algorithms typically are useful for solving only small instances. Robertson developed "TRANSYT," a program for synchronizing traffic signals, that applies local search (Foulds 1986, Robertson 1969, Tsay and Wang 1989). In this local search algorithm, a new solution differs from the previous one by the offsets of the arcs incident with a single node.

In this paper we develop a local search heuristic for network synchronization. Our definition for the neighborhood of a solution allows moves to a much larger set of new solutions relative to the simple local search adopted by TRANSYT. Specifically, we allow changes in offsets of arcs incident with a subset of the node set. Our numerical

*Subject classifications:* Network flow algorithms: application to synchronization of node variables. Production scheduling: synchronization of cyclic work stations. Transportation traffic models: traffic signal syncronization.
*Area of review:* OPTIMIZATION.



0030-364X/96/4404-0570 $01.25

tests indicate that our more sophisticated local search is superior. We note that we only considered separable loss functions while the typical application of TRANSYT assumes nonseparable functions. However, the general idea of extending the neighborhood of the local search applies in these cases as well. The problem that is left open is that of selecting the subset of nodes whose variables are changed. The specific method that we apply relies on the assumption of separability.

We next characterize the locally optimal solutions in our network synchronization problem. After discussing the complexity of the problem we observe that a local optimum can be reached by applying certain dual algorithms for network flow optimization. We also provide some numerical data indicating that our approach can be successful in obtaining good solutions to medium size (and possibly also to large size) problems.

## 1. A MATHEMATICAL FORMULATION

We now present the problem as a mathematical program:

$$\text{Minimize} \sum_{(i,j)\in A} f_{ij}(\Delta_{ij}) \tag{1}$$

subject to

$$\Delta_{ij} = u_j - u_i, \qquad (i,j) \in A.$$

Problem 1 can be approximated by replacing $f_{ij}$ by an appropriate piecewise linear function. Such an approximation requires the use of binary variables.

We note that Problem 1 can also include single variable functions $f_i(u_i)$ by introducing a dummy variable $u_0$ and presenting these functions as $f_{0i}(u_i - u_0)$. Also, bounds such as $a_i \leq u_i \leq b_i$ or $a_{ij} \leq u_j - u_i \leq b_{ij}$ can be incorporated by appropriate convex penalty functions.

We now comment on some aspects of the above formulation in the case of periodic loss functions. In this case, the node variables, $u_i$, can be restricted to $[0, c)$, so that it is sufficient to define $f_{ij}$ on $[-c, c]$ adding constraints $0 \leq u_i \leq c$, $i \in N$, or $-c \leq \Delta_{ij} \leq c$, $(i,j) \in A$.

Gartner and Little, Gartner et al., and Importa and Sforza, formulated the periodic problem with the arc variables $\Delta_{ij}$ directly defined, not through the node variables $u_i$. Since the arc variables are not independent the problem is formulated with additional constraints expressing this dependence. These constraints state that the sum of offsets over each cycle, $C_l$, of $G$ (weighted by $+1$ or $-1$ according to the arc's orientation in $C_l$) must be equal to $k_l + c \cdot n_l$, where $k_l$ is a constant and $n_l$ is a variable restricted to be integral. The minimum number of such constraints is the size, $|A| - |N| + 1$, of a basic set $B$ of cycles (Gartner 1972a). This formulation requires $|A| - |N| + 1$ integral variables, $n_l$, $l \in B$. If $f_{ij}$ is convex within some interval of size $c$ then a linear approximation for $f_{ij}$ can be formulated with no additional integer variables. This is done by an appropriate translation of $\Delta_{ij}$.

Our formulation eliminates the integral variables $n_l$. However, binary variables are needed even in the convex case since $\Delta_{ij}$ cannot be constrained to an interval of size $c$ but only to an interval of size $2c$, and convexity is lost. The main advantage of our formulation is in its suitability to the heuristic approach to be presented below.

## 2. COMPLEXITY OF THE PROBLEM

Serafini and Ukovich (1989a, b) proved that the problem is NP-hard, assuming that the node variables are discrete. Their reduction from the traveling salesman problem is valid even for binary periodic loss functions that are symmetric within each cycle. For completeness we repeat this reduction in a form that suits our goals.

Suppose we want to find out whether a given undirected connected graph with node set $N$ and edge set $E$ is Hamiltonian. Define a network synchronization problem on the complete directed graph $G = (N, A)$, with period $c = |N|$ and loss functions

$$f_{ij}(u_j - u_i) = \begin{cases} 0 & \text{if } u_j - u_i \bmod c \in [1, c-1] \text{ and } (i,j) \in E, \\ 0 & \text{if } u_j - u_i \bmod c \in [2, c-1] \text{ and } (i,j) \notin E, \\ 1 & \text{otherwise.} \end{cases}$$

It is now claimed that the graph is Hamiltonian if and only if there are values $u_i \in [0, c]$ $i \in N$ such that the total loss is zero. Such a solution must have $|u_j - u_i| \geq 1$ $\forall i, j \in N$ $i \neq j$, and therefore $u_i$ $i \in N$ are distinct and obtain all the values in $\{1, \ldots, c\}$. The difference $|u_j - u_i|$ may be 1 only if $(i,j) \in E$. Therefore, the existence of such a solution is possible only if the graph is Hamiltonian. The "if" part is also clear now.

It should be noted that the above loss functions can be replaced by continuous piecewise linear functions that are zero in the same domains and strictly positive elsewhere. Thus one can easily verify that the hardness of the problem holds also for continuous symmetric and periodic piecewise linear loss functions that have at most two breakpoints within a period.

Clearly, when the node variables are continuous, $c$ can be normalized to any value. However, when they are discrete, $c$ represents the number of distinct values that the node variables may have. We now strengthen the hardness result for the discrete case by showing that it holds also for $c = 2$. In this case $u_i$ are binary variables. The offsets $u_j - u_i$ may obtain values in $\{0, 1, -1\}$. We will use, however, symmetric loss functions so that only the absolute values of the offsets matter, and they are in $\{0, 1\}$.

Our reduction is from the *minimum cluster problem* (the minimization version of the max cut problem (Garey and Johnson 1978)): given an undirected graph with node set $N$ and edge set $E$, we want to find a partition $(S, N \setminus S)$ of $N$ such that the number of edges with both ends in the same part is minimized. To reduce this problem to ours we consider the directed graph $G = (N, A)$ where $A$ is an arbitrary orientation of $E$, and the following loss functions:

$$f_{ij}(u_j - u_i) = \begin{cases} 0 & \text{if } |u_j - u_i| = 1, \\ 1 & \text{if } u_j - u_i = 0, \end{cases} \qquad (i,j) \in A.$$

Clearly, minimizing the sum of these functions over $u_i \in \{0, 1\}$ is equivalent to solving the minimum cluster problem on this graph.

It may be interesting to note that for loss functions

$$f_{ij}(u_j - u_i) = \begin{cases} c_{ij} & \text{if } |u_j - u_i| = 1, \\ 0 & \text{if } u_j - u_i = 0, \end{cases} \quad (i, j) \in A,$$

the problem turns out to be that of computing a *minimum* (weighted) cut in a graph, so that this case is polynomially solvable.

## 3. OPTIMALITY CONDITIONS

The NP-hardness of the network synchronization justifies the development of heuristic approaches. The approach we suggest is that of local search. It requires a method for producing initial solutions, and a definition of a *neighborhood* for each feasible solution. At each iteration, the neighborhood of the current solution is scanned for a solution with a better objective value, and if none exists the algorithm is stopped and the last solution is declared as local optimum. Since in general a local optimum may not be optimal, the procedure is repeated from different initial solutions, and finally the best local optimum is selected. Various techniques have been suggested, that extend and sometimes improve this simple scheme. The main difference is that they also allow moving to solutions with worse objective value or infeasible solutions (for example, simulated annealing or tabu search).

In our case, the feasibility issue doesn't appear because any vector $u \in \mathbb{R}^{|N|}$ is feasible. Therefore, a simple method for selecting initial solutions is that of randomly generating such vectors. In the periodic case, each component, $u_i$, can be uniformly selected from $[0, c)$.

In this section we characterize a most natural choice of a neighborhood, namely, the geometric neighborhood of the solution vector. Thus, a solution is locally optimal if there is no direction in $\mathbb{R}^{|N|}$ in which the objective function improves when moving from the current solution. We note that similar results are implicit in Rockafellar's work (mainly Chapter 8).

Let $\mathbb{B}^n$ be the set of $n$ dimensional binary vectors with at least one positive coordinate. For $x, d \in \mathbb{R}^n$ we denote by $f'(x, d)$ the directional derivative of $f$ at $x$ along the direction $d$.

**Theorem 1.** *Let $g_{ij}: \mathbb{R} \to \mathbb{R}$ be given functions having right and left derivatives everywhere. Let $f: \mathbb{R}^n \to \mathbb{R}$ be of the form*

$$f(x_1, \ldots, x_n) = \sum_{i,j} g_{ij}(x_i - x_j) .$$

*Let $x \in \mathbb{R}^n$. If $f'(x, b) \geq 0$ for all $b \in \mathbb{B}^n$, then, $f'(x, d) \geq 0$ for all $d \in \mathbb{R}^n$. (Similarly, if $f'(x, b) \leq 0$ for all $b \in \mathbb{B}^n$, then, $f'(x, d) \leq 0$ for all $d \in \mathbb{R}^n$.)*

**Proof.** Since $f$ is a function of the differences $x_j - x_i$, it is clearly sufficient to consider vectors $d$ such that $\min\{d_i | i = 1, \ldots, n\} = 0$.

It is sufficient to prove that if $f'(x, d) < 0$ for a vector $d$ with more than two distinct coordinate values then there must exist $b \in \mathbb{B}^n$ such that $f'(x, b) < 0$.

Let $d$ be a direction such that

$$f'(x, d) < 0 , \tag{2}$$

and $d$ has a minimal number of distinct coordinate values among all directions with negative directional derivatives at $x$. We want to prove that $d$ has at most two distinct coordinate values. Assume on the contrary that it has at least three distinct coordinate values.

Let $d_{\max} = \max\{d_i | i = 1, \ldots, n\}$, $S = \{i | d_i = d_{\max}\}$, $\bar{S}$ the complement of $S$. Let $d_{\max'} = \max\{d_i | d_i < d_{\max}\}$ be the largest coordinate value which is smaller than $d_{\max}$. Let $b \in \mathbb{B}^n$ be the direction defined by $S$, that is, $b_i = 1$ if $i \in S$ and $b_i = 0$ if $i \in \bar{S}$. Since $b$ has only two distinct coordinate values, it follows from our assumption on the minimality of $d$ that

$$f'(x, b) \geq 0 . \tag{3}$$

Define a direction $d'$ as follows:

$$d'_i = \begin{cases} d_i & i \in \bar{S}, \\ d_{\max'} & i \in S. \end{cases}$$

Since $d'$ has less distinct coordinate values than $d$, it follows from our minimality assumption on $d$ that

$$f'(x, d') \geq 0 . \tag{4}$$

From the structure of $f$,

$$f'(x, d) = \sum_{i,j} g_{ij}^{(+)}(x_j - x_i)(d_j - d_i)^+ - \sum_{i,j} g_{ij}^{(-)}(x_j - x_i)(d_i - d_j)^+ ,$$

where $g_{ij}^{(+)}$ and $g_{ij}^{(-)}$ are the one-sided derivatives of $g_{ij}$, and for a given $a$ $a^+ = \max(a, 0)$.

Let $\delta = d_{\max} - d_{\max'}$.

$$\begin{aligned} f'(x, d) - f'(x, d') &= \sum_{i \in S} \sum_{j \in \bar{S}} g_{ij}^{(+)}(x_j - x_i)\delta - \sum_{i \in \bar{S}} \sum_{j \in S} g_{ij}^{(-)}(x_j - x_i)\delta \\ &= \delta f'(x, b) \geq 0 , \end{aligned}$$

where the inequality follows from (3). Therefore, $f'(x, d) \geq f'(x, d') \geq 0$, where the second inequality is (4). This is in contrast to (2).

As a corollary of Theorem 1 we obtain the following theorem:

**Theorem 2.** *Let $f: \mathbb{R}^n \to \mathbb{R}$ be of the form*

$$f(x_1, \ldots, x_n) = \sum_{i,j} g_{ij}(x_i - x_j) ,$$

*where $g_{ij}: \mathbb{R} \to \mathbb{R}$ are convex functions. A necessary and sufficient condition for a vector $x \in \mathbb{R}^n$ to be a global minimum of $f$ over $\mathbb{R}^n$ is that for all $b \in \mathbb{B}^n$, $f'(x, b) \geq 0$. (Similarly, for concave functions, the condition for a global maximum is $f'(x, b) \leq 0$.)*

**Proof.** Since $\{g_{ij}\}$ are convex, $f$ is also convex, and the necessary and sufficient condition that $x$ is a global minimum is that the directional derivatives $f'(x, d)$ are nonnegative in every direction $d \in \mathbb{R}^n$. The claim now follows from Theorem 1.

## 4. NETWORK FLOW

Let $G = (N, A)$ be a directed graph. The *minimum cost circulation problem* is to

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to

$$\sum_{j|(i,j)\in A} x_{ij} - \sum_{j|(j,i)\in A} x_{ji} = 0, \quad i \in N, \tag{5}$$

$$l_{ij} \leq x_{ij} \leq u_{ij}, \qquad (i, j) \in A.$$

In (5), $x_{ij}$ is the flow along arc $(i, j)$, $c_{ij}$ is the unit cost of this flow, and $l_{ij}$ and $u_{ij}$ are lower and upper bounds on this flow satisfying $l_{ij} \leq u_{ij}$. The dual problem can be written as follows:

$$\text{Maximize} \sum_{(i,j)\in A} g_{ij}(v_{ij}),$$

subject to (6)

$$v_{ij} = p_j - p_i + c_{ij}, \qquad (i, j) \in A,$$

where $g_{ij}(v_{ij}) = l_{ij}\max\{v_{ij}, 0\} + u_{ij}\min\{v_{ij}, 0\}$.

Consider a given vector $p \in R^{|N|}$. Define modified bounds as follows:

$$(l_{ij}^p, u_{ij}^p) = \begin{cases} (l_{ij}, l_{ij}), & \text{if } c_{ij} - p_i + p_j > 0, \\ (l_{ij}, u_{ij}), & \text{if } c_{ij} - p_i + p_j = 0, \\ (u_{ij}, u_{ij}), & \text{if } c_{ij} - p_i + p_j < 0. \end{cases}$$

A set $S \subset N$, $S \neq \phi$, $S \neq N$, is called a *cut*. The set of arcs $\{(i, j) \in A | i \in S, j \in \bar{S}\}$ is denoted $(S, \bar{S})$. With respect to a given vector $p$ we define

$$I(S) = \sum_{(i,j)\in(\bar{S},S)} l_{ij}^p - \sum_{(i,j)\in(S,\bar{S})} u_{ij}^p. \tag{7}$$

Note that $I(S)$ is the derivative of the dual objective in the direction $b \in \mathbb{B}^{|N|}$ defined by $S$. A cut $S$ is *positive* if $I(S) > 0$.

An increase of $p_i$ by $\epsilon > 0$ increases all $v_{ji}$ values by $\epsilon$ and decreases all $v_{ij}$ values by $\epsilon$. Therefore, an increase of all $p_i$, $i \in S$, by $\epsilon > 0$ small enough adds $\epsilon I(S)$ to the dual objective function. Noting that the functions $g_{ij}$ are concave, the following theorem (Hassin 1983) can be obtained as a corollary of Theorem 2:

**Theorem 3.** *A vector $p$ is optimal for Problem 6 if and only if $G$ contains no positive cuts with respect to the modified bounds $(l^p, u^p)$.*

Note that each $g_{ij}$, $(i, j) \in A$, is piecewise linear and concave. Its single breakpoint is at $p_i - p_j = c_{ij}$. By replacing an arc $(i, j) \in A$ by several arcs $(i, j)_1, \ldots, (i, j)_k$ in parallel, with bounds $l_{ij1}, \ldots, l_{ijk}$ and $u_{ij1}, \ldots, u_{ijk}$, and costs $c_{ij1} < \cdots < c_{ijk}$, the dual program is changed so that $g_{ij}$ is replaced by a piecewise concave function with breakpoints at $p_i - p_j = c_{ij1}, \ldots, c_{ijk}$ and slopes $\sum_{r\geq p} l_{ijr} + \sum_{r<p} u_{ijr}$, $p = 1, \ldots, k + 1$. Indeed, any piecewise concave function can be obtained this way (see Rockafellar, sections 8C, 8E and 8F), and this construction could be used to prove Theorem 2 from Theorem 3.

Theorem 3 suggests a general approach for solving the dual problem. At each iteration a positive cut $S$ is computed and each $p_i$, $i \in S$, is increased by the same amount until some modified bound changes.

Some strategies for computing positive cuts have been shown to yield polynomial time algorithms for the minimum cost flow problem and its dual (see the next section). These algorithms can be used to minimize piecewise linear functions of the type considered in Theorem 2 in polynomial time (where the input contains explicit descriptions of the $g$ functions by their slopes and breakpoints).

## 5. THE ALGORITHM

We observe that Theorem 2 justifies the correctness of a class of "dual" network flow algorithms. This class of algorithms is quite broad as shown by Hassin (1983) (see also Sandi (1986), Lovetskii and Melamed (1987)). On similar grounds, Theorem 1 can be used to prove that these algorithms terminate in a local optimum for any loss function. Below, we describe the general procedure as well as a detailed description of one variation.

We assume that the loss functions $f_e$, $e \in A$, are piecewise linear. For a given vector of offsets $\Delta$, let $z_e^+$ ($z_e^-$) be the right (left) derivative of $f_e(\Delta_e)$.

Let $t_e^+$ ($t_e^-$) be the size of the interval in the positive (negative) direction within which the derivative is fixed.

Let

$$I(M) = \sum_{e\in(\bar{M},M)} z_e^+(\Delta) - \sum_{e\in(M,\bar{M})} z_e^-(\Delta). \tag{8}$$

In the network flow model $z_e^+(\Delta) = l_e^p$ and $z_e^-(\Delta) = u_e^p$. Hence, Equation (7) is a special case of Equation (8). Let

$$T(M) = \min\{\min_{e\in(\bar{M},M)}\{t_e^+\}, \min_{e\in(M,\bar{M})}\{t_e^-\}\}.$$

Note that an increase in $u_i$ reduces $\Delta_e$ if $e = (i, j)$ for some $j$, and increases $\Delta_e$ if $e = (j, i)$ for some $j$. Therefore, for $\epsilon \leq T(M)$, an increase of all $u_i$, $i \in M$, by $\epsilon$ adds $\epsilon I(M)$ to the objective function $\sum_{e\in A} f_e(\Delta_e)$. It comes out that by Theorem 1, a necessary and sufficient condition for *local optimality* is $I(M) \geq 0$ for all $M \subset N$.

The algorithm that we propose starts with an arbitrary set $u_i$, $i \in N$. At each iteration a *negative cut*, that is, a set $M$ with $I(M) < 0$, is found and $u_i$, $i \in M$, are increased by $T(M)$. The algorithm stops when no negative cut exists. Since the resulting solution is only a local optimum, the process repeats with different starting solutions $u_i$, $i \in N$, and the best local optimum is finally chosen. In our numerical study we have selected the starting solutions randomly, but there may be better selection rules such as those suggested by Wong and Morris (1989).

A crucial part of the algorithm is the computation of a negative cut or determination that none exists. Several methods have been developed for the network flow problem, and they can be applied to the synchronization problem; see for example Ervolina and McCormick (1993), Hassin (1983, 1992), McCormick and Ervolina (1994). The methods differ both by the type of a positive (for the network flow problem) cut they search for, and the way the search is conducted. Hassin (1983) developed a *tree-search* algorithm for computing a *most positive cut.* McCormick and Ervolina compute such a cut by applying a maximum flow algorithm. Alternatives discussed in the above papers are the cut whose mean (node-wise or arc-wise) value is maximum, and they require computing a maximum cut as a subproblem. The complexity of finding the first (or most) positive cut by the tree-search algorithm is linear in the number of arcs. The other methods that use a maximum flow algorithm require greater effort per iteration but may require less iterations. In particular, some of the resulting algorithms have been shown to have a polynomially bounded number of iterations while the others do not have this property. However, a comparison of all of these approaches based on computational experience is not available. For completeness, we now describe in detail one such algorithm. It performs the same sequence of steps as the tree-search algorithm but stops when the first (rather than the most) negative cut is found. We have selected this version because it is simpler to describe.

For a given vector of offsets $\Delta$ let $B(\Delta) \subset A$ be the set of arcs such that $\Delta_e$ is a breakpoint of $f_e$. This set plays a role similar to that of a "basis" in linear programming. As will be shown, the step where a negative cut is computed can be executed very efficiently when $B(\Delta)$ does not contain a cycle. This is always the case if the breakpoints are "independent" in the following sense: for any solution $u$, the arcs that attain breakpoints of their respective loss functions induce no cycle. This nondegeneracy property cannot be assumed in practice. However, cycles can be avoided by applying any of the well-known rules used to eliminate cycling in linear programming, such as perturbation of the data or lexicographic ordering (see, for example, Schrijver (1986). We will describe the algorithm with a lowest-index rule. The arcs are assumed to be arbitrarily indexed. Then, whenever the offsets of several arcs reach a breakpoint simultaneously, only the one with the lowest index actually joins $B$. This guarantees that $B(\Delta)$ never contains a cycle since only the $\Delta$ values in $(M \times \bar{M}) \cup (\bar{M} \times M)$ were changed, and exactly one of these arcs will be in the new set $B(\Delta)$.

We are now ready to describe the tree-search algorithm for computing a locally optimal solution. For more details see Hassin (1983).

### Negative Cut Algorithm

**Input:** A set, $u$, of node variables, with the corresponding set of offsets, $\Delta$.

**Output:** A locally optimal solution $u$.

$B :=$ a maximal subset of $\{e \in A | \Delta_e$ *is a breakpoint of* $f_e\}$, containing no cycles;
**while** (a local optimum has not been found)
{ $S := N$;
**for** ($i \in N$)
{ $d(i) :=$ the number of arcs of $B$ incident with $i$;
$P(i) := \{i\}$;
}
**while** (a negative cut has not been found)
{ **if** ($S = \phi$) **return** $u$ [$u$ is a local optimum];
**find** $i \in S$ such that $d(i) \leq 1$;
$S := S \setminus \{i\}$;
**if** ($I(P(i)) \geq 0$ and $d(i) = 1$)
{ $e :=$ the unique arc of $B$ incident with $i$;
$j :=$ the other end of $e$;
$d(j) := d(j) - 1$;
**if** ($I(P(i)) + z_e^- - z_e^+ < 0$) $P(j) := P(j) \cup P(i)$;
}
**if** ($I(P(i)) < 0$)
{ $M := P(i)$ [$M$ is a negative cut];
**for** ($i \in M$) $u_i := u_i + T(M)$;
**for** ($(i, j) \in A$) $\Delta_{ij} = u_j - u_i$;
$B := B(\Delta)$;
}
}
}

### Comments

$S$ is the set of nodes that were not scanned yet.

$P(i)$ is a set defined for each $i \in S$, with the following properties: $i \in P(i)$; $P(i) \cap S = \{i\}$; for every $j \in P(i)$ there exists a negative cut containing $j$ if and only if there exists such a cut containing $P(i)$.

The revision of the sets $P(i)$ is the crucial step. Suppose, for example, that $e = (i, j)$. An increase in $u_j$ results in a similar increase in $\Delta_e$, and consequently the objective function also increases in rate $z_e^+$. If, however, $P(i)$ is appended to $P(j)$ and thus $u_j$ is increased together with all $u_l$, $l \in P(i)$, this rate changes to $I(P(i)) + z_e^-$. Therefore, we append $P(i)$ to $P(j)$ if and only if $I(P(i)) + z_e^- - z_e^+ < 0$.

When revising the set $B$, if several arcs formerly not in $B$ are supposed to join it, join only the one of lowest index among them. The other ones retain their $z^- = z^+$ value and stay out of $B$ (with $t_e^+ = 0$ if $\Delta_e$ was increased to its breakpoint, or $t_e^- = 0$ if $\Delta_e$ was decreased to that point). This way we preserve the property that $B$ does not contain cycles, and consequently, as long as the set $S$ is not empty it must contain a node $i$ such that $d(i) \leq 1$.

Note that $T(M) = 0$ is possible, and then $\Delta$ is not changed, only $B$.

## 6. COMPUTATIONAL EXPERIENCE

We applied the algorithm of Section 5 to an example used by Gartner (1972b) and Importa and Sforza. In this example, $c$ is evenly divided to eight, and losses are defined by discrete functions on the eight possible offset values. This

setting is particularly convenient for implementation of our algorithm. We normalized $c$ to 8 so that the step size is always either zero or one. The slopes of the functions are defined by the differences of the loss function in the relevant directions. The typical situation is very degenerate in the sense that all the functions are always at a breakpoint. Initially integer values $u_i \in \{0, \ldots, 7\}, i \in N$ are sampled, the initial offsets are computed, and a random tree is selected as the initial basis. The nontree arcs are initially assumed to be slightly above the breakpoint. At each iteration the right and left slopes of the tree arcs differ, while for the nontree edges they are equal and their size depends on whether the offset was reached from above or from below.

The network, illustrated in Figure 5 of Importa and Sforza's paper, has 8 nodes and 13 arcs. We used a SPARC 2 computer (SunOS Release 4.1) to run two FORTRAN 77 versions of our algorithm. In one we computed a most negative cut in each iteration, in the other we stopped at the first negative cut. We found no difference in the quality of solutions produced by the two versions, and Table I describes the results of the total 200 runs. It shows that the optimal value (as found by Importa and Sforza) of 1,720 was reached in more than a quarter of the runs. The total cpu time for 100 runs of the most negative cut version was 2.8 seconds (the same order of time was required in to optimally solve this small problem by a branch-and-bound algorithm), with average of 22.7 iterations per run, while for the first negative cut it was 3.0 seconds with average of 47.7 iterations.

In the local search algorithm used in TRANSYT only cuts consisting of a single node are considered. We call this algorithm *restricted local search* and denote it **RLS**. In contrast, we call our algorithm, that considers all the possible cuts, *extended local search* and denote it **ELS**. As expected, **RLS** reaches a local optimum faster (on the average) than **ELS**. Therefore, to allow a fair comparison we executed **RLS** for the same length of time as required by the **ELS** to compute the 200 solutions of Table 1. **RLS** completed within this time 2400 iterations (each iteration computes a local optimum, starting from a random initial solution). The best solutions were obtained as follows: 1720 (34 times), 1731 (15), 1744 (21), 1752 (18), 1764 (18), 1769 (6), 1775 (25). We see that the optimal solution was obtained less times, but more solutions that are close to it were obtained than by **ELS**.

**Table I**
Distribution of Results for Importa and Sforza's Example

| Cost | Frequency | Cost | Frequency | Cost | Frequency |
|---|---|---|---|---|---|
| 1,720 | 55 | 1,911 | 4 | 2,022 | 2 |
| 1,775 | 19 | 1,912 | 1 | 2,025 | 2 |
| 1,782 | 26 | 1,937 | 5 | 2,052 | 1 |
| 1,793 | 4 | 1,947 | 12 | 2,074 | 2 |
| 1,799 | 3 | 1,987 | 1 | 2,078 | 1 |
| 1,827 | 8 | 1,989 | 3 | 2,115 | 2 |
| 1,840 | 2 | 1,993 | 5 | 2,117 | 1 |
| 1,860 | 9 | 2,008 | 1 | 2,150 | 1 |
| 1,883 | 1 | 2,009 | 1 | 2,168 | 1 |
| 1,907 | 20 | 2,013 | 2 | 2,202 | 1 |
| 1,909 | 2 | 2,021 | 1 | 2,227 | 1 |

To learn about the relative performance of the algorithms on larger problems we used the network of 34 nodes and 71 arcs described in Figure 1. This network represents a section of the road network of the center of Tel Aviv, in which all traffic signals have a common cycle time. For each arc we randomly sampled one of the 13 loss functions provided in Gartner's example. Again, there was no significant difference in the quality of the solutions produced by the first negative cut and the most negative cut versions. However, the most negative cut version required altogether about half the time required by the first negative cut, and therefore we applied it in the rest of our study.

While there is no guarantee to the quality of the solution obtained by the local search approach relative to the optimal one, some indication to the probability of improving it by continuing the search can be obtained by considering the number of times that the best solution value has been obtained. This parameter seemed to be correlated with the dispersion of the random minima obtained, which varied considerably among the problems considered (with different loss functions selected for each arc from the 13 possibilities on the same network). For example, in some situations the best solution appeared more than 100 times during 2,000 iterations, while in several others 20,000 iterations were required until for the first time the frequency of the best solution was at least 6 or 8.

We concluded that the following rule gives very good results for **ELS**: compute locally optimal solutions until the best solution value has been obtained 10 times. We checked this rule by running **ELS** for many more iterations (up to 100,000) but never found a better solution than the one attained by this criterion.

We randomly generated ten problems on the network of Figure 1, differing by the loss functions assigned to the edges. The details are given in the appendix. Each of these problems was solved by both **RLS** and **ELS**. We used the stopping rule of 10 appearances of the best solution with a limit of 100,000 iterations (from random initial solutions).

One could suggest a naive algorithm that just randomly samples solutions and finally selects the best. We checked the quality of this approach by examining the numerous initial solutions that we sampled during the execution of **RLS**. The best value, denoted Init* is given in Table II.

The average number of solution improvements until a locally optimal solution is obtained was about the same for all of the ten problems. It was 45 improvements for **RLS** and 115 improvements for **ELS**. The average time per iteration (obtaining a local optimum) was roughly 0.35 seconds for **RLS** and about 0.72 seconds for **ELS**. Thus, the

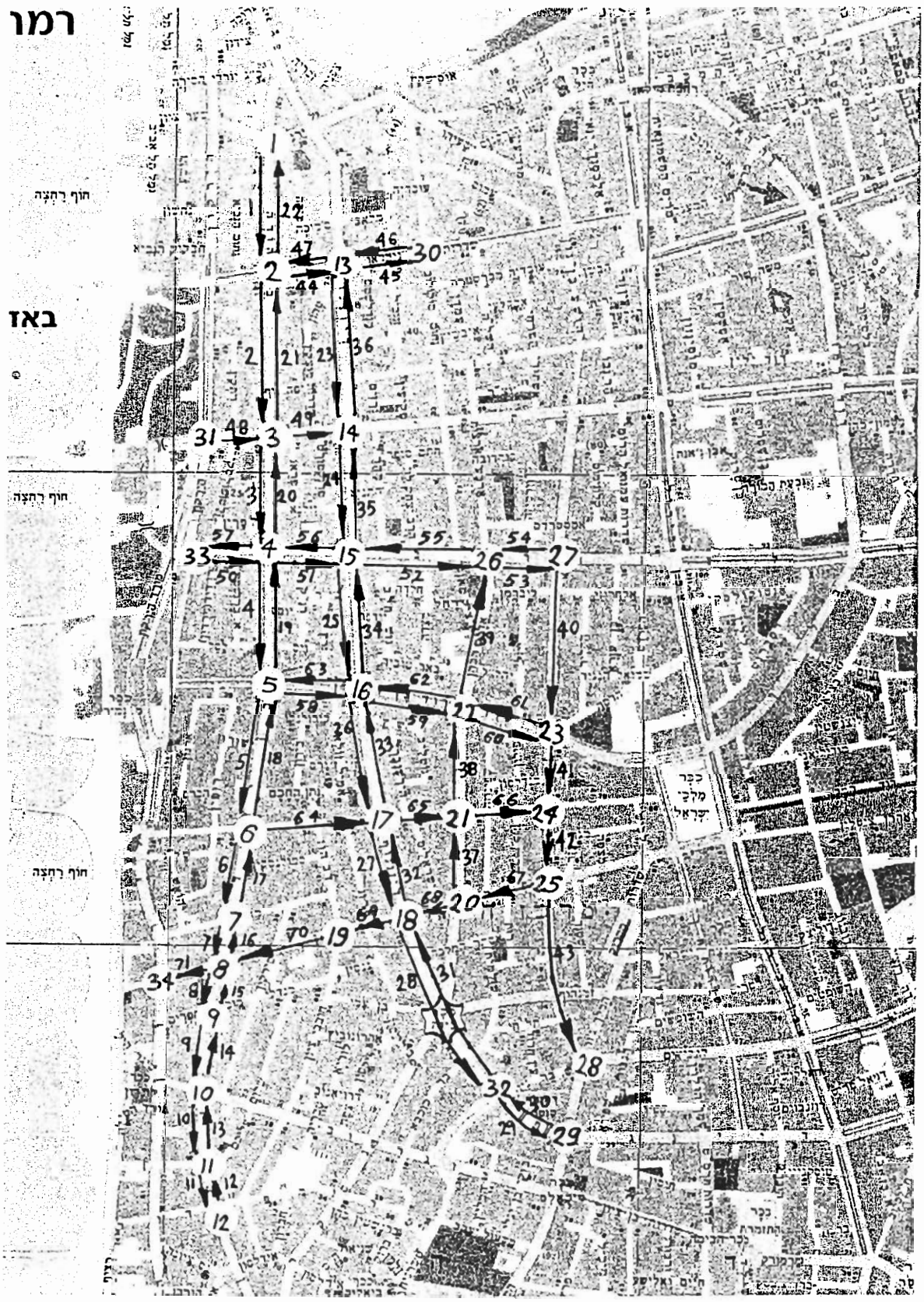

**Figure 1.** The Tel Aviv network.

running times can be computed from Table II by multiplying these figures by the number of iterations. The number of iterations are denoted ItRLS and ItELS.

Table II contains for each of the 10 test problems and each of **RLS** and **ELS**, the number of iterations, the average value of a local optimum (denoted by $\overline{RLS}$ and $\overline{ELS}$)

**Table II**
Comparison of RLS and ELS

| | Init* | ItRLS | $\overline{RLS}$ | RLS* | NRLS | ItELS | $\overline{ELS}$ | ELS* | NELS |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 11,708 | 100,000 | 11,363 | 9,793 | 1 | 570 | 10,201 | 9,580 | 10 |
| 2 | 10,435 | 100,000 | 10,705 | 9,311 | 5 | 2,246 | 9,527 | 8,970 | 10 |
| 3 | 11,121 | 100,000 | 11,726 | 10,291 | 2 | 2,812 | 10,663 | 10,001 | 10 |
| 4 | 11,172 | 100,000 | 11,061 | 9,893 | 3 | 1,497 | 9,879 | 9,506 | 10 |
| 5 | 11,289 | 100,000 | 11,009 | 9,782 | 5 | 7,722 | 10,165 | 9,495 | 10 |
| 6 | 11,939 | 90,063 | 11,016 | 9,932 | 10 | 60,517 | 10,272 | 9,649 | 10 |
| 7 | 11,827 | 100,000 | 11,050 | 9,929 | 3 | 11,019 | 10,284 | 9,722 | 10 |
| 8 | 10,765 | 49,144 | 10,246 | 9,117 | 10 | 4,059 | 9,416 | 8,846 | 10 |
| 9 | 12,040 | 100,000 | 10,666 | 9,179 | 6 | 121 | 9,449 | 9,000 | 10 |
| 10 | 12,023 | 100,000 | 10,771 | 9,464 | 6 | 380 | 9,615 | 9,213 | 10 |

and the best solution value (denoted by RLS* and ETS*). The number of appearances of the best solution (denoted NRLS and NELS) is also given. We see that in these test problems, quality of the solutions produced by **ELS** is superior to those produced by **RLS**, and in particular, the best **RLS** solution was never as good as the best **ELS** solution.

It is very interesting to note the differences among the 10 instances that we considered, as reflected in Table II. For example, in the ninth instance the best solution appeared in about 8% of the iterations (this figure was further verified by executing 100,000 iterations). In contrast, in the sixth instance, the best solution value, 9,649, appeared in only 10 out of 60,517 iterations! The second best solution value of 9,662 appeared 74 times. If we accept these proportions as estimates for the probability of obtaining these solutions, then there is a chance of about 0.24 that 9,662 will appear 10 times before the first appearance of 9,649.

Ben-Zvi (1991) studied traffic light synchronization under a novel loss function. The traffic through each node $i \in N$ was labeled according to its incoming and outgoing arcs, or in other words, by the green light at the node that permits this traffic to cross it. The flow through any given arc $(i, j) \in A$ was classified then by its labels at the nodes $i$ and $j$.

For each class associated with the arc $(i, j) \in A$ a loss function was defined as follows. First a loss function associated with each fixed travel velocity along $(i, j)$ was assumed to be given. Then, for each possible time that a car from a given class could enter $(i, j)$ at $i$ a minimum cost velocity was determined such that the car would reach $j$ during the green interval relevant to its type. Finally, the loss function at the arc was determined as the maximum (weighted) loss of any of its classes. The resulting loss functions in the study have two possible forms depending on whether the green interval at $i$ is longer or shorter than that at $j$. In one case the function is convex on part of the cycle and flat on its complement. In the other case it has two convex parts and two flat parts. The loss functions were approximated by piecewise linear functions with three or four breakpoints.

Our algorithm was applied to the subnetwork of Figure 1 induced by nodes 20–25. Thus it has six nodes and eight arcs (37, 38, 42, 42, 60, 61, 66, 67). The results are given in Table III where the objective values were normalized so that the optimal solution is 100. As we see, the optimal solution (as verified by solving an integer program) was reached 8 times out of 30 iterations, and nearly optimal solutions were reached 17 times. The algorithm was also applied to the whole network of Figure 1. The time needed for generating 50 locally optimal solution was 51 seconds on IBM 3090.

## 7. SUMMARY

We stated and proved a characterization of the locally optimal solutions in the network synchronization problem. It says that a solution is locally optimal if and only if it cannot be improved along binary directions. This characterization can be naturally used to produce a family of algorithms since it can be tested efficiently, and in the case of a negative outcome an improving binary direction can be found.

While this extension to network flow algorithm may be natural, none of the existing algorithms for network synchronization, motivated by applications from scheduling and traffic light synchronization, is based on a similar approach. In particular, the problem has been traditionally formulated in a different way, and only solutions of very small problems have been reported. We believe that our approach can extend substantially the size of network synchronization problems that can be successfully solved.

The success of our proposed method strongly depends on the type of loss functions under considerations. For example, the *traveling salesman problem* can be modeled as a network synchronization problem with period $|N|$, a large penalty on zero offsets, a loss equal to the arc length for a unit offset, and zero loss otherwise. We experimented with

**Table III**
Ben-Zvi's Results

| Objective | 100 | 112 | 114 | 117 | 119 | 172 | 191 | 220 | 298 | 345 |
|---|---|---|---|---|---|---|---|---|---|---|
| Frequency | 8 | 4 | 2 | 8 | 3 | 1 | 1 | 1 | 1 | 1 |

this problem and obtained discouraging results since the high penalty creates numerous locally optimal solutions. We believe, however, that for many real life instances that are naturally formulated as network synchronization problems our approach is capable of providing good solutions.

## ACKNOWLEDGMENT

The author is grateful to the referees for their comments and suggestions of various improvements to the paper.

## APPENDIX

**Table AI**
The Loss Function Attached to Each of the 71 Arcs

| d = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 250 | 196 | 120 | 54 | 50 | 100 | 160 | 211 |
| 2 | 150 | 160 | 290 | 398 | 350 | 295 | 244 | 194 |
| 3 | 120 | 168 | 214 | 260 | 306 | 350 | 320 | 173 |
| 4 | 131 | 141 | 272 | 379 | 331 | 276 | 225 | 175 |
| 5 | 136 | 188 | 240 | 291 | 282 | 168 | 49 | 82 |
| 6 | 295 | 266 | 225 | 180 | 133 | 88 | 100 | 209 |
| 7 | 100 | 224 | 298 | 250 | 230 | 212 | 185 | 146 |
| 8 | 180 | 233 | 334 | 441 | 387 | 359 | 358 | 230 |
| 9 | 138 | 170 | 217 | 267 | 260 | 187 | 120 | 100 |
| 10 | 100 | 224 | 208 | 250 | 230 | 212 | 185 | 146 |
| 11 | 176 | 223 | 273 | 328 | 384 | 345 | 207 | 137 |
| 12 | 252 | 144 | 79 | 114 | 161 | 207 | 250 | 290 |
| 13 | 105 | 54 | 116 | 240 | 298 | 262 | 210 | 156 |

## REFERENCES

ALLSOP, R. E. 1986. Selection of Offsets to Minimize Delay to Traffic in a Network Controlled by Fixed-time Signals. *Trans. Sci.* **2**, 1–13.

BEN-ZVI, R. 1991. Optimization of Traffic Signals Control. M.Sc. Thesis, Statistics Department, Tel Aviv University, Israel.

BURKARD, R. E. 1986. Optimal Schedules for Periodically Recurring Events. *Discr. Appl. Math.* **15**, 167–180.

ERVOLINA, T. R. AND T. MCCORMICK. 1993. Cancelling Most Helpful Cuts for Minimum Cost Network Flow. *Networks* **23**, 41–52.

FOULDS, L. R. 1986. TRANSYT Traffic Engineering Program Efficiency Improvement via Fibonacci Search. *Trans. Res.* **20A**, 331–335.

GAREY, M. R., AND D. S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Company, San Francisco.

GARTNER, N. H. 1972a. Constraining Relations Among Offsets in Synchronized Signal Networks. *Transp. Sci.* **6**, 88–93.

GARTNER, N. H. 1972b. Algorithms for dynamic road-traffic control. *Proc. 5th IFAC World Cong.*, Paris. 1–6.

GARTNER, N. H. AND J. D. C. LITTLE. 1975. Generalized Combination Method for Area Traffic Control. *Trans. Res. Record* **531**, 58–69.

GARTNER, N. H., J. D. C. LITTLE, AND H. GABBAY. 1975. Optimization of Traffic Signal Settings by Mixed-Integer Linear Programming. *Trans. Sci.* **9**, 321–363.

HASSIN, R. 1983. The Minimum Cost Flow Problem: A Unifying Approach to Dual Algorithms and a New Tree-search Algorithm. *Math. Prog.* **25**, 228–239.

HASSIN, R. 1992. Algorithms for the Minimum Cost Circulation Problem Based on Maximizing the Mean Improvement. *Opns. Res. Lett.* **12**, 227–233.

HILLIER, J. A. 1967. The Synchronization of Traffic Signals for Minimum Delay. *Trans. Sci.* **1**, 81–94.

IMPORTA, G., AND A. SFORZA. 1982. Optimal Offsets for Traffic Signal System in Urban Networks. *Trans. Res.* **16B**, 143–161.

KARAZANOV, A. V., AND S. T. MCKORMICK. 1993. Polynomial Methods for Separable Convex Optimization Linear Spaces with Applications to Circulations and Co-circulations in Networks. To appear in *SIAM J. Computing*, extended abstract in *Proc. Sixth SODA* 1995, 78–87.

LOVETSKII, S. E., AND I. I. MELAMED. 1987. Static Flows in Networks. *Aut Remot R* **48**, 1269–1291. Translated from *Avtomatika i Telemekhanika* **10**, 2–29, October 1987.

MCCORMICK, T., AND T. R. ERVOLINA. 1994. Computing Maximum Mean Cuts. *Discrete Appl. Math.* **52**, 53–70.

ROBERTSON, D. I. 1969. TRANSYT: A Traffic Network Study Tool. Road Research Laboratory Report LR 253, Crowthorne, Berkshire, England.

ROCKAFELLAR, R. T. 1984. Network Flows and Monotropic Optimization. John Wiley & Sons, New York.

SANDI C. 1986. On a Nonbasic Dual Method for the Transportation Problem. *Math. Prog. Study* **26**, 65–82.

SCHRIJVER, A. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York.

SERAFINI, P., AND W. UKOVICH. 1989a. A Mathematical Model for the Fixed-time Traffic Control Problem. *European J. Opnl. Res.* **42**, 152–165.

SERAFINI, P. AND W. UKOVICH. 1989b. A Mathematical Model for Periodic Scheduling Problems. *SIAM J. Discrete Math.* **2**, 550–581.

TAMIR, A. 1993. Complexity Results for the $p$-median Problem with Mutual Communication. *Opns. Res. Lett.* **14**, 79–84.

TSAY, H-S., AND K-T. WANG. 1989. Use of Three-Dimensional Conjugate Directions Search Method to Improve TRANSYT-7F Computational Efficiency. *Trans. Res. Record* **1225**, 116–129.

WONG, W. S., AND R. J. T. MORRIS. 1989. A New Approach to Choosing Initial Points in Local Search. *Info. Processing Lett.* **30**, 67–72.

**Table AII**
Details of the Ten Test Problems

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem number 1 | | | | | | | | | | | | | | | | | | | |
| 12 | 4 | 7 | 12 | 2 | 3 | 10 | 10 | 6 | 9 | 6 | 7 | 12 | 7 | 1 | 3 | 7 | 3 | 3 | 2 |
| 7 | 11 | 2 | 7 | 3 | 1 | 3 | 10 | 6 | 6 | 10 | 2 | 7 | 6 | 2 | 4 | 8 | 5 | 6 | 8 |
| 13 | 7 | 6 | 1 | 3 | 7 | 8 | 10 | 6 | 9 | 7 | 9 | 1 | 12 | 1 | 9 | 8 | 5 | 8 | 2 |
| 7 | 7 | 3 | 1 | 8 | 13 | 8 | 3 | 8 | 9 | 12 | | | | | | | | | |
| Problem number 2 | | | | | | | | | | | | | | | | | | | |
| 8 | 1 | 12 | 6 | 7 | 6 | 2 | 3 | 13 | 11 | 7 | 3 | 4 | 11 | 12 | 7 | 6 | 3 | 5 | 5 |
| 5 | 1 | 9 | 2 | 2 | 5 | 13 | 3 | 9 | 13 | 11 | 12 | 7 | 13 | 3 | 10 | 11 | 9 | 7 | 3 |
| 2 | 4 | 4 | 12 | 2 | 7 | 6 | 3 | 12 | 7 | 12 | 5 | 2 | 10 | 13 | 11 | 3 | 13 | 11 | 3 |
| 11 | 1 | 13 | 8 | 1 | 10 | 9 | 10 | 11 | 9 | 7 | | | | | | | | | |
| Problem number 3 | | | | | | | | | | | | | | | | | | | |
| 11 | 5 | 11 | 9 | 10 | 1 | 4 | 8 | 3 | 13 | 3 | 12 | 10 | 6 | 8 | 2 | 7 | 10 | 2 | 8 |
| 3 | 10 | 8 | 4 | 2 | 11 | 9 | 3 | 5 | 10 | 4 | 11 | 8 | 2 | 3 | 4 | 13 | 3 | 10 | 8 |
| 13 | 3 | 9 | 1 | 12 | 12 | 5 | 8 | 13 | 8 | 1 | 4 | 7 | 5 | 1 | 10 | 8 | 10 | 6 | 2 |
| 10 | 11 | 2 | 3 | 9 | 1 | 2 | 3 | 3 | 12 | 12 | | | | | | | | | |
| Problem number 4 | | | | | | | | | | | | | | | | | | | |
| 2 | 7 | 11 | 12 | 4 | 8 | 3 | 9 | 3 | 1 | 9 | 1 | 8 | 13 | 12 | 3 | 11 | 11 | 11 | 10 |
| 13 | 3 | 10 | 11 | 9 | 7 | 3 | 2 | 4 | 4 | 12 | 2 | 7 | 6 | 3 | 12 | 7 | 12 | 5 | 2 |
| 10 | 13 | 11 | 3 | 13 | 11 | 3 | 11 | 1 | 13 | 8 | 1 | 10 | 9 | 10 | 11 | 9 | 7 | 3 | 2 |
| 4 | 4 | 12 | 2 | 7 | 6 | 3 | 12 | 7 | 12 | 5 | | | | | | | | | |
| Problem number 5 | | | | | | | | | | | | | | | | | | | |
| 5 | 2 | 11 | 13 | 11 | 4 | 2 | 6 | 10 | 12 | 8 | 8 | 10 | 4 | 8 | 7 | 4 | 4 | 4 | 7 |
| 7 | 7 | 6 | 4 | 4 | 6 | 12 | 4 | 9 | 13 | 5 | 10 | 8 | 9 | 8 | 6 | 11 | 8 | 5 | 5 |
| 7 | 10 | 8 | 12 | 5 | 3 | 13 | 13 | 1 | 7 | 1 | 8 | 9 | 5 | 4 | 13 | 9 | 3 | 2 | 1 |
| 1 | 10 | 3 | 6 | 11 | 13 | 4 | 1 | 5 | 11 | 4 | | | | | | | | | |
| Problem number 6 | | | | | | | | | | | | | | | | | | | |
| 4 | 13 | 2 | 13 | 4 | 1 | 4 | 13 | 2 | 9 | 2 | 8 | 1 | 9 | 11 | 1 | 1 | 1 | 10 | 1 |
| 3 | 1 | 12 | 8 | 7 | 5 | 6 | 5 | 8 | 1 | 11 | 4 | 3 | 10 | 10 | 6 | 13 | 1 | 12 | 6 |
| 4 | 5 | 11 | 4 | 9 | 10 | 2 | 2 | 4 | 5 | 6 | 2 | 3 | 3 | 7 | 8 | 8 | 3 | 2 | 13 |
| 5 | 6 | 1 | 9 | 10 | 2 | 13 | 1 | 3 | 12 | 4 | | | | | | | | | |
| Problem number 7 | | | | | | | | | | | | | | | | | | | |
| 9 | 5 | 3 | 7 | 1 | 3 | 3 | 3 | 3 | 5 | 5 | 8 | 13 | 1 | 12 | 11 | 1 | 13 | 11 | 7 |
| 6 | 7 | 4 | 13 | 4 | 6 | 11 | 13 | 6 | 10 | 10 | 11 | 2 | 11 | 4 | 12 | 5 | 2 | 13 | 9 |
| 9 | 3 | 13 | 10 | 12 | 4 | 11 | 8 | 8 | 9 | 11 | 4 | 8 | 8 | 9 | 12 | 2 | 12 | 10 | 4 |
| 11 | 10 | 12 | 1 | 1 | 9 | 5 | 6 | 1 | 6 | 9 | | | | | | | | | |
| Problem number 8 | | | | | | | | | | | | | | | | | | | |
| 5 | 13 | 2 | 13 | 12 | 4 | 11 | 5 | 11 | 6 | 13 | 2 | 3 | 2 | 4 | 13 | 1 | 13 | 5 | 6 |
| 7 | 2 | 12 | 10 | 9 | 6 | 4 | 5 | 4 | 11 | 8 | 9 | 5 | 10 | 7 | 7 | 13 | 11 | 1 | 3 |
| 13 | 1 | 13 | 1 | 9 | 4 | 9 | 9 | 13 | 9 | 7 | 7 | 5 | 2 | 11 | 13 | 1 | 1 | 13 | 4 |
| 13 | 8 | 5 | 1 | 5 | 12 | 9 | 10 | 2 | 12 | 10 | | | | | | | | | |
| Problem number 9 | | | | | | | | | | | | | | | | | | | |
| 4 | 1 | 12 | 3 | 3 | 3 | 4 | 12 | 4 | 8 | 11 | 2 | 10 | 12 | 1 | 10 | 1 | 2 | 3 | 3 |
| 11 | 3 | 2 | 2 | 12 | 1 | 5 | 5 | 3 | 9 | 6 | 1 | 11 | 9 | 2 | 8 | 11 | 12 | 12 | 2 |
| 1 | 12 | 12 | 9 | 5 | 3 | 8 | 12 | 1 | 2 | 10 | 5 | 4 | 9 | 4 | 6 | 5 | 6 | 1 | 4 |
| 8 | 6 | 12 | 10 | 6 | 13 | 1 | 12 | 6 | 4 | 5 | | | | | | | | | |
| Problem number 10 | | | | | | | | | | | | | | | | | | | |
| 2 | 10 | 9 | 10 | 9 | 9 | 3 | 5 | 8 | 11 | 8 | 12 | 3 | 3 | 2 | 11 | 12 | 1 | 2 | 7 |
| 3 | 12 | 9 | 10 | 11 | 10 | 10 | 5 | 12 | 3 | 9 | 13 | 9 | 7 | 13 | 4 | 6 | 1 | 5 | 5 |
| 3 | 9 | 6 | 1 | 11 | 9 | 2 | 8 | 11 | 12 | 12 | 2 | 1 | 12 | 12 | 9 | 5 | 3 | 8 | 12 |
| 1 | 2 | 10 | 5 | 4 | 9 | 4 | 6 | 5 | 6 | 1 | | | | | | | | | |